CodeWorks 2nd Year Programs for TANDY MODEL III

The disks provided contains the following programs. The number after the filename indicates the size in bytes. Refer to the appropriate issue for full information on any program. Please do not remove our credit lines, they show where these programs come from, and that helps us.

********** Disk #1 **********

QKEY	BAS	10017	Issue 10 Quick search, relational data-base.
CWINDEX	DAT	15386	Issue 10 Works with Qkey. Index to all of CodeWorks.
QTEXT	BAS	2604	Issue 9 A basic editor text editor program.
QTEXT1	BAS	3359	Issue 12 Addressing update to Qtext.Bas
RANDEMO5	BAS	7808	Issue 12 Random file database series.
RANINDEX	BAS	3584	Issue 13 Indexing program for Randemo5.
TEST	MAP	145	Issue 11 Used with Randemo5.Bas to map files
TEST	SCN	273	Issue 11 Used with Randemo5.Bas to map screen.
TEST	DAT	128	Issue 12 Data file used with Randemo5.bas.
RENUM	BAS	4648	Issue 9 A basic renumber program.
RETIRE	BAS	1522	Issue 11 Calculates retirement income.
SCREEN	BAS	2569	Issue 11 Used as demo with Randemo5 series.
SMART	BAS	5348	Issue 9 Learns to play Tic-Tac-Toe.
SPEED	BAS	2898	Issue 12 Checks your subliminal vision
TICKET	BAS	6670	Issue 13 Makes short-run tickets to events.
TYPER	BAS	5308	Issue 12 Exercises printers, use as a letter writer.

********** Disk #2 **********

AMORT	BAS	7322	Issue 12 Loan Amortization table program.
BBASIC3	BAS	262	Issue 10 Basic graphs program.
CARD1	BAS	7194	Issue 12 Card.Bas updated with changes.
CARD	DAT	6	Issue 12 Empty data file for Card1.Bas
CROSSLEY	BAS	511	Issue 12 Short addressing program.
DAY	BAS	801	Issue 13 Calculates day of week.
DAYDATE	BAS	2437	Issue 13 Calculates day of week/days between dates.
DCHECK	BAS	2068	Issue 13 Data checker program.
DIFF	BAS	2674	Issue 11 Finds differences between programs.
ROLL1	BAS	572	Issue 11 Used with Diff.Bas as a demo.
ROLL2	BAS	586	Issue 11 Used with Diff.Bas as a demo.
FRACT	BAS	1740	Issue 12 Makes fractions from decimals.
GRAPH2	BAS	1889	Issue 11 Puts more than one graph on screen.
INDEX	BAS	2048	Issue 13 Demo indexing program. See randemo5.
IRA	BAS	1670	Issue 11 Compute how your IRA grows.
KEYCODE	BAS	256	Issue 11 Finds what code your keys return.
LEAP	BAS	421	Issue 13 Finds if a year is a leap year.
LOTTO	BAS	1685	Issue 10 Finds winners in Lotto, fast.
MCARD	BAS	933	Issue 8 Used with mini-card-data system.
FORM	BAS	4297	Issue 8 Used with mini-card-data system.
RCARD	BAS	4957	Issue 8 Used with mini-card-data system.
MENUMAKE	BAS	3265	Issue 13 Program that makes menu programs.
NAMES	BAS	1931	Issue 12 Program to switch first/last names.
NFL87	BAS	6875	Issue 13 NFL prediction program 1987-88
STAT	DAT	1	Issue 13 Empty file used with Stat87.Bas.
STAT87	BAS	5802	Issue 13 Maintains stats for NLF87.Bas.
PAGES	BAS	761	Issue 12 Figures how pages work together.
PIPE	BAS	934	Issue 12 Calculates scales of Organ pipes.
POINT	BAS	1013	Issue 10 Demos using pointers to strings.
POKER	BAS	18218	Issue 8 5 player, open on any pair draw poker.

CodeWorks 2nd Year Programs for PC/MS-DOS GW-BASIC

The disk provided contains the following programs. The number after the filename indicates the size in bytes. Refer to the appropriate issue for full information on any program. Please do not remove our credit lines, they show where these programs come from, and that helps us.

ADDRESSR	BAS	7062	Issue 11 Multipurpose addressing program.
AMORT	BAS	7322	Issue 12 Loan Amortization table program.
BBASIC3	BAS	262	Issue 10 Basic graphs program.
CARD1	BAS	7194	Issue 12 Card.Bas updated with changes.
CARD	DAT	6	Issue 12 Empty data file for Card1.Bas
CROSSLEY	BAS	511	Issue 12 Short addressing program.
DAY	BAS	801	Issue 13 Calculates day of week.
DAYDATE	BAS	2437	Issue 13 Calculates day of week/days between dates.
DCHECK	BAS	2068	Issue 13 Data checker program.
DIFF	BAS	2674	Issue 11 Finds differences between programs.
ROLL1	BAS	572	Issue 11 Used with Diff.Bas as a demo.
ROLL2	BAS	586	Issue 11 Used with Diff.Bas as a demo.
FRACT	BAS	1740	Issue 12 Makes fractions from decimals.
GRAPH2	BAS	1889	Issue 11 Puts more than one graph on screen.
INDEX	BAS	2048	Issue 13 Demo indexing program. See randemo5.
IRA	BAS	1670	Issue 11 Compute how your IRA grows.
KEYCODE	BAS	256	Issue 11 Finds what code your keys return.
LEAP	BAS	421	Issue 13 Finds if a year is a leap year.
LOTTO	BAS	1685	Issue 10 Finds winners in Lotto, fast.
MCARD	BAS	933	Issue 8 Used with mini-card-data system.
FORM	BAS	4297	Issue 8 Used with mini-card-data system.
RCARD	BAS	4957	Issue 8 Used with mini-card-data system.
MENUMAKE	BAS	3265	Issue 13 Program that makes menu programs.
NAMES	BAS	1931	Issue 12 Program to switch first/last names.
NFL87	BAS	6875	Issue 13 NFL prediction program 1987-88
STAT	DAT	1	Issue 13 Empty file used with Stat87.Bas.
STAT87	BAS	5802	Issue 13 Maintains stats for NLF87.Bas.
PAGES	BAS	761	Issue 12 Figures how pages work together.
PIPE	BAS	934	Issue 12 Calculates scales of Organ pipes.
POINT	BAS	1013	Issue 10 Demos using pointers to strings.
POKER	BAS	18218	Issue 8 5 player, open on any pair draw poker.
POKER7	BAS	18954	Issue 9 7 player, jacks or better, draw poker.
QKEY	BAS	10017	Issue 10 Quick search, relational data-base.
CWINDEX	DAT	15386	Issue 10 Works with Qkey. Index to all of CodeWorks.
QTEXT	BAS	2604	Issue 9 A basic editor text editor program.
QTEXT1	BAS	3359	Issue 12 Addressing update to Qtext.Bas
RANDEMO5	BAS	7808	Issue 12 Random file database series.
RANINDEX	BAS	3584	Issue 13 Indexing program for Randemo5.
TEST	MAP	145	Issue 11 Used with Randemo5.Bas to map files
TEST	SCN	273	Issue 11 Used with Randemo5.Bas to map screen.
RENUM	BAS	4648	Issue 9 A basic renumber program.
RETIRE	BAS	1522	Issue 11 Calculates retirement income.
STACK	BAS	2229	Issue 10 Used as demo with Randemo5 series.
SCREEN	BAS	2569	Issue 11 Used as demo with Randemo5 series.
SMART	BAS	5348	Issue 9 Learns to play Tic-Tac-Toe.
SPEED	BAS	2898	Issue 12 Checks your subliminal vision
TICKET	BAS	6670	Issue 13 Makes short-run tickets to events.
TYPER	BAS	5308	Issue 12 Exercises printers, use as a letter writer.

3838 South Warner Street, Tacoma, Washington 98409 (206) 475-2219

Dear Friend,

CodeWorks is a bimonthly magazine devoted to problem solving in BASIC programming. All programs in CodeWorks are described in fine detail, showing both how and why the program was written as it was. The programs stand alone and are easily adaptable and useful. We support Tandy III/IV, CP/M and all PC machines.

Here is an opportunity for you to get a current subscription or sets of back issues and diskettes containing as many as 40 programs at HALF PRICE!

That's right. Order any set of issues for any year and we will include the diskette for that year at only $10 (instead of the regular $20.) Any one of the programs on these diskettes could easily be worth this price!

Hurry. . . we don't expect our back issue sets to last too long with this offer, so ORDER YOURS TODAY. (See reverse side for information on diskettes.)

ORDER FORM

❑ Year 3 issues	$24.95	❑ 3rd year diskette	$20.00
❑ Year 2 issues	$24.95	❑ 2nd year diskette	$20.00
❑ Year 1 issues	$24.95	❑ 1st year diskette	$20.00

❑ Fourth year subscription (starts in Nov 88) $24.95

Deduct $10 for each diskette ordered with a year's issues. Please specify computer type below:

My computer is a: _____

❑ Check, MO is enclosed
❑ Charge to VISA/MC #_____ Exp. _____

Name _____

Address _____

City,State,Zip _____

Return this form to: CodeWorks, 3838 S. Warner St. Tacoma, WA 98409

CodeWorks

Issue 9 January/February 1987

CONTENTS

CODEWORKS

Issue 9 **Jan/Feb 1987**

Editor/Publisher
Irv Schmidt
Associate Editor
Terry R. Dettmann
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Advisor
Al Mashburn

Editor's Notes

Well, week thirteen of the NFL just passed and our projection program hit less than 50 percent for the first time. Dismal, to say the least. That knocks our overall average for the season so far to something below 70 percent. At this point it looks like the Bears and Broncos may be fighting it out. The birds seem to have flown the coop. Except for the Eagles. Every time we pick them to win they lose and vice versa.

By our next issue it will all be over and we will do a little recap on what happened - which teams we picked the best, worst, etc. We are also thinking of making a few changes to it for next season, like putting in a little more user interaction, where you can add your "gut feel" to the base that the program already gives you.

Thank you all for the vote of confidence. Renewals are well over the 50 percent mark and still coming in (I'm writing this on the 3rd of December.) Our goal is to get as close to 100 percent renewal as possible and then double the number of subscribers by this time next year. (Don't laugh, goals are *supposed* to be tough.) You can help by having a free sample copy sent to a friend.

When we first tabulated the various computers you use, about nine months ago, MSDOS was sitting at about 18 percent. It must be catching on. Our current count shows almost half of you have MSDOS now. Perhaps it's due to the reduction in prices and the introduction of the various PC work-alikes. The other half is made up of earlier Tandy models (I,II,III,IV) and the CP/M models. We will continue to do our best to cover the BASICs used by all of these machines.

We were quite surprised at the number of diskettes you ordered. We had ordered a few hundred blank diskettes, thinking that there would probably be some left over. They only lasted for two weeks. We had to put in another crash order for more. One MSDOS disk could hold the entire number of programs we published last year. Most of the other computers required two diskettes and the old work-horse Tandy Model I took three.

Since we are supposedly a problem solving type publication, we would like to recommend a new book. It is published by Microsoft Press, and is called "Programmers at Work" written by Susan Lammers, and consists of interviews with the "big guns" who wrote dBASE, PFS:, VisiCalc, Lotus 1-2-3, CP/M, Multiplan, among others, and especially Microsoft BASIC, with Bill Gates himself. It's a mixture of arrogance, ego, humility and some darn good insight to how those guys think. And I don't think that one of them mentioned using a flow chart. If you don't think the book is worth the $14.95 price, then check it out at your local library.

And speaking of programming ideas, it seems that of late some of our best ideas are coming from you. I think that it was mentioned before that we recently wrote five or six utility programs, all of which were from reader suggestions. Renum, in this issue, was suggested some time ago by a reader in Florida. In the meantime, we are working on two or three more which were suggested in letters or phone calls. Thank you for all those good leads. If the problem is real to you it probably is to many others as well, so give us a crack at it and we'll do our best.

Thank you for sending along all those nice remarks with your renewals - it always makes our day. And thanks for making the first year of CodeWorks as interesting to us as we hope it was for you.

WE WISH YOU ALL A MERRY CHRISTMAS and HAPPY NEW YEAR!

Forum

An Open Forum for Questions & Comments

In as much as I am a self employed independent programmer and program exclusively in compiled BASIC, I shouldn't really need your magazine. Wrong! I definitely need it. I am not a narrow minded programmer in as much as I always look for a better mouse trap.

Granted, for me the programs published are crude and elementary, but, (there are) new and interesting ways other people deal with routines. I learned a long time ago that my way is not the only way, nor the best. I'm sure I could teach a few tricks myself.

Actually it feels good to read a publication dedicated to BASIC programming as it helps mend the bruised ego one gets from so called higher level programmers. I'm sure you know the feeling. "What! you still fooling around with BASIC. Ha, Ha, Ho, Ho." Your publication is most welcome at $24.95 as I have purchased many books in the past for more and got less.

Rodney Starcher
Akron, OH

Elementary maybe, crude - never! Perhaps you could enlighten the rest of us with some of those tricks you refer to.

You all seem to have some clever ideas. Reminds me of the BOSS group for the Sinclair several years ago - they had some real winners on that little machine. Made 16K do what 64 and 640 do now! Am using ZBASIC. Fantastic. But how do we defeat the 64K barrier in MSDOS?

L. M. Stern
Bonifay, FL

We had a Sanyo 555 here for awhile. It's an MSDOS machine that gives you the entire remainder of memory when you get into BASIC. The downside of that was that you couldn't run very many applications programs that would run on most other PC clones. Using the DEF SEG statement in GW BASIC you can address different 64K segments in high memory and you can get to and from them by using PEEK, POKE and USR. We believe you would still need to swap the 64K segments to use them effectively, however. Maybe a BASIC compiler would do the dirty work for you. Is there any way to use the compiler to allocate more than the normal 64K in BASIC after the program is compiled? We'll add this to our list of things to do and let you know.

...Wouldn't miss an issue if I could help it. Some of us who are interested in finances and investment could use a tutorial showing how to construct a moving average as well as articles relating to property management. Keep up the good work.

A. H. Kightlinger
Chico, CA

We see property management programs through a haze right now, but the moving averages idea is clearer and should be a reality sometime this year.

Re the NFL programs, specifically NFLSTAT.Bas, I have incorporated the following changes in lines 890-930 so as to make editing easier. This change will re-use the present value if there is no change, just by pressing ENTER.

```
890 INPUT"Enter correct team number --";Z$;
        IF Z$="" THEN A(I,1)=VAL(Z$)
```

Change the portion after the semicolon in lines 890 through 930 as above.

Larry Abbott
Wyomissing, PA

As we discussed, (see Issue 8, Editor's Notes), adding your line 625 to NFLSTAT.Bas will prevent the inadvertent destruction of weekly data already on file, but it will not provide an escape hatch for the operator who may have wandered into the 1-Update.. area by mistake.

Since I have two young sons whom I'm trying to interest in gathering the NFL data and punching it, I felt that some mild form of error trapping was in order. Thus, for line 630, I made this change:

```
630 IF W=L1/28 THEN PRINT"The file appears
to be updated through that week. PRESS ENTER
for menu.";:INPUT X:GOTO 420
```

That code seems to retain the author's intent and provides the escape.

Additionally, because I wanted error trapping for both of the "write" functions, i.e., 1-Update and 2-Edit, I also added a line 795 as follows:

```
795 PRINT"To exit this routine, enter 30 both for
```

team and week number."
30 seemed appropriate because it not only exceeds both the number of teams and the number of weeks in the program, but also, I understand, once was used by reporters to indicate "END."

Regardless, I prefer the absence of commercial "bells and whistles" in CodeWorks programs. Part of the fun is to find areas where improvements may be needed to suit our needs...

John R. Miller
Anderson, SC

Thanks for the changes. Although we are still above the 66% or so that we thought the program would do. We're working on ways to make it do better. As I write this, week 13 is just coming to a close and sports news compared to our weekly sheet is not very encouraging. Like most programs of this sort, you can keep looking for that elusive "something" that will make it work better. We just haven't found it yet, and are not even certain that it exists.

Perhaps you could suggest a method of connecting my PC compatible computer to my Tandy Model III for transferring ASCII files without going through the telephone. I have a modem and program for the Model III and am getting a modem and program for the clone.

Charles B. Stevenson
Wilmington, CA

We do it here on a regular basis. Naturally, both machines must have an RS-232 (for direct connection, the modems are not needed.) Make sure your cable connecting the two machines has a null-modem adapter (or pins 2 and 3 swapped on one end only will do the same as a null-modem adapter.)

Any good communications program at both ends will work. If the data files you transfer are too large for the buffer in either machine you might try to find the option in the communications program that flows data directly to the disk via the buffer (if there is one.) Otherwise, you may need to break the file into segments before transmitting them and then append them once they are in the destination machine.

*If you transfer BASIC programs be aware of the differences between GW BASIC and the Model III BASIC. Some of the most obvious are PRINT@ and LOCATE, RND and INT(RND(0)*X), RANDOMIZE is required in GW BASIC, not in Model III. SWAP is available in GW BASIC, not on Model III. Spacing around key words is required in GW BASIC, not in Model III. MODulo is available with GW BASIC, not in Model III. Clearing string space is necessary on Model III, CLEAR does something entirely different in GW BASIC, which dynamically clears string space. Hope this helps.*

Thanks for one of the best publications I have read to date. My problem is I use MS-DOS, GW BASIC, Heath Z100. When I run NFL86 I get ... Input past end in 560. Can you help?

Roderick Legnon
San Diego, CA

I am sure that you must be aware that you must have the statistics file available and that it must be updated at least through week three before NFL86.Bas can work. If the STAT.Dat file is ok and you still have the problem you describe, then try exchanging lines 540 and 550 in NFL86.Bas. In other words, put the EOF check inside the J loop. I found one other computer type that wants it that way (Model II Tandy). In general, Input Past End errors occur when you try to read more data than exists, i.e., read 10 items when there are only nine.

I really became enthused when I received Issue 8 and saw MCARD.Bas. I thought this would be the answer to my problem of several small card files in which I had changed CARD.Bas to print out a different number of items (fields) for each different program. I thought I would be able to call up the xxx.dat and set the format for printout. However, to my dismay, if my program data was not labeled CARD.DAT I could not use it...

...What changes could I make to the program to be able to simply use different xxx.dat files and still be able to change the print format and print them...

Emory Howell
Tyler, TX

The Mcard system was designed specifically to do what you say you can't do. To summarize, you use FORM.Bas to create a report definition file that will tell the report generator, RCARD.Bas, how to print out the data in your xxx.dat file. Notice that in line 270 of FORM.Bas you can give any name you like to the report format file. If you do not name it, line 280 will name it CARD.PRT by default. In the program RCARD.Bas in line 360 you are asked for the name of the data file. Here again, you can specify any valid data file name. If you don't, the file name defaults to CARD.DAT. Line 370 again lets you specify any valid name or it defaults to CARD.PRT. The report format you choose in line 370 must conform to the data file you selected in line 360.

Irv

Renum.Bas

A selective BASIC renumber program

Staff Project. BASIC's RENUM is fast and easy to use, but it has some limitations. Although slower, this program will let you renumber any or all of a program and keep all references intact. It can be compiled for greater speed.

BASIC's renumber facility is nice - and it's quick, too. But there are a few things it will not do that can be annoying at times.

For one, it will renumber from any line in the program to the end, but will not change line references in the earlier (un-renumbered) part of the program. It won't let you renumber a section *within* the program at all. Its syntax is: *newline, startline, increment*. If increment is not specified, it uses the standard 10. There is no provision for *stopline*, and the only part of the program renumbered is what is specified.

Renum.Bas is our answer to this dilemma. It's in BASIC. It's slower. You must save the program to renumber in ASCII first. But it will do what you want it to, *and take care of all the references in the other parts of the program*. With Renum.Bas, you can elect to renumber the entire program, or any section within it. If you like to keep your subroutines at 1000, 2000, 3000, etc., and happen to run out of space between 2000 and 3000 because you have all your lines bunched up near 2000, you can renumber just that section and give yourself some more space. While it renumbers, this program will look at, and change if necessary, all line number references into and out of the area you are renumbering. It also makes a check first to see if the numbers and increment will fit within the area you specify, and if not, will tell you the maximum increment to use to make it fit.

The Program Design

The idea to do this type of a program came from a reader in Florida, who suggested it may be a nice thing to have as an additional utility (along with the regular BASIC renumber program.) We thought so too, but at first it seemed like a real tough problem. Terry (as usual) suggested simply reading in the program line by line and making the appropriate changes to the line number references. If they didn't need to be changed, leave them alone, else exchange. In answer to where the new numbers come from, he suggested creating two parallel arrays, one with the old numbers and the other with new numbers calculated from the user's answers to input questions about where to start, stop and increment.

With that as a starting point, we sat down and took a good look at some of the details we would need to consider. It became obvious that we would need to read the program from disk and strip off the line numbers and put them into an array. We called that array A().

After the A array was filled with line numbers, we needed user input to determine where to start and stop renumbering. We then created a B() array which contained zeros for any corresponding index number in the A array that didn't need to be changed, but contained the new number if it did.

Now all we needed to do was to read the target (program to be renumbered) again, one line at a time, check all line numbers in it and find them in the A array and then check the B array. If the B array had a zero in it, leave the original number be, otherwise, swap the number with what was in the B array. This had to include not only the line number itself, but any line number reference within the line.

But how could we tell the difference between a genuine line number reference and an ordinary number? How could you tell the difference, for example, between GOTO 370 and A=370?

It turns out that there are only a handful of BASIC statements that can be followed by a line number reference. The most obvious of these are: GOTO, GOSUB, THEN and ELSE. These, plus the ON GOTO and ON GOSUB, covered at least 95 percent or more of the cases. There are others, like: LIST and RUN, but we decided not to include them in order to keep the program as simple as possible. Actually, RUN would not be difficult to add to the program, but LIST, like ON GOTO and ON GOSUB, would require special handling because it

can be written in several ways: LIST -#### or LIST ####- or even, LIST ####-####. The ON GOTO and ON GOSUB problem was that there were numbers following numbers which may have to be changed, but only one word GOTO or GOSUB. In this case, we would need to detect the keyword "ON" followed by a GOTO or GOSUB, and then fall into a subroutine that would look for numbers followed by commas.

The next question that came up was how to parse the line. You can't just pull three characters out of the middle of a line and replace them with four characters. It won't work - you lose a character that way. You cannot make a string longer than it already is except by concatenation.

The first way we considered parsing the line was to search (using INSTR) for the occurrence of one of the keywords. Then make a temporary LEFT$ of that much of the line and a temporary RIGHT$ of the remainder of the line. Then the number in question would be at the left end of the RIGHT$ portion, and could be stripped off and replaced if necessary. That turned out to be very sloppy. First of all, searching for the keyword with INSTR gets you to the *beginning* of the keyword, not the end where you really want to be. So further manipulation was necessary to get to the end of the keyword. Next was the problem of creating all those temporary strings. And what if there were more than one line number reference in one line of code, as in: 300 IF A=5 THEN GOSUB 1000:GOSUB 2000:GOTO 100? In that case we would be taking the line apart and putting it back together three times. It was apparent that those strings would slow the program down and clutter memory rather quickly.

The method we finally settled on was to read the line one character at a time and build a new temporary string (C$ in the program). At the same time that C$ was being built, we would look at its right four characters every time a new character was added to the string. When the right four characters matched the keywords we were looking for, the loop counter stepping down the loop to create C$ would be exactly on the last character of the keyword, and using the VAL function in conjunction with the MID$ function we could examine the number following the keyword. The number could then be compared to all the numbers in the A array and if there was a counterpart for it in the B array, the B array number would be added to C$. If the number had no counterpart in the B array the original number could be put back into C$. By this time, we would know what the number was and how long (how many characters) it was. That length number would then be used to skip ahead in the line in question and continue building C$. This way we would only need to make one pass through the line and make all the adjustments necessary. Well, almost. We found it easier to handle the line number at the beginning of the line separately.

Another question which arose was the unreferenced line number. How would we handle a reference to a line that does not exist? We decided to do what the BASIC renumber program does, simply print out "Undefined *line* in *line*, and then place a question mark at that spot in the line.

Program Description

The following discussion will provide a rather detailed account of the program by line numbers. In it, we will refer to the "target" file or program (F$) and the "output" file or program (F1$). The target file is the file on disk which we wish to renumber. The output file will be the renumbered file we create and put on disk with the extension .NEW. The target program should have been saved in ASCII format, and should have the extension .BAS (users of machines using the /BAS extension should make the appropriate change wherever the period is encountered in the program.)

The first few remark lines need no explanation. Line 140 is a CLEAR XXXX line for those BASIC's requiring string space to be cleared in advance. Remark this line if you don't need it.

Line 150 sets the maximum number of lines of code the target program can contain. Six hundred seems like a reasonable number, but if you have longer programs (and have space in memory for the arrays) change it to a number large enough to accommodate your program.

Line 160 dimensions the two arrays, A() and B(), to the number set in the previous line.

Lines 170 through 330 are heading and description lines. Line 340 asks for the name of the file that will be renumbered and calls it F$. Variable SN in line 370 sets the starting line number to be renumbered. The ending line number is established in line 400 and becomes EN. The increment between lines is input in line 420 and becomes variable IC. Line 440 simply prints a message on the screen saying that the file is being renumbered.

The section of code from lines 470 through 540 reads the target program, finds the line number of each line, converts the line number (which is in string form) to an integer and puts it into the A()

array. Line 470 opens the file for input. The loop that starts at line 480 reads lines from the target program file from 1 to NL (NL was set in line 150.) In line 490, if we reach the end of the file (EOF) prior to reaching NL, we jump out of the loop because all lines must have been input. Line 500 "line inputs" the lines from the target file through buffer #1 and assigns the line to A$. Line 510 looks at each line as it comes in and searches for the first space from the left end of the line. BASIC demands that there be a space after the line number. The variable S will tell us what character position from the left of the string that space occupies. Line 520 then assigns the actual line number to variable LN by taking the VAL (value) of the left end of A$ up to the space (S). By taking the VAL of the number, we have changed it from a string to an integer. The remainder of A$ at this point is discarded - all we want right now is the line number.

You may ask why we don't keep the entire line and operate on it as long as we have it. It's because we don't yet have any idea of what the new line numbers will be or which ones need to be changed. Line 530 now assigns LN (the line number) to the A array. When the loop between lines 480 and 540 has reached EOF, the A array will contain the integer line numbers of the target program. Because the loop counter runs one number in advance of the actual number of lines read, we make N (the number of lines in the target program) equal to the loop counter less one. This happens in line 560. From here on then, we will not need to detect EOF or use IF statements to tell how long the file is, we simply loop or read from one to N.

In line 550 we close file buffer number 1 because we are done with it for now. In lines 570 and 580 we set some limits to the lines to be renumbered. The first item in the A array, A(1), is the lowest (first) line number of the target program. Back in lines 360 and 370 we claimed that pressing ENTER would start the renumber process at the first line. In line 570 we check to see if the start number (SN) is less than A(1), (pressing ENTER will leave SN at zero), and if it is less, then we make SN equal to A(1). If the number we entered into SN is equal or greater than A(1) then we use it and ignore line 570. Unless we specified an ending number in lines 390 and 400, line 580 will now make the ending number equal to 65500, which is almost the highest line number BASIC can use (actually, it's 65529.) This only comes into play when you want to renumber the entire program and use large increments between line numbers.

We now have the start number, the ending number and the increment between numbers, SN, EN, and IC, respectively. In lines 610 through 640 we are going to build another array (the B array), using the same index number (I) that the A array has. If the number in the A array falls between the start and ending numbers that we have specified, we will put the new number into the B array at that index location. It all starts in line 610, where we make variable SU equal to SN. Variable SU will be used temporarily and then dumped. The reason we need it is because we don't want to change the start number (SN) - we need it later. But we need to increment the start number by the increment (IC) each time through the loop. So rather than destroy the value in SN, we assign SU to be equal to SN and use it instead. The loop from 620 through 640 will read each value in the A array. If that value is larger than the start number and less than the ending number, the corresponding B array value will be determined by the start number (now SU) plus the increment. After this, SU is again increased by the increment for the next time around. Variable CT at the end of line 630 is an accumulating counter. It tells how many lines will be changed. We will need this a little later, when we try to see if that number of lines with the given increment will fit in the space without running into existing line numbers. When we are done with this loop, we still have the original start number (SN) intact, and for every line number in the A array that needs to be changed, there will be a number (properly incremented) in the B array at the same index location.

Lines 670 through 700 are used to see if the number of lines we want to change, and the space between them (the increment), will fit in the allowed space without crashing into the ending number (EN). Line 670 says that if the starting number (SN), added to the number of lines (CT) multiplied by the increment (IC) is less than the ending number, to go on with the program because they will fit. If the number is equal to or larger than the ending number we know they cannot fit, and so we print a message to that effect, and in line 680 we calculate what the maximum increment is that *would* fit. If we find that what we would like to do will not work, we are stuck with the B array full of bad numbers. So in line 690 we ask for input of a new increment, then in line 700 we go through the B array and clear it out. Next, we go back to line 610 where we re-define SU and clear CT (the number of lines to change counter.) If we didn't clear CT, the number of lines to change would double because CT is an accumulating counter.

Either way, if the lines fit or not, we eventually get to line 710. This line prints the index number (I)

and the corresponding values in the A and B arrays on the screen. It should look something like figure 1. This line is very useful in checking out the program to see if what you wanted to happen really did. When you are satisfied that the code up to this point is working correctly, you can remark or remove this line. If you remove it, however, you must change the reference to it in line 670 to line 740.

Summary - so far

At this point the target program is still on disk. We have read one line of it at a time and stripped off the line numbers and put them into the A array. We have the information about where to start and end renumbering and the increment, and we know that what we want to do can be done. The B array has been established and contains the numbers that will replace the numbers in the A array when we renumber.

Find and Replace

What we need to do now is to read the target program again, one line at a time, find any line number references that need to be changed, change them and write them back out to a new file. We could build the new file in memory and then put it back where it came from with the same name, but that would give you only one shot at getting it right. It's a little more comforting to know that the original file is still there. So we will leave it alone (except to read it) and build a new, renumbered, file. For that, we need a new filename. We get it in line 740, where we take the old filename and strip off the last four character positions (the period and BAS.) Then we attach a new extension to the old name (.NEW) and call it F1$. This is done by finding the length of the old filename, then taking the MID$ (mid string) of the old filename starting at position one to the length less four. That strips the old extension from the filename. We call that much of it TP$ (temporary string). We then make the new filename by taking TP$ and adding .NEW to it.

Now that we have our new output filename, we open it for output, using file buffer #2, in line 770. Buffer 2 is used here because we are going to have two file buffers open at the same time, one to read from the old file and one to write to the new file. Sequential files cannot be opened for both read and write at the same time.

The Main Loop

The main processing loop (I) for this program is from lines 790 through 1000. Inside this loop there is a secondary loop (Q) from lines 870 through 960. Let's run through the main loop first to see what it does and leave the secondary Q loop for later.

At line 800 we read one line from the target program on disk. Notice that we are not accumulating these lines in memory. In fact, the only program in memory is the renumber program itself (and the two arrays we built earlier.) The LINE INPUT #1, A$ will read the first line in the target program *up to the carriage return at the end of the line.* Every line of BASIC code ends with a carriage return.

Line 810 is a checkout line that can be remarked or removed when the program is running. It prints the line just input on the screen so you can see what it looks like before any changes take place.

Line 820 finds the length of the line just input and adds one to it. Without adding one, lines that end with a line number reference (i.e., 100 GOTO 1025) would end up with the last character missing.

As we did earlier, S in line 830 finds the first space from the left end of the line. In line 840, LN again becomes the integer value of the line number. Now that we have the line number in integer form, we can check through the A array to see if it needs to be changed. This occurs in line 850. Here we are checking only for the line number itself, not any line number references within the line. Line 850 says that if the line number is greater than our starting number and is also less than our ending number, then it needs to be changed, otherwise go on. The change takes place by making another temporary string (T$) which is the MID$ of A$, starting at position S (the first blank from the left end of the line) and going to the length of the line less whatever number of characters S was. Then, further down the line, we build a new A$ by starting it with the string value of what was in the B array (at index I) and adding T$ (the rest of the line) back to it.

In line 860 we again find the length of A$. We did this earlier, why do it again? Because we could have replaced a line number two digits long with one four or five digits long, or the other way around, and the Q loop which follows must know exactly how long the line is.

For now, lets just say that the Q loop will find line number references within A$ and change them. Let's finish with the I loop first. Assuming that the Q loop has done its job, line 970 will print the new A$, through buffer #2, to the new file. Notice that the Q loop will have changed the name

from A$ to C$. In line 980 we have another checkout line. It prints the changed line directly below the original line we printed to the screen earlier so you can see what happened to it. Remark or delete this line after you are satisfied with the way it works. In line 990 we set C$ back to a null string and clear a couple of flags that may have been set inside the Q loop.

After all the lines in the target program have been checked and written back to the output file, we close all files with the CLOSE statement in line 1010, print the filenames on the screen and end the program.

The Q Loop

To effectively renumber any or all of a program, you must look at every line in the program because there may be references into or out of the renumbered area. There may also be references within the renumbered area that refer to other lines within the renumbered area. All these must be considered.

The I loop brought in one line and, if necessary, changed its line number. Now it's the job of the Q loop to examine the line in detail for any line references within the line that must be changed. The Q loop, in line 870, reads the line of code from the first position of the line to the total length of the line, one character at at time. In line 880, we let C$ equal itself plus the next character. The mechanism for doing this is the MID$ of A$, starting at Q for one character. This effectively "builds" C$. Let's take an example line (A$) and see how it does it. If A$ was this line: 100 IF A=5 THEN 850, then as the Q loop increments, C$ would look like this:

```
1
10
100
100
100 I
100 IF
100 IF
100 IF A
100 IF A=
100 IF A=5
100 IF A=5
100 IF A=5 T
100 IF A=5 TH
100 IF A=5 THE
100 IF A=5 THEN
```

Now let's jump back into the logic for a bit and see what's going on there. As C$ builds, we let S$ equal the right four characters of C$ in line 890. As we add each character to C$ we make checks in lines 900 through 950 to see if certain things have occurred. The first of these is in line 900, where we check first of all to see if the right two characters of the right four characters in S$ are equal to ON. This will be used later in the ON GOTO and ON GOSUB cases. If the characters ON are found, we don't do anything except set flag F1 to 1. The next things we look for are the keywords "THEN", "ELSE", "OSUB" and "GOTO". The word "OSUB" is unique enough to identify the keyword GOSUB, which is five characters long, and keeps us from making a special case, in regards to length, for the GOSUB.

If you look back to where we built C$, you will see that we stopped where C$ reached the "N" in "THEN". Keep in mind that S$ is always equal to the right four characters in C$. Now in line 910, S$ does equal "THEN" and the Q index number is at 15 because the "N" is the fifteenth character from the left of the line, and we go to a subroutine at line 1070 to find out if there is a number following the word "THEN". Notice that in any program the keywords "THEN" and "ELSE" do not necessarily need to be followed by a line number, as in "THEN A=4" or "ELSE J$=MID$(A$,P,Q)". They *may* be followed by line numbers. The keywords "GOTO" and "GOSUB" are always followed by line numbers. Let's take a look at the find and replace subroutine at 1070.

Find/Replace Subroutine

First of all, we wouldn't even be here if we had not encountered one of the keywords earlier. But in our example line, we found the keyword "THEN" and so we need to know what follows that word. The first thing we do at line 1070 is to let A equal the integer value of whatever follows the "N" in THEN. To do that, we look at the MID$ of A$, starting at Q plus 1 (Q was at the fifteenth position), for the next nine characters. Why Q plus 1? Because looking at Q would give us the integer value of the "N" which would be zero. Why look nine characters ahead? Because the number (if there is one there) could be as long as five characters, and some people like to (or inadvertently) put extra spaces in their lines.

Let's talk about VAL for a bit, it's a very interesting function. VAL returns the integer value of numbers that are in string form. If there are no numbers in the string, it returns a zero. It only reads numbers up to, but not including, the

I	A(I)	B(I)
1	100	0
2	110	0
3	120	0
4	130	0
5	140	0
6	150	0
7	160	0
8	170	0
9	180	0
10	190	0
11	200	0
12	210	202
13	220	204
14	230	206
15	240	208
16	250	210
17	260	212
18	270	214
19	280	216
20	290	218
21	300	220
22	310	222
23	320	224
24	330	226
25	340	228
26	350	230
27	360	232
28	370	234
29	380	236
30	390	238
31	400	240
32	410	242
33	420	244
34	430	246
35	440	248
36	450	250
37	460	252
38	470	254
39	480	256
40	490	258
41	500	0
42	510	0
43	520	0
44	530	0
45	540	0
46	550	0
47	560	0
48	570	0
49	580	0
50	590	0
51	600	0
52	610	0
53	620	0
54	630	0
55	640	0
56	650	0
57	660	0
58	670	0
59	680	0
60	690	0
61	700	0
62	710	0
63	720	0

Figure 1

This figure shows the A and B arrays after asking the program to renumber Qtext.Bas (in this issue) starting from line 200 to line 500 with an increment of two.

first non-numeric character. It will ignore leading spaces. The VAL of the string "A123" would be zero because of the letter "A". The VAL of the string, " 12:30" would be 12 because of the colon. In our example above, if we did not increment Q by one, the VAL of the characters following "THEN" would be zero because of the "N" but the VAL from Q plus one will be 850. Now back to the subroutine.

If we do not find an integer value starting at Q plus 1, line 1080 will send us back to where we came from (via the RETURN at line 1150) and C$ will continue to build while looking for other keywords. In our case, however, we found the number 850 following the word "THEN" and so now we need to do something with it. We know that we are either going to replace that number with a different one from the B array (if it is in the range to renumber) or put it back (if it is not in the range to renumber.) In either case, we are not going to handle the number one character at a time, we are going to replace the whole number in one chunk. That being the case, we had better advance the Q counter *past* the number of spaces it originally took up. That little operation takes place in line 1090. In 1090 we let Q equal the number represented by the INSTR (instring) value of the *position* of the string value of A (850), plus the length of the string that A would make, less one. Let's do that again. The STR$(A), the string value of A, will be a space followed by the digits 850. Q will be positioned first at that space. Then we add the length of STR$(A) to this position, which will be four because there are three digits and the leading space that STR$ creates when you make STR$ from a number. So Q will be repositioned in A$ at the next position following the number we just removed. The minus one is necessary because of the extra space that STR$ puts in front of the STR$(A). Without it, we may position Q one space too far down the line and miss the first character following the number we are replacing. In our example that doesn't happen because the number is at the end of the line, but what about this: 100 IF A=5 THEN GOSUB 850:GOTO 120? We wouldn't want to lose the colon when we replace the number 850.

Fine. We now know that we are dealing with a number and have repositioned the Q count past it. Now we need to deal with the number itself. The first thing we need to do is to check and see if it is a valid line number reference. In other words, does it refer to a real line number or does it just point to nowhere? In line 1100 we leave the subroutine we are in for a while and go to yet another subroutine

at 1330. In the subroutine at 1330 we scan the entire A array, looking for a match for our number 850. If we find it, we simply return because it is a valid reference. If we do not find it in the list in the A array, we print a message on the screen that says "Undefined line XX in XX". Then we add a quote, a question mark, and another quote to C$. This will place the question mark at the spot in A$ where the undefined line reference was. Why use quotes around the question mark? Because BASIC will convert the question mark to the word PRINT if we don't. Either way, we do return from this subroutine to the one that called it in line 1100.

Having taken care of the valid or invalid line reference, we now check to see if our number (850) is within the range to be renumbered in line 1110. First, let's take the case where 850 is in the range to be renumbered. If it is, then line 1110 will send us to the very next line. There is something about sending program flow to the very next line that doesn't seem right, but in this case it seems to be the most straightforward way to do it. The little loop between 1120 and 1140 searches the A array to find our number (850). We already know that it will find it because line 1110 said it was within the range. What we want to know now is *where* it is. When we find it, the index M will tell us where in the B array to get our new number. In line 1130 we then add the STR$ of the number in the B array to C$ and return to line 920.

If our number (850) was not in the range to be renumbered then line 1110 would simply add the STR$ of the original number (850) back to C$ and go to 1150 to return to line 920.

Back in the Q loop between 870 and 960 we will then check for other keywords and make the appropriate changes if necessary.

The ON GOTO problem

In line 900, if we encounter the characters "ON", we have set flag F1 to 1. If, in the same line, we later encounter the keywords "GOTO" or "OSUB" we set another flag F2 to 1. The first number following an ON GOTO or ON GOSUB will be treated in the fashion just described above. The numbers following the first can't be handled that way. This program assumes that there is nothing following the list of numbers following an ON GOTO or ON GOSUB. In line 950 if both flags are set to 1 we know that there is an ON GOTO or ON GOSUB situation in that line. We then go to the subroutine at line 1180.

At 1180 we use a loop (X) to find the commas in the line. The variable P tells us the position of the first comma. If P is equal to zero it means that there are no more numbers and we return via line 1300. If there is a comma, line 1210 looks for the number after it (we have already taken care of the first number by the normal method.) Again, as before, if A is zero there is no number there and we return. We use the subroutine at 1330 again to check and see if the number is a valid line number. We use the same method as described earlier to find the number in the A array and replace it with the number in the B array or replace the same number if it is not in the area to be renumbered. There is a slight difference here. Notice lines 1240 and 1260. Instead of making the STR$ of the number, we are stripping off the leading space by using the MID$ starting at position 2 first. If we didn't, the number we put back would overwrite the following comma.

Having replaced the number, we now (in line 1280) increment the X index to the position of the most recent comma. The NEXT will move X to one past that position, where P will again look for the next occurrence of a comma. We build the entire remainder of C$ this way and then return to line 950 where a hard GOTO takes us around the NEXT Q in 960 because we are all done with this line.

Limitations

No check is made in this program to see if the new line created by C$ grows longer than 255 characters. If you use very long lines, the new line numbers (if larger than the old ones), may push the limit for line length. In that case you will get a "String too long" error.

There are some additional keywords that may be followed by line number references. Some you may want to add are: RESTORE, RESUME and RETURN. Some BASIC's allow the use of RETURN with a line number following it. If you want to add any of these three, put them between lines 920 and 930. Use the last four characters of each command, i.e., RESTORE would be: 922 IF S$="TORE" THEN GOSUB 1070.

There was no doubt that compiling this program would improve its speed of execution. We did, with the Microsoft Quickbasic compiler, and found a reduction from almost four minutes to less than 45 seconds on a 100 line program.

Some BASIC's do not require spaces around keywords. We have tried several lines of long,

tightly packed code trying to cover as many cases as possible. The program worked well, however, there is no guarantee it will work in all such conditions.

As always, there are many possible ways to do a program such as this. This is just one of them. ■

```
100 REM * RENUM.BAS * A Selective renumber program *
110 REM * Created for CodeWorks magazine, 3838 South Warner St.
120 REM * Tacoma, WA 98409 (206)475-2219 voice and
130 REM * (206)475-2356 300/1200 baud modem
140 'CLEAR 2000: 'Use only if you need to clear string space.
150 NL=600: ' Sets the max number of lines the program can handle.
160 DIM A(NL),B(NL)
170 CLS:' Clear screen command, change to suit your machine.
180 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
190 PRINT"                R E N U M B E R   P R O G R A M"
200 PRINT"          Selectively renumbers sections of BASIC code"
210 PRINT STRING$(60,45)
220 PRINT"  This program allows you to renumber any section of code"
230 PRINT"WITHIN a BASIC program, and keeps all references into"
240 PRINT"and out of the affected section in order. It can also
250 PRINT"renumber the entire program, but the new starting line"
260 PRINT"number will be the current starting line number."
270 PRINT"  This program will create a new program on your diskette"
280 PRINT"with the same name but with the extention .NEW, keeping
290 PRINT"the old program (.BAS) as an unspoiled backup.
300 PRINT"  The program to be renumbered must have been saved"
310 PRINT"in ASCII format, i.e., (SAVE''filename.BAS'',A)"
320 PRINT
330 PRINT"Press ENTER to continue";:INPUT X
340 CLS:INPUT"What is the name of the program to renumber";F$
350 PRINT
360 PRINT"Starting line number"
370 INPUT"  (ENTER will start at current lst line)";SN
380 PRINT
390 PRINT"Ending at line number"
400 INPUT"  (ENTER will renumber thru end of program)";EN
410 PRINT
420 INPUT"  With what increment";IC:IF IC=0 THEN IC=1
430 PRINT
440 PRINT"Renumbering program ";F$
450 '
460 REM * Read the file - put its line numbers into the A( ) array.
470 OPEN"I",1,F$
480 FOR I=1 TO NL
490   IF EOF(1) THEN 550
500   LINE INPUT #1,A$
510   S=INSTR(A$," ")
520   LN=VAL(LEFT$(A$,S))
530   A(I)=LN
540 NEXT I
550 CLOSE 1
560 N=I-1
570 IF SN<A(1) THEN SN=A(1)
580 IF EN=0 THEN EN=65500!
590 '
```

```
600 REM * Read the A( ) array and put new line numbers in B( ) array.
610 SU=SN:CT=0
620 FOR I=1 TO N
630   IF A(I)>SN AND A(I)<EN THEN B(I)=SU+IC:SU=SU+IC:CT=CT+1
640 NEXT I
650 '
660 REM * Check to see if what you want to do will work.
670 IF SN+(CT*IC)<EN THEN 710 ELSE PRINT"It won't fit. Reduce the
    increment"
680 PRINT"to";INT((EN-SN)/CT)-1;" or less to prevent line clashes."
690 INPUT"Enter the new increment ";IC:IF IC=0 THEN IC=1
700 FOR I=1 TO N:B(I)=0:NEXT I:GOTO 610
710 FOR I=1 TO N:PRINT I,A(I),B(I):NEXT I
720 '
730 REM * Set up the name for the new output file.
740 IF INSTR(F$,".") THEN X=LEN(F$):TP$=MID$(F$,1,X-4):F1$=TP$+".NEW"
    ELSE F1$=F$+".NEW"
750 '
760 REM * Now read the file again and substitute line numbers.
770 OPEN"O",2,F1$
780 OPEN"I",1,F$
790 FOR I=1 TO N '<----------------<< The main processing loop
800   LINE INPUT #1,A$
810   PRINT A$
820   L=LEN(A$)+1
830   S=INSTR(A$," ")
840   LN=VAL(LEFT$(A$,S))
850   IF LN>SN AND LN<EN THEN T$=MID$(A$,S,L-S):A$=STR$(B(I))+T$
860   L=LEN(A$)
870   FOR Q=1 TO L
880     C$=C$+MID$(A$,Q,1)
890     S$=RIGHT$(C$,4)
900     IF RIGHT$(S$,2)="ON" THEN F1=1
910     IF S$="THEN" THEN GOSUB 1070
920     IF S$="ELSE" THEN GOSUB 1070
930     IF S$="OSUB" THEN F2=1:GOSUB 1070
940     IF S$="GOTO" THEN F2=1:GOSUB 1070
950     IF F1=1 AND F2=1 THEN GOSUB 1180:GOTO 970
960   NEXT Q
970   PRINT #2,C$
980   PRINT C$
990   C$="":F1=0:F2=0
1000 NEXT I
1010 CLOSE
1020 PRINT:PRINT"DONE. The renumbered program is called: ";F1$
1030 PRINT"       The original program remains as: ";F$
1040 END
1050 '
1060 REM * Find & replace number after keyword and increment Q *
1070 A=VAL(MID$(A$,Q+1,9))
1080 IF A=0 THEN 1150
1090 Q=INSTR(Q,A$,STR$(A))+LEN(STR$(A))-1
1100 GOSUB 1330
1110 IF A>SN AND A<EN THEN 1120 ELSE C$=C$+STR$(A):GOTO 1150
```

```
1120 FOR M=1 TO N
1130   IF A=A(M) THEN C$=C$+STR$(B(M))
1140 NEXT M
1150 RETURN
1160 '
1170 REM * Take care of the 'On Goto' and 'On Gosub' cases *
1180 FOR X=Q TO L
1190   P=INSTR(X,A$,",")
1200   IF P=0 THEN 1300
1210   A=VAL(MID$(A$,P+1,9))
1220   IF A=0 THEN 1300
1230   GOSUB 1330
1240   IF A>SN AND A<EN THEN 1250 ELSE C$=C$+","+MID$(STR$(A),2):GOTO
       1280
1250   FOR M=1 TO N
1260     IF A=A(M) THEN C$=C$+","+MID$(STR$(B(M)),2)
1270   NEXT M
1280   X=P
1290 NEXT X
1300 RETURN
1310 '
1320 REM * Find Unreferenced line numbers *
1330 FOR M=1 TO N
1340   IF A=A(M) THEN M=N:GOTO 1380
1350 NEXT M
1360 PRINT "Undefined line";A;"in";LN
1370 C$=C$+CHR$(32)+CHR$(34)+CHR$(63)+CHR$(34)
1380 RETURN
1390 END:'OF PROGRAM
```

Programming Notes

There is a BASIC error message that says, "Too Many Files in (line #)." Checking the books disclosed that this message is generated when the diskette directory is full. We got the message in an entirely different way. In the program Qtext.Bas in this issue, for example, we ask for a file name and then append .DOC to it. If you forget that the program does the appending and you inadvertently enter "Filename.DOC" it appends another .DOC to it, and you get the Too Many Files error. We found this in GW BASIC, then tried it on machines running Microsoft BASIC prior to version 5.1 and found that they simply strip off the extra file extension and go on their merry way.

Getting the number of an error is different on BASIC versions prior to 5.1 and after. After version 5.1 an error 53 is "File not found" and you can check for it with: IF ERR = 53 THEN ... With earlier BASICs you need to do this to get the same thing: IF (ERR/2)+1 = 54 THEN ... We ran into this one with Tandy Models I and III while checking out some of our programs where we use the error code to create a file if none exists (see Form.Bas in Issue 8, starting at line 1430.)

Beginners in BASIC sometimes get quite confused when they see a line, early in the program, that says DEF FNX=LOF(X) where the variable X has not even yet been defined. DEF FN is the defined function in BASIC. It lets you define sequences that will be processed when you call the function and feed it (in this case) the value of X. When BASIC reads the DEF FN line it just remembers that it has that function and when called, what to do with it. Defined functions can even include other defined functions. For an example of them, see Randemo3.Bas in this issue, lines 40 and 41.

Beginning Basic

The many uses of INSTR

INSTR is a powerful BASIC function. A BASIC function is one that is made up of a series of internal steps and usually called into play by one BASIC keyword. All functions return a value when called or executed. INSTR is usually described as:

INSTR((*number,*)*string1,string2*)

The explanation which follows is somewhat cryptic and says something to the effect that INSTR searches for the first occurrence of string2 in string1 and returns the position at which that match is found. It then goes on to say that *number* is optional, and if left out, the search starts with the first position in string1, otherwise the search starts at the character position given by *number*.

All of which is true - but you guessed right when you assume there may be more. There is. To begin with, INSTR probably stands for "instring," at least that is the way most of us pronounce it. If A$="The dog was chasing the cat", and B$="dog", then if P=INSTR(A$,B$), P would equal 5 because the match for "dog" was found at the fifth position in A$. If B$ were "do" it would still be found at the fifth position.

So now that we have found the dog and the do, so what? Well, for one thing, we can now take the LEFT$ of A$ up to position P, then take the MID$ of A$ starting at P plus the length of B$, to the length of A$, then change B$ to "horse" and put A$ back together again by concatenation so that A$ will read: "The horse was chasing the cat".

We need to go through this exercise when changing words in a string because you cannot change the length of any string *except by concatenation*, which means you cannot stuff more characters into a string than already exist. Note that "horse" is a longer word than "dog", and although we could have replaced the three letters in "dog" to any other three letters without concatenation, we could not get the "horse" in there by any other method. If the replacement string is as long (as many characters) as the string being replaced, you can avoid the use of INSTR and use a MID$ substitution instead. Replacing equal length strings is not always the case, however.

Now what about the possibility that two dogs may be in A$? In that case we can say P=(I,A$,B$), where I was previously initialized to one. Then, when you have found the first dog, set I equal to P+1 and do it again. If you don't add one to P it will keep finding the same dog over and over again. If we found the first dog at position five, then we want to start our search for the second dog (assuming there may be one) at position six.

There are more uses for INSTR. For example, if the string being searched for is not found, then INSTR returns a zero. So if you simply want to know if A$ contains a dog, and are not necessarily concerned with *where* the dog is, you can say:

```
IF INSTR(A$,B$) THEN... ELSE...
```

which says that if "dog" is *anywhere* in A$, then do something, otherwise, do something else. In this example INSTR is used to test for a logical true or false condition. If INSTR finds no match it will be zero, or false, and the ELSE route will taken.

INSTR, along with INKEY$, can be used to *stop* a process. Consider the following code:

```
10 IF INSTR(" Ss",INKEY$)>1 THEN 30
20 GOTO 10
30 PRINT"You pressed S or s to stop"
```

This illustrates how INSTR can be used inside a loop to pause or quit operation of that loop. The space before the first S is important because if INKEY$ is a null then INSTR would find that null every time. So we move the Ss over one position and make the condition greater than one. This could be used to stop the printer temporarily while you adjust the paper feed, for example. Depending on how you use it inside a loop, many other functions can be performed with this little bit of code.

INSTR also comes in handy when checking input prompts. Here is another example:

```
100 PRINT"Do you want to continue (Y/N)"
110 X$=INKEY$:IF X$="" THEN 110
```

```
120 P=INSTR("YyNn",X$)
130 IF P=0 THEN 110
140 IF P=1 OR P=2 THEN 100
150 IF P=3 OR P=4 THEN 160
160 PRINT"You said N":END
```

This code will accept either upper or lower case Y or N. When program flow gets to line 110 it will loop on itself, waiting for a key to be pressed. INKEY$ accepts one character from the keyboard, and until it sees one, it will loop because of the second statement in line 110. In line 120 a check is made to see if the key which was pressed (and now contained in X$) is part of "YyNn". If the key was Y or y, P will equal one or two, if the key was N or n, P will equal three or four. *Any other key pressed will not be found in "YyNn" and so P will equal zero and line 130 will send control back to line 110 to wait for upper or lower case Y or N.* This code is quite discriminating, as you can see when you try it.

Here is still another way to do it:

```
10 INPUT"Do you want to continue (Y/N)";X$
20 IF INSTR(".YES.yes.Yes.Y.y.","."+X$+"."
    )THEN ...
```

In this one you can answer "yes" almost any way you want to (you can add .Yep. to the string if you like, or even .Oui., if that suits you.) The periods separating the words in the string are necessary to prevent you from being able to ask for "Sye" or "esY" and getting it. Because of the periods, we also need to concatenate a period before and after our X$ input so that a match can be found.

We all use ON GOTO and ON GOSUB, but did you know that ON INSTR works too? Here is an example where A$ is always a four-letter command and can be in either all upper or lower case. The commands we will use in this example are HELP, QUIT, EDIT, and SAVE. The code that picks out the command and sends control to the proper place in the program looks like this:

```
100 INPUT"What is your command, Sire";A$
110 ON INSTR("....HELPhelpQUITquitEDIT
  editSAVEsave", RIGHT$(A$,4))/4 GOTO 130,
130, 140, 140, 150, 150, 160, 160
120 PRINT"Invalid command":GOTO 100
130 PRINT"Your command was help":GOTO 100
140 PRINT"Your command was quit":GOTO 100
150 PRINT"Your command was edit":GOTO 100
160 PRINT"Your command was save":GOTO 100
```

The dual line numbers in the GOTO portion of line 110 take care of both the upper and lower case possibilities. If an invalid command is entered, flow falls through to line 120, which sends control back to ask for another input. The four periods (they could be any character) placed before the first HELP in line 110 are necessary to make the division by four come out in the numbers 1 through 8. This example, taken out of context as it is, will allow you to cheat by typing "H" and getting help, which means that no other command should start with the letter H. Inside the program it was used in, however, a command processor had already screened the commands properly before they arrived at this point in the code.

As you can see, INSTR has many useful purposes. It should be a handy addition to your collection of programming tools. ■

"I think we're in trouble. The instruction manual is printed on a bar napkin."

Random Files

The start of our random database

Terry R. Dettmann.

In earlier issues we presented some of the more elementary concepts of random files. Beginning with this issue we will start to build towards a random file data base system. The program with this article is the core upon which we will expand.

Randemo3.Bas is a small data base system built to illustrate simple random file handling. It also brings out some problems in dealing with random files. We will deal with these as we come to them.

The purpose of the program is to build a simple data base and show the basic functions needed in any such program. When you start the program, it will show you each option:

1. Add Records to the data base
2. Edit Records in the data base
3. Delete Records in the data base
4. Print the data base

These minimum functions must always be provided in some way.

Random files are different from sequential files in that you can jump immediately to any record in the file if you know its record number. In order to do this, all records in the file must be the same length (the record length.) By knowing the record length, we can compute the location of any record in the file from its record number.

The Program

The program starts in lines 10 to 60 with initialization of the system. The array FP$ is defined to be the string of data for each record. The function FNCTR$ is defined to provide a centering function for labels. Function FNLF is defined for MSDOS machines to provide the number of records in the data base. On machines using BASIC versions prior to 5.1 (notably Tandy Models I and III), LOF gives the number of records in a file. In GW BASIC it gives the number of bytes, so we have to divide by the record length (128 in this case) to get the number of records. (See Adjustments, at the end of this article.)

Variables WD and LN are set up to define the screen size (width 80 characters, length 24 lines.) Only WD is used in this program. Variables FALSE and TRUE are defined to use for decision making.

Lines 100 to 160 are for file setup. We first get the name of the data base file we want to use and add an extension (.dat) to it. Then we open the file as a random access file ("R" says that this is a random file rather than a sequential one.) Note that, unlike sequential files, if the file does not exist the random open statement will create it.

Every time we use a random access file, we need to tell the computer where to find the data within a record. The FIELD statement does this. The FIELD is set up to use the array FP$ to point to the locations of the data within each record. Each data item is 25 characters long to make it simple for now. The For...Next loop provides a simple way to lay out the field statement without making a very complex statement. ZZ$ is a throw-away variable that gets redefined to cover a larger and larger space in the record to make the FIELD statement come out right for each element of the array FP$.

Lines 200 through 350 are the menu of options and command handling. When we select a command, we error-check it (line 320) to see if it's a good command, then execute it if it is ok (lines 330-340.) Let's look at the commands.

Adding Records

Whenever you have a data base, the first thing you have to do is get data into it. Lines 1000-1999 are reserved for that purpose. Lines 1000 to 1110 ask for one field at a time until all five fields have been entered into the data base. As each line is typed, it is entered into a data record (LSET FP$(I)=LN$.) When all are in, line 1065 saves them to the data base. You are then allowed to add more or get out if you want to.

Lines 1200 to 1299 save records to the data base. In line 1065, we set RN (record number) equal to zero before we call the record save routine at line 1200. That means we don't know the record number to save to, so we get one from the subroutine at 1300. Subroutine 1300 simply says "Put all new records at the end of the file."

With a record number assigned, line 1220 puts it into the data base (PUT 1,RN.) Whenever we want it back, subroutine 1400 will get it back into memory for use (GET 1,RN.)

Entering records is rather easy, isn't it? But look for a minute. All new records are added to the end

of the file. What if we delete some records, how can we reuse the holes we created in the file? There are several ways, which we will deal with in future installments of this series.

Editing Records

Once we have some records in the data base, we may want to correct the information in them. This calls for an editing function (Lines 2000-2999.) To edit a record, you first have to find it. Line 2020 enters some information which has been stored in the record and subroutine 2100 looks for it. If it finds the record, the variable FOUND is set to TRUE. If it is not found, the variable is set to FALSE.

If we find the record, line 2040 directs us to subroutine 2200 where we can edit it, otherwise, it tells us it couldn't be found.

To find a record (subroutine 2100,) we are using a brute-force search. We start at the beginning of the file and examine each record until we find the one that matches. This is a very easy thing to write, but if your file gets big, it gets very slow. For now, we'll leave complex search procedures alone and come back to them as this series progresses.

Notice that each field in the data record is examined to see if the search string appears (INSTR(FP$(I),SF$<>0)). In this way, we can search for *anything* within the record. More sophisticated, faster, search methods generally give up this ability to search the file for anything you want, or else they add a lot of overhead to the search, requiring considerably more disk storage.

Once we have found a record, we display it and allow fields to be changed. When all changes are complete, the record is saved back to the data base on diskette.

Deleting Records

Record deletion starts out the same way that editing does: by finding the record. In fact, it even uses the same subroutine (subroutine 2100) to do the search. Once a record is found, it is displayed to check and make sure it is really the right one, and then the first field is set to the word "DELETED" if you agree to delete it.

Notice that the subroutine at 1200 is still called to save the record back to the data base. Everything goes through that subroutine.

Printing Records

Printing the data base is simple: just go through the records one at a time and if they don't show DELETED, then print them. But what about sorting them into order before printing?

If the data base is pretty small, we could sort the data into order for printing by reading them all into memory and then sorting them there. Once the file grows larger than memory (and random files can, and do) then we must do something else. We could sort the data in the file itself, but that is *very* slow. Generally, we build an INDEX to the file, like a directory with the record numbers in it, to work with instead. This too, will be covered as we continue this series.

Adjustments

The line numbering of this program does not conform to our usual system because we are going to continue to build on it.

If you have more than two or three records, they will fly by fast when you print the records. You can either change the PRINT in line 4040 to LPRINT, or insert a temporary line, 4065 INPUT"Press ENTER for more.";XX.

Machines requiring the /EXT instead of the .EXT should make the appropriate changes in all file designations. Also, machines with screens smaller than 80 x 24 should set their size in line 50.

BASICs prior to version 5.1 (i.e., Tandy I and III and others) should remove the /128 (divide by 128) in line 41.

Machines unable to use more than two- letter variable designators need not worry about FOUND, FALSE, etc. The first two letters of those variables have not been used elsewhere in the program. ■

```
10 REM --- Random File Demonstration 3
20 REM --- Terry R. Dettmann for Codeworks Magazine
30 DIM FP$(5)
40 DEF FNCTR$(X$)=STRING$((WD-LEN(X$))/2," ")+X$
41 DEF FNLF(X) = LOF(X)/128
50 WD=80:LN=24
60 FALSE=0:TRUE = NOT FALSE
100 REM --- file setup
110 CLS:PRINT FNCTR$("RANDOM FILE DEMO 3"):PRINT:PRINT
120 LINE INPUT"FILENAME: ";FF$
125 FD$=FF$+".dat"
```

```
130 OPEN "R",1,FD$
140 FOR I=1 TO 5
150     FIELD 1, (I-1)*25 AS ZZ$, 25 AS FP$(I)
160 NEXT I
200 REM --- main menu
210 CLS:PRINT FNCTR$("RANDOM FILE DEMO 3"):PRINT:PRINT
220 PRINT TAB(15)"1. Add Records to the File"
230 PRINT TAB(15)"2. Edit Records from the File"
240 PRINT TAB(15)"3. Delete Records from the File"
250 PRINT TAB(15)"4. Print the File"
260 PRINT
270 PRINT TAB(15)"0. End the Program"
280 PRINT:PRINT
290 PRINT TAB(15)"Command: ";
300 LINE INPUT CD$
310 CD = VAL(CD$)
320 IF CD<0 OR CD>4 THEN PRINT"Bad Command":GOTO 290
330 IF CD=0 THEN 500
340 ON CD GOSUB 1000,2000,3000,4000
350 GOTO 200
500 REM --- End of Program
510 CLS:PRINT"All Done":CLOSE:END
1000 REM --- Add Records
1010 CLS:PRINT FNCTR$("ADD RECORDS"):PRINT:PRINT
1020 FOR I=1 TO 5
1030     PRINT TAB(15)"FIELD(";I;"): ";:LINE INPUT LN$
1040     IF LEN(LN$)>25 THEN PRINT"TOO LONG, TRY AGAIN":GOTO 1030
1050     LSET FP$(I)=LN$
1060 NEXT I
1065 RN=0:GOSUB 1200
1070 LINE INPUT"Add More (y/n)? ";YY$
1080 C$=MID$(YY$,1,1)
1090 IF C$="Y" OR C$="y" THEN 1010
1100 IF C$="N" OR C$="n" THEN RETURN
1110 GOTO 1070
1200 REM --- save current record to data base
1210 IF RN=0 THEN GOSUB 1300
1220 PUT 1,RN
1230 RETURN
1300 REM --- assign a record
1310 RN = FNLF(1) + 1:RETURN
1400 REM --- get record from data base
1410 IF RN<1 OR RN>FNLF(1) THEN RETURN
1420 GET 1,RN
1430 RETURN
2000 REM --- Edit Records
2010 CLS:PRINT FNCTR$("EDIT RECORDS"):PRINT:PRINT
2020 LINE INPUT"SEARCH FOR: ";SF$
2030 GOSUB 2100
2040 IF FOUND THEN GOSUB 2200 ELSE PRINT"NOT FOUND"
2050 LINE INPUT"Edit More (y/n)? ";YY$
2060 C$=MID$(YY$,1,1)
2070 IF C$="Y" OR C$="y" THEN 2010
2080 IF C$="N" OR C$="n" THEN RETURN
```

```
2090 GOTO 2050
2100 REM --- search for record
2110 FOUND=FALSE
2120 FOR RN=1 TO FNLF(1):GOSUB 1400
2130    FOR I=1 TO 5:IF INSTR(FP$(I),SF$)<>0 THEN FOUND=TRUE:RETURN
2140    NEXT I
2150 NEXT RN
2160 RETURN
2200 REM --- edit record
2210 CLS:PRINT FNCTR$("EDIT RECORDS"):PRINT:PRINT
2220 FOR I=1 TO 5
2230    PRINT TAB(15)"FIELD (";I;"): ";FP$(I)
2240 NEXT I
2250 PRINT:PRINT
2260 LINE INPUT"FIELD TO CHANGE OR '0' TO END: ";FC$
2270 C$=MID$(FC$,1,1):IF C$="0" THEN GOSUB 1200:RETURN
2280 C=VAL(C$)
2290 LINE INPUT "CHANGE: ";LN$
2300 LSET FP$(C) = LN$
2310 GOTO 2200
3000 REM --- Delete Records
3010 CLS:PRINT FNCTR$("DELETE RECORDS"):PRINT:PRINT
3020 LINE INPUT"SEARCH FOR: ";SF$
3030 GOSUB 2100
3040 IF FOUND THEN GOSUB 3200 ELSE PRINT"NOT FOUND"
3050 LINE INPUT"Delete More (y/n)? ";YY$
3060 C$=MID$(YY$,1,1)
3070 IF C$="Y" OR C$="y" THEN 3010
3080 IF C$="N" OR C$="n" THEN RETURN
3090 GOTO 3050
3100 REM --- delete the current record
3110 LSET FP$(1)="DELETED"
3120 GOSUB 1200
3130 RETURN
3200 REM --- delete record
3210 CLS:PRINT FNCTR$("DELETE RECORDS"):PRINT:PRINT
3220 FOR I=1 TO 5
3230    PRINT TAB(15)"FIELD (";I;"): ";FP$(I)
3240 NEXT I
3250 PRINT:PRINT
3260 LINE INPUT"Are you sure (y/n)? ";YY$
3270 C$=MID$(YY$,1,1)
3280 IF C$="Y" OR C$="y" THEN GOSUB 3100:RETURN
3290 IF C$="N" OR C$="n" THEN RETURN
3300 GOTO 3050
4000 REM --- Print File
4010 CLS:PRINT FNCTR$("PRINT FILE"):PRINT:PRINT
4020 FOR RN=1 TO FNLF(1):GOSUB 1400
4030    IF INSTR(FP$(1),"DELETED")<>0 THEN 4070
4040       FOR I=1 TO 5:PRINT "FIELD(";I;"): ";FP$(I)
4050       NEXT I
4060       PRINT
4070 NEXT RN
4080 RETURN
```

Poker7.Bas

Updates on Poker.Bas from the last issue

Staff Project (with some help). Here are changes to make the Tandy Model III print the suit symbols, thanks to Robert Hood. Also there are changes to make the game work with seven players (if you have space on your video screen.)

Card suits for Tandy Model III

Shortly after Issue 8 went into the mail we received a letter from Robert Hood of Bremerton, Washington with changes to Poker.Bas to make the Tandy Model III show the card suits. There are not many, they are not difficult. We just didn't know about them when we wrote the program. Here are the lines to change:

```
140 CLEAR 1000:POKE 16420,1
3070 IF FNI(M(I,J))=1 THEN SU=192:GOTO
3110
3080 IF FNI(M(I,J))=2 THEN SU=193:GOTO
3110
3090 IF FNI(M(I,J))=3 THEN SU=194:GOTO
3110
3100 IF FNI(M(I,J))=4 THEN SU=195
3110 B$(J)=D$+CHR$(SU)
```

We had the opportunity to try this change on a Model III and the suit symbols are even more sharply defined than they are on the MSDOS machines we tried.

Jacks or better for 7 players for 24-line screens only

The listing following this text shows the lines in the original Poker.Bas which need to be changed to convert it to seven players and "Jacks or better" to open, making it a true Jackpot Poker program.

Only two lines needed to be added, they being the ones to print the two additional players on the screen. The remainder of the lines are lines that need to be changed from the original version. Many of them do nothing but change the various loop counters to go to seven instead of five.

Something new popped up when we first tried this. It turns out that with seven players there aren't enough cards when all seven players draw three. So we do what they do in the real game. We put the discard under the deck and continue to deal.

The most expedient way to handle the "Jacks or better" problem was to add an additional array position (13) to the M array. It holds a temporary value for the hand used to make the determination of who can open.

This change makes an entirely new game of Poker.Bas. It seems that two pair is usually not good enough any more. Aside from that, if two or three players fold, you still have a viable game going. It's not as easy to win as with five players. We have also made it a slightly more expensive game; the ante is now $10.

A merged version of Poker7.Bas will be available on the download under the Issue 9 menu.

```
100 REM * POKER7.BAS ** FIVE CARD DRAW POKER PROGRAM *
170 DIM B(74),M(7,13),B$(52)
230 PRINT"          J A C K P O T   P O K E R   P R O G R A M"
270 PRINT"You may watch 7 players or be one. Choose 0 to watch, 1 to"
280 PRINT"7 to sit in. The game ends when any player goes broke."
290 PRINT"There are no wild cards, a 3-card draw limit, jacks or
    better"
300 PRINT"to open. Chips are $5, $10 and $25. Bet in multiples of $5"
350 INPUT"Which position do you want to play: 0 = none or 1-7";U
360 IF U<0 OR U>7 THEN 350
410 FOR I=1 TO 7:M(I,10)=DO:NEXT I
420 CLS:GOTO 460
450 FOR T=1 TO 1000:NEXT T      'delay loop
460 '
```

```
470 FOR I=1 TO 7:IF M(I,10)=<0 THEN 5000
500 LOCATE 15,20:PRINT"Ante $10          " '9 spaces
510 IF A=>8 THEN A=1
520 FOR I=1 TO 14 STEP 2:LOCATE I,1:PRINT"PLAYER";I/2+.5:NEXT I
540 FOR I=2 TO 14 STEP 2:LOCATE I,1:PRINT M(X,10)
660 IF A1=>8 THEN A1=1:A=A+1:GOTO 430          'IF NO ONE OPENED
860 FOR I=1 TO 7:M(I,7)=0:M(I,9)=0:M(I,11)=0:M(I,13)=0:NEXT I
1040 FOR I=1 TO 7:M(I,6)=1:M(I,12)=0:M(I,13)=0:NEXT I:DS=53
1080 FOR I=1 TO 7
1160 REM ** EVERYBODY ANTE UP 10 BUCKS **
1170 FOR I=1 TO 7:POT=POT+10:M(I,10)=M(I,10)-10:NEXT I
1220 FOR I=1 TO 7
1360 FOR I=1 TO 7
1390 FOR I=1 TO 7
1420       IF FNM(M(I,J))=FNM(M(I,K))THEN V=V+1:M(I,13)=FNM(M(I,K))
1460   IF V=<3 THEN M(I,8)=V ELSE IF V=4 THEN M(I,8)=14:M(I,13)=14
       ELSE IF V=6 THEN M(I,8)=16:M(I,13)=14
1470   IF F1=0 AND S=6 THEN M(I,8)=4
1480   IF S=10 THEN M(I,8)=0:M(I,12)=1:M(I,13)=14
1530 FOR I=1 TO 7
1610   IF N(5)-N(1)=4 THEN M(I,8)=10:M(I,13)=14 '5 CARD STRAIGHT
1620   IF N(4)-N(1)=3 AND N(5)=14 AND N(1)=2 THEN M(I,8)=10:M(I,
       13)=14:AL=1:'ACE LO STRGT
1730 FOR I=1 TO 7
1850     IF FNI(M(I,J))<>S THEN B(DS)=M(I,J):M(I,J)=B(DK):DK=DK+1:
         DS=DS+1:M(I,7)=1
1920 FOR I=1 TO 7
1980   IF S=1 THEN B(DS)=M(I,3):DS=DS+1:M(I,3)=B(DK):DK=DK+1:
       B(DS)=M(I,4):DS=DS+1:M(I,4)=B(DK):DK=DK+1:B(DS)=M(I,5):DS=DS+1:
       M(I,5)=B(DK):DK=DK+1:M(I,7)=3:GOTO 2070
1990 IF S=2 THEN B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:B(DS)=M(I,
     4):DS=DS+1:M(I,4)=B(DK):DK=DK+1:B(DS)=M(I,5):DS=DS+1:M(I,
     5)=B(DK):DK=DK+1:M(I,7)=3:GOTO 2070
2000 IF S=3 THEN B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:B(DS)=M(I,
     2):DS=DS+1:M(I,2)=B(DK):DK=DK+1:B(DS)=M(I,5):DS=DS+1:M(I,
     5)=B(DK):DK=DK+1:M(I,7)=3:GOTO 2070
2010 IF S=4 AND FNM(M(I,5))>12 AND U<>0 AND INT(RND(1)*4)+1=2 THEN
     M(I,7)=0:BL=1:GOTO 2070
2020 IF S=4 THEN B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:B(DS)=M(I,
     2):DS=DS+1:M(I,2)=B(DK):DK=DK+1:B(DS)=M(I,3):DS=DS+1:M(I,
     3)=B(DK):DK=DK+1:M(I,7)=3:GOTO 2070
2050 B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:B(DS)=M(I,2):DS=DS+1:
     M(I,2)=B(DK):DK=DK+1:M(I,7)=2:GOTO 2070
2060 B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:B(DS)=M(I,2):DS=DS+1:
     M(I,2)=B(DK):DK=DK+1:B(DS)=M(I,3):DS=DS+1:M(I,3)=B(DK):DK=DK+1:
     M(I,7)=3
2110 FOR I=1 TO 7
2190 FOR I=1 TO 7
2350 FOR I=1 TO 7
2510 FOR I=1 TO 7
2620 FOR I=1 TO 7
2660 FOR I=1 TO 7
2700 FOR I=1 TO 7
2730 FOR I=1 TO 7
```

```
2780 LOCATE 15,20:PRINT STRING$(40,32):'CLEAR POT HAS LINE
3170 FOR I=1 TO 7
3270 FOR I=1 TO 7
3410 FOR I=1 TO 7
3480 FOR I=1 TO 7
3620 FOR I=1 TO 7
3700 I=A-1:A1=1:IF I=<0 THEN I=7
3720 I=I+1:IF I=>8 THEN I=1
3730 IF M(I,8)=0 OR M(I,8)=4 OR M(I,8)=5 OR M(I,8)=6 OR (M(I,8)=1 AND
     M(I,13)<11) THEN GOSUB 4450:PRINT"Passes":GOTO 3760
3750 IF M(I,8)>1 OR M(I,13)>10 THEN BO=I:GOSUB 4450:LOCATE X,Y-35:
     PRINT"Opener":RETURN
3770 I=I+1:IF I=>8 THEN I=1
3780 A1=A1+1:IF A1=>8 THEN RETURN
3840 IF BT<>10 AND BT<>15 AND BT<>20 THEN 3830
3850 IF M(I,8)=>10 THEN BT=BT+5
3860 IF FB=1 THEN BT=BT+5
3900 BO=BO+1:IF BO=>8 THEN BO=1
4040 I=I+1:IF I=>8 THEN I=1
4060 IF FL=>6 THEN RETURN
4130 IF HC=28 AND M(I,7)=0 AND BL=1 AND M(I,8)=1 THEN B1=1:R1=R1+1:
     GOTO 4320
4170 IF M(I,8)=0 AND BT>14 AND INT(RND(1)*4)+1=<2 THEN B1=3:GOTO 4360
4240 IF M(I,8)=3 AND INT(RND(1)*3)+1<>2 THEN B1=1:R1=R1+1:GOTO 4320
4250 IF R1<6 AND M(I,8)>2 THEN B1=1:R1=R1+1:GOTO 4320
4260 IF R1<4 AND M(I,8)=2 AND INT(RND(1)*3)+1<=2 THEN B1=1:R1=R1+1:
     GOTO 4320
4270 IF M(I,8)=0 OR (M(I,8)=1 AND M(I,13)<11) THEN B1=3:GOTO 4360
4280 IF M(I,8)<3 AND R1>5 THEN B1=3:GOTO 4360
4360 IF B1=3 THEN GOSUB 4450:PRINT"folds     ":M(I,6)=0:FL=FL+1:GOTO
     4040
4420 I=I+1:IF I=>8 THEN I=1
4502 IF I=6 THEN X=11:Y=36
4504 IF I=7 THEN X=13:Y=36
4520 LOCATE 17,1:RETURN:'STATUS LINE LOCATION
4530 LOCATE 15,20:PRINT"POT HAS $";POT:RETURN
4540 FOR I=1 TO 14 STEP 2:LOCATE I,9:PRINT STRING$(54,32):NEXT I:
     RETURN
4570 LOCATE 18,1:PRINT"How much do you bet";
4630 LOCATE 18,1:PRINT STRING$(40,32)
4680 LOCATE 18,1:PRINT"Discard how many (0 to 3)";
4720 IF HM=>4 THEN LOCATE 18,1:PRINT STRING$(40,32):GOTO 4680
4740   LOCATE 18,1:PRINT"Which ones (1 to 5, left to right)";
4780   LOCATE 18,1:PRINT STRING$(40,32)
4810 LOCATE 18,1:PRINT STRING$(40,32)
4850 LOCATE 18,1:PRINT"Do you (F)old, (C)all or (R)aise"
4870 IF Q$="F" THEN B1=3:M(I,6)=0:LOCATE 18,1:PRINT STRING$(40,32):
     GOTO 4360
4880 IF Q$="C" THEN B1=2:LOCATE 18,1:PRINT STRING$(40,32):GOTO 4350
4900 LOCATE 18,1:PRINT"How much do you wish to raise the bet";
4950 LOCATE 18,1:PRINT STRING$(50,32)
5040 FOR I=1 TO 7:PRINT"PLAYER ";I;" ";:PRINT USING "$$#,#####";M(I,
     10)-DO:NEXT I
5100 FOR I=1 TO 7
```

Puzzler # 9

NOTE: Good puzzlers are becoming scarce. If you have or know of some, please share them with the rest of us. The puzzler for this issue was sent in by Mr Richard Oberdorfer of Newport, Washington. Our thanks also to Mark Gardner of North Hollywood, California for his recent contributions.

Last issue's Puzzler asked how to insert string Y into string X at position P using the shortest code. Everyone who answered got it. It takes this general form:

X$=LEFT$(X$,P-1)+Y$+MID$(X$,P)

You could also have used RIGHT$ instead of MID$, but then you get into +RIGHT$(X$, LEN(X$)-P), which is a bit lumpy compared to the MID$ way to do it.

If you are inserting a word and it needs spaces around it you can enclose Y$ in quotes (with appropriate spaces around your word) when you input it. Otherwise, BASIC will only take the characters you type and even if you space after the INPUT Y$ prompt it doesn't see it.

Thanks to Col. C. F. Mowery, Jr. of Brandon, Florida, Peter W. Frye of Calais, Maine, Arthur M. Calhoun of Huntington, New York and others who took the time to write.

Mr. Calhoun also had what is possibly the best answer to the afterthoughts to Puzzler 8. He said, "My solution to the second part of Puzzler 8, wherein we want to list the differences between two versions of the same program, is this. Load MSDOS and type: FC PROG1 PROG2 >PRN. Try it, it works. See MSDOS version 2 manual, page 11.156 or MSDOS version 3 manual page 11.182 for details." He even included a sample run which compared our Poker program with a modified version. FC, by the way, stands for "File Compare" and is a utility included with most (but not all) MSDOS machines.

This is a very good solution, but does not address those who do not have FC or the half of our readers who do not use MSDOS. Several others sent in actual listings of file compare programs in BASIC. No one, as yet, has written with a *method* to do it in BASIC. In the meantime, we have been working on a program to do it, and it will be in these pages after we have polished it and there is space for it. We asked for method because once you have decided on the basic way to do something, the only thing left is to fill in the code around the core method. Sometimes, the method is very short and sounds simple, and the code grows out of sight.

Puzzler 9

Our next puzzler comes from Mr. Richard Oberdorfer, of Newport, Washington. We don't know if Mr. Oberdorfer is a science fiction buff or not. Astronomer Carl Sagan's recent book "Contact" had quite a bit to do with the value of Pi. If you are into science fiction, we highly recommend it. In any case, Mr. Oberdorfer sent the following code:

```
10 A=0:B=1
20 FOR X=1 TO 56
30    A=A+B
40    B=SQR(1+A^2)
50    G=3*2^(X+1)/(A+2*B)
60    PRINT G
70 NEXT X
```

It appears at first glance that this algorithm will generate the value of Pi to any number of decimal places. The question is: can it? If it can, then why? If it can't, then why? He states that he would be interested in hearing from anyone as to why this code does what it does. We have been playing with it and it is an intriguing problem. There are very many ways to generate Pi, but this is one we couldn't find in the books. Perhaps some of the math-minded computerists among you will find the answer to our question. ■

Qtext.Bas

A quick and simple text editor

Staff Project. For those of you who also wondered how Maker.Bas could be used as a text editor; here it is, a very simple and quick way to enter and edit text files.

Back in Issue 1 of CodeWorks we presented a program called Maker.Bas which generated data statements complete with line numbers and all the commas placed properly. We were just a little surprised when some of you wrote letters saying that the very letter you had written was done with Maker.Bas.

If you will recall, Maker.Bas was a BASIC program that actually wrote code that could be used in another BASIC program. Based on your input, we played around with the idea and have come up with Qtext.Bas, a quick (and easy) text editor.

The main thing this program has going for it is that if you have used your BASIC editor functions you already know all the editing commands to use with this editor - they are one and the same. The reason for this is simple: Qtext.Bas writes a BASIC program that is made up of LPRINT statements. When you enter your lines of text, the program creates a new program, complete with line numbers, containing the LPRINT statements containing your text. It even puts a remark line at line 10 to tell you what the name of your text is, and depending on how many lines you want on a printed page, will insert page eject commands at the proper place.

There is only one thing that you cannot do with this text editor that others handle with ease, and that is to print real quote marks. Because of the way the program works, quote marks signify the end of a literal statement and cannot be easily included in your printed text. It's not impossible, just very difficult and clumsy to handle. So in this program we go through the text you enter and change quote marks to single quotes (the apostrophe.)

After you have entered text with Qtext.Bas, you save it on disk with the .DOC extension. It can then be loaded like an ordinary BASIC program and you can edit it, if necessary, just like any other program. It doesn't make any difference which BASIC editor you are using. It will still work, even if you have one of the newer full-screen editors or the older Microsoft line editors. You can insert or delete lines, change words, or whatever before you run the program to print it out. You can even boiler-plate letters by editing in a new name and address on letters, for example, and using the same body text over and over.

The program does not have the features of most good text editors, but for many small jobs like dashing off a few letters it works very well. See Figure 1 for a small sample of how the beginning of this article would look using Qtext.Bas.

Program Details

The program clears string space (if you need it) in line 140. In 150, we dimension two single dimension arrays, A$() and Z$(). A$ will contain each line as you enter it and Z$() will be that same line with the line number and the LPRINT statement inserted in it. Sometimes, Z$() will only contain the line number and LPRINT" " (if you enter a blank line, for example.) At the place to page eject, Z$() will contain the line number and the LPRINT CHR$(12) statement to force the form feed.

When you tell the program what your document name is, do not use an extension. The program will automatically append .DOC to whatever name you give. In line 330, F$ becomes your document name. The .DOC extension is added in line 350. The number of lines on a printed page is entered in line 340 and becomes variable NL.

In line 360 we create the very first line of the program, a remark line which contains the name of the document. Back in line 170, we initialized L at 100 and IC at 10. Variable L is the starting line number for the document (after 10) and IC is the increment between line numbers. This should provide plenty of space between (or before) line numbers for you to edit in additional lines of text when needed.

The prompt character for this program is the open paren symbol. It lets you know that you are starting to type on a new line, and the symbol does not become part of the text stored in your .DOC file.

In line 390 we run into A$, and set it equal to a

space and LPRINT plus a quote. Don't confuse this A$ with the A$ array. They are two entirely different variables.

The little loop that controls this program is between lines 400 and 500. To start with, we set the loop to 1000 lines, more than we would probably be using, but we will jump out when we have had enough. Inside this I loop we must do certain things in a certain order. First, we want to print the prompt character and accept input into A$(I), which happens in line 410. Next, we must check to see if this is the end of input, and if so, jump out of the loop and close things up. This happens in line 420. If it is not the end of text then the loop from 430 to 450 is necessary to check each character in the A$(I) we just entered and find the quote marks in it (if there are any) and replace them with the single quote mark (CHR$(39)). Next, in line 460 we "build" our next line of code in the Z$ array. Line 460 says that Z$(I) will equal the string value of the line number, plus a space and LPRINT plus a quote mark (all previously defined in A$), plus the line of text we have just entered in A$(I). Line 460 might make a line looking like this: 100 LPRINT"Now is the time for all good men..."

In line 470 we check to see if what we had input in A$(I) is a null string (ENTER key only). If it was we need to create a line of code that will make a line feed on the printer without printing anything. So, still in line 470, we let Z$(I), which already contains a line number, space and the LPRINT" statement, equal to itself and add a space and another quote to it, giving 110 LPRINT" "

Next, in line 480, since we have used up a line number, we increment the line number by the variable IC, the increment, so it will be ready for the next A$(I) input text. We make one more check before going back to do it all over again. This is in line 490, where we find out if we have already printed as many lines on the page as were specified earlier in line 340 with variable NL. If we have, then we must make a line containing the page eject command, increment the line count (L) and also bump the loop counter (I) by one. We find out if we have the number of lines specified by NL by dividing the number of lines we have by NL and checking to see if there is a remainder. If there is no remainder we must have as many lines as specified by NL and so we go ahead and page eject, do the incrementing required and go on to the next I.

When we exit the loop (are done with text input) in line 420 we must type the word .END and the loop will then be broken and we go to line 510. In 510, because the loop counter is always one ahead of what was actually input (and we don't want the word .END to become part of our text) we subtract one from I and call that number N. Now, in order to make the last page (which may be just a partial page) eject like the others, we add the proper code to do that in line 520 and make that the last line of our .DOC file.

Our entire text (program) is now sitting in memory. If you now told the computer to loop from zero to N and print Z$(I), you would see your entire text, complete with line numbers and page ejects. Instead of doing that though, we are going to print it out to a disk file, just like any other program would be. This happens from lines 630 through 670.

Now that the text in .DOC is on disk with line numbers and all, you can load it like any other program and do whatever editing you want to it. You can RUN the program to print it out, and you can save it again with the changes you have made, or you can kill it after it has been printed. By the way, the .DOC extension is arbitrary, if it conflicts with other things you do, you can change it to whatever you like in lines 320 and 350. It would be a good idea to use a unique extension however, because that way if you do not want to keep notes and letters written with this program on disk, you can use the global form of kill to get them all at the same time. From BASIC you could say, for example: KILL"*.DOC" or from DOS, DEL *.DOC depending, of course on which machine you are using.

Changes for other Machines

If your BASIC is prior to version 5.1 you will need to un-remark line 140 to clear string space. If 5000 bytes is not enough, try clearing more.

The clear screen command (CLS) appears in line 180 and again in line 530. Most MBASICs running under CP/M will need to change this to whatever PRINT CHR$(##) you need to clear the screen and home the cursor.

Tandy model numbers less than 1000 should change the period in the file extension to a slash. The file extension appears in lines 320 and 350.

The program will run as is with MS-DOS and GW BASIC.

For all machines using the MOD function, you can change the first part of line 490 to: IF I MOD NL=0 THEN ... ■

```
10 REM * THE NAME OF THIS FILE IS: QTEXT.DOC
100 LPRINT"       Back in Issue 1 of CodeWorks we presented a
110 LPRINT"program called Maker.Bas which generated data
120 LPRINT"statements complete with line numbers and all the
130 LPRINT"commas placed properly. We were just a little surprised
140 LPRINT"when some of you wrote letters saying that the very
150 LPRINT"letter you had written was done with Maker.Bas.
160 LPRINT"       If you will recall, Maker.Bas was a BASIC
170 LPRINT"program that actually wrote code that could be used in
180 LPRINT"another BASIC program. Based on your input, we
190 LPRINT"played around with the idea and have come up with
200 LPRINT"Qtext.Bas, a quick (and easy) text editor.
210 LPRINT"       The main thing this program has going for it
220 LPRINT"is that if you have used your BASIC editor functions
230 LPRINT"you already know all the editing commands to use
240 LPRINT"with this editor - they are one and the same.
250 LPRINT" "
260 LPRINT"       (The above sample is from the first part of
270 LPRINT"this article. Here is how it handles 'quotes' in
280 LPRINT"a line.
```

```
        Back in Issue 1 of CodeWorks we presented a
program called Maker.Bas which generated data
statements complete with line numbers and all the
commas placed properly. We were just a little surprised
when some of you wrote letters saying that the very
letter you had written was done with Maker.Bas.
        If you will recall, Maker.Bas was a BASIC
program that actually wrote code that could be used in
another BASIC program. Based on your input, we
played around with the idea and have come up with
Qtext.Bas, a quick (and easy) text editor.
        The main thing this program has going for it
is that if you have used your BASIC editor functions
you already know all the editing commands to use
with this editor - they are one and the same.

        (The above sample is from the first part of
this article. Here is how it handles 'quotes' in
a line.
```

Figure 1

```
100 REM * QTEXT.BAS * A QUICK AND SIMPLE TEXT EDITOR * WRITTEN FOR
110 REM * CODEWORKS MAGAZINE 3838 S. WARNER ST. TACOMA, WA 98409
120 REM * 206 475-2219 VOICE AND 206 475-2356 300/1200 MODEM
130 REM * (c)1986 80-NW Publishing Inc. Placed in Public Domain 1987
140 'CLEAR 5000: ' Use only if your machine needs to clear string
    space
150 DIM A$(1000),Z$(1000)
160 DEFINT I,L,N
170 L=100:IC=10
180 CLS: ' Clear the screen and home the cursor
190 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
200 PRINT"                  Q U I C K   T E X T   E D I T O R
210 PRINT"    a simple Text Editor using commands already in Basic
```

```
220 PRINT STRING$(60,45)
230 PRINT
240 PRINT"  This simple text editor allows you to enter text lines"
250 PRINT"into what becomes a program which can then be edited"
260 PRINT"using the BASIC program editor.
270 PRINT"  You must press RETURN at the end of each input line.
280 PRINT"  Quote marks in the text will be changed to apostrophes.
290 PRINT"  Use a period and the word end  (.END or .end)"
300 PRINT"at the start of a new line to terminate entry of text."
310 PRINT
320 PRINT"This program will append the extension .DOC to your
    filename."
330 INPUT"What will you name this text file";F$
340 INPUT"How many printed lines do you wish per page.";NL
350 F$=F$+".DOC"
360 Z$(0)="10 REM * THE NAME OF THIS FILE IS: "+F$
370 PRINT"The prompt for a new line is the ( symbol."
380 PRINT
390 A$=" LPRINT"+CHR$(34)
400 FOR I=1 TO 1000
410   PRINT"(";:LINE INPUT A$(I)
420   IF A$(I)=".END" OR A$(I)=".end" THEN 510
430   FOR Q=1 TO LEN(A$(I))
440     IF MID$(A$(I),Q,1)=CHR$(34) THEN MID$(A$(I),Q,1)=CHR$(39)
450   NEXT Q
460   Z$(I)=STR$(L)+A$+A$(I)
470   IF A$(I)="" THEN Z$(I)=Z$(I)+CHR$(32)+CHR$(34)
480   L=L+IC
490   IF INT(I-NL*INT(I/NL))=0 THEN Z$(I+1)=STR$(L)+
      " LPRINT CHR$(12)":L=L+IC:I=I+1
500 NEXT I
510 N=I-1
520 Z$(N+1)=STR$(L)+" LPRINT CHR$(12)":N=N+1
530 CLS
540 PRINT"You must save the file on disk before it can be"
550 PRINT"recalled and edited or printed."
560 PRINT"Do you want to (S)ave the file on disk or,"
570 PRINT"abort and (R)erun the program."
580 PRINT"   Enter S or R"
590 INPUT"Your choice ";X$
600 IF X$="R" OR X$="r" THEN 100
610 IF X$="S" OR X$="s" THEN 630
620 GOTO 580
630 OPEN"O",1,F$
640 FOR I=0 TO N
650   PRINT #1,Z$(I)
660 NEXT I
670 CLOSE 1
680 PRINT
690 PRINT"You may now load the file like an ordinary BASIC"
700 PRINT"program and edit it using the BASIC editing commands."
710 PRINT"To print the text, simply ready your printer and
720 PRINT"RUN the program called ";F$
```

Smart.Bas

A program that learns and remembers

Steve Kelley and R. C. Bahn. This program starts with a blank memory and must learn that it is playing a game, the rules and how to win. Its learning can be saved from session to session. The side-panel on artificial intelligence was written by the CodeWorks staff.

After my curiosity was aroused by several articles about "smart games" - games that can actually learn from their mistakes, I began to wonder how far this idea could be taken. *(See Byte Magazine, Knowledge Representation, November 1986. - Ed)*

In this article I am exploring the idea of a program that can not only learn from mistakes, but must teach itself to play the game in the first place.

Artificially intelligent games have been around for some time now, and I imagine that many people have experimented with them. The programs I have seen in the past played very simple games, such as the one played with six chess pawns (Hexpawn) and would eventually become unbeatable. On looking at these game programs closely, however, I realized that while these programs could learn from their mistakes, they lacked the ability to learn to play the game in the first place. For the program to be able to play, every possible combination of board positions had to be loaded into an array with every possible subsequent move loaded into similar arrays. To pick a move, the computer would find its present position in the first array, then choose its move from the possible moves array. If this move resulted in the computer losing on the next move, the array element containing the losing move was zeroed, effectively deleting the move from the list of possible moves.

This is how the program was able to learn from its mistakes. Since each losing move was deleted from its list, the program would never make the same mistake twice. This may be the easiest way to implement a program of this nature, but it results in the programmer, rather than the computer, being faced with the task of calculating every possible board position and every possible corresponding subsequent move. Beside this being a lot of work, both the brain work and the keying in of many data statements, it also cheats for the computer, in that the program has complete familiarity with the game before it ever plays it.

My idea was a simple one: Have the computer start with a blank memory and build a table of moves based on its actual experience. Couple this with the ability to learn from its mistakes and you have a program that is not only easier to implement, but much more flexible as well. To illustrate this and to test my idea I needed a game with thousands of game board combinations that was still simple enough to be learned by the computer. Tic-Tac-Toe seemed a reasonable enough choice based on this criteria. It is simple, has many combinations, and has the added advantage that it is almost universally known.

The program to learn and play the game of Tic-Tac-Toe is quite simple and relatively short. In fact, the portion of the program involved with the actual decision making is one of the smallest. Most of the program merely handles the mechanics of displaying the game on the screen. The data statements contain the graphics characters (in the Tandy version) and data for a fast matrix manipulation routine. The latter is used to check to see if the game being played is a mirror image (in any direction) of some previous game. The only other decision-making routines are to determine if the user's move is valid and to check for a win after each move. Other routines are the timed input routine which starts the computer playing against its random number generator if you don't make a move for about 45 seconds. Other routines load and save the cumulative "learning" to disk.

The program will prompt you to load in data from previous games once at the beginning of the program. To save the current level of play simply enter "9999" as your move. The data will be saved and the program will prompt for your move again.

The computer's strategy is the simplest:

A) take the first empty square not zeroed in the array.

B) concede if they are all zero

C) never make the same mistake twice.

You will find after just a few games that the computer is very quick to learn defenses to your

(Continues on page 31)

Artificial Intelligence

At what point does a computer change from being naturally stupid and become artificially intelligent? Does it need to acquire a LISP to do it? Where has all this talk about expert systems and artificial intelligence come from? Is it something we can implement on our own personal computer? What can it do that normal computer processing cannot do?

The experts don't agree on what artificial intelligence is. They don't even agree on how to define intelligence, much less artificial intelligence. Minsky (Marvin Minsky, founder of the AI Lab at MIT) developed a test for artificial intelligence. He said that if you were communicating with another computer which you could not see, and could not tell if the other computer was operating on its own or being operated by a human, then that would be the real test for of artificial intelligence.

To get a feel for the difference between what we normally associate with computing and expert systems we first need to examine more closely what we do with computers now. Most computer programs contain an algorithm of one kind or another. That algorithm sets up a series of steps that will be taken when the program is run. You put in data that conforms to the range of data which the algorithm can accept and you get answers. Exactly. To the cent. Every time. If you don't get the expected answers, then there is good cause to believe that either the hardware or software has crashed or that the input was not in the prescribed range of allowable data. We are talking about expert systems here because we don't actually believe that there is such a thing as artificial intelligence.

An expert system is a computer program which has the accumulated expertise of a given area stored in it as "rules." In the parlance of AI, the human expert (or experts) are known as "domain experts," which is to say that they know a particular field very well. The "knowledge engineer" then, is the person who works with the domain expert to distill the essence of the expertise and convert it into the rules which will be used in the program. There are two types of rules, and a very fine distinction between them. One set of rules is called the "knowledge base" and the other set is called the "inference rules." Again, the AI people like to call the inference rules the "inference engine," presumably because these are the rules which form the logical steps needed to find an answer to a given problem. That sounds like an algorithm, doesn't it? In a way it is, but with a decided difference. Large expert systems may have as many as 500 inference rules, not all of which may be needed for a given problem. Further, those large systems have the ability to modify their own rules or add new ones. In other words, they have the ability to learn.

One of the earliest expert systems is called MYCIN, and was built to help medical people diagnose infectious blood diseases. It prescribes the proper antibiotics and can, when asked why, describe the reasoning behind its conclusions. Once the inference rules were installed in MYCIN its makers found that refining the rules produced vast improvement at first, and then as more refinement was added the improvement curve began to flatten out. They also found that the knowledge base could be completely stripped out of the system and a different knowledge base could be put in to make it an expert system in a different field, still using the same inference rules.

Expert systems can never know more than the experts whose knowledge they contain, but they tend to forget less. Further, they allow not-so-expert people to draw upon the accumulated knowledge of the experts.

Unlike programs which use algorithms, expert systems rarely give absolute answers. They are more apt to state: "There is a 95% certainty that the cause of the problem ... etc." They operate on heuristic (rule of thumb) rules, which work pretty well most of the time but without assurance that they will work every time. Expert systems, therefore, are never finished; you can continue to refine and improve them forever. They are a tinkerer's delight, and if you plan to get into one you should plan on spending inordinate amounts of time on it.

Is AI a misnomer? It probably is. A machine will never be able to possess the rich bed of life experience and emotion which we all draw on automatically. We don't even understand our own thinking processes well enough to endow a machine with them. Expert systems, although they are usually included in the same meaning as Artificial Intelligence, do have some commercial promise. It's a question of weighing the time and expense of putting one together against what benefits can be gained from it. ■

(continued from page 29)

particular strategy, even though its trial and error method makes it look pretty stupid. The computer is not programmed with any knowledge of the game, or with any strategy to win or keep from losing, only how to detect a win. This can be demonstrated by changing the win detection routine to check for three in a corner being the only way to win. The computer will learn to beat you just as fast this way or any other way as it did in the original game.

Food for Thought

The whole idea of artificial intelligence involves some fascinating concepts. One is that the program has the ability to learn to play better than its programmer. A not-so-simple case of the pupil passing the teacher. Another is that since the computer is learning to play from your play, it will compliment your particular strategies. If you continuously use your sure-fire best way to win, it will soon block you every time. Taking this a bit farther, after you can no longer beat the computer, let someone else who is pretty good at the game try. They probably will beat it, because the computer is only ready for your method of winning. Very soon though, the newcomer can no longer win, nor can anyone else who plays well.

Games Played vs Learning

While playing against the program, I learned a few things from the computer about Tic-Tac-Toe. For instance, when I first learned the "secret" of being able to beat anyone who didn't know the secret, I was told to start in a corner whenever I started first. The worst I've done this way was to tie. I didn't think it was possible to lose using this method knowing the counters I knew, even against an expert.

However, after playing by itself for quite a while, the computer always started in the center square. This can only mean that there is a way to be beaten every time for every other starting move! This was discovered after about 50,000 games played 24 hours a day while the computer was not in other use.

The statistics for the learning level (number of different games played) and the ability to be beaten are quite remarkable. When I could no longer beat the machine without considerable effort, the learning level was up to about 48. Left playing by itself the first night, the computer got the learning level up to 512, which completely filled the HASH table and stopped the program. This was totally unexpected, since we had played manually for about four hours and only got to 60, and especially since the computer can look at each game from so many angles to check for a similar game in memory. Final figures are 25,000 games played, learning level well over 600, games lost by the computer 2,500. These figures are for playing against the random generator, and would be much different if played against a human opponent (or another artificially intelligent computer?)

Practical Applications

So now it plays an unbeatable game; tomorrow the world? Not quite. I've tried everything from a self-adjusting inventory program to a routine to give advice on betting at draw poker, but the program just learns too slow. It took 25,000 Tic-Tac-Toe games to get good at that, so just picture an inventory controller that made 25,000 mistakes, or a poker advisor that gave you 25,000 bum steers.

So here is the tool for computer learning. It's not perfect, or even very practical right now. But I know there will come a time when we will see a program that will figure your income tax while automatically and dynamically readjusting itself to your exact needs, based on some kind of artificial intelligence.

Program Details
by R. C. Bahn

This is an artificial intelligence program which "learns" to play as described in the previous text. The program assumes a rather bureaucratic concept of success by primarily avoiding failure.

This discussion will primarily deal with the Tandy Model I and III version, however, considerable care was taken with the GW BASIC version so as not to disrupt the line numbering sequence.

Eventually, against a random opponent, the computer will win about 75 percent of the games and tie the remainder. This amounts to winning essentially all games in which the computer has the first move and winning half the games in which the computer has the second move. When the computer is adequately trained, usually all games will be tied against an alert, knowledgeable opponent.

The program provides two types of instructive activities. The first is the study of the automated "learning" process. The second is the study of the programming techniques.

To get to the study of the automated "learning" process, copy the listing of the program very

carefully. Every parenthesis, semicolon, comma, apostrophe, line number and line arrangement is important. Once the program is running, skip to the section of this article entitled "Evaluation of the Learning Process."

The program listing is relatively short but very compact. The flow of the program is circuitous. However, there are numerous comments which help to divide the program into logical modules.

To fully understand the programming techniques one should be familiar with the following topics: DATA statements, concatenation of string variables, subscripted variables, For...Next loops, logical IF statements, subroutines, RND statements, binary numbers, binary to decimal conversion and particularly Boolean logical operations.

Major Variables

Variables with beginning letters of A to H are string variables. The remaining variables (P through Z) are integers. These definitions can be found in line 50 of the program.

The most important string variable is A(N) which is formed in line 70. A(0) is a blank. A(1) is the screen symbol for "X", A(2) is the screen symbol for "O". These symbols are formed from the data associated with lines 960 to 990 in the data statements. In line 70, the individual numbers in the data are read as Z2. Also in line 70, each of these are concatenated as an ASCII character to form the respective A(N). To check this part of the program, copy lines 10-70 and lines 960 to 990. After you have run these few statements, press BREAK and type the following one-liner program in the direct mode using no statement number. FOR I=0 to 3:PRINT A(I):NEXT I. The three symbols will be displayed on the screen.

Several integer variables deserve description. QC is the number of games won by the computer. QH is the number of games won by the opponent. U1 is the number of new games studies. QG is the total number of games played. The number of ties is equal to QG-(QH+QC). QC, QG, QH and U1 are continuously displayed at the top of the screen.

Q1(I) is indexed 1 to 9 and consists of the screen addresses of the nine possible positions of the screen symbols. Q2(I) is indexed 1 to 8 and consists of the decimal representations of the binary coding of the eight winning positions. Q1(I) and Q2(I) appear in lines 1000 and 1010 respectively.

Q4(I,J) is an 8 x 9 array containing the eight possible positions for the tic-tac-toe figure if it were rotated 180 degrees around the vertical axis, the horizontal axis or the major and minor diagonal axes. This array is used in tests for geometrically similar moves. The data appears in lines 1020 to 1060.

The tic-tac-toe figure is composed of nine positions. Unique coding of the occupancy of nine squares numbered 0, 1, 2, 3, 4, 5, 6, 7, 8 can be obtained by assigning each the value of its respective power of two, i.e., 1, 2, 4, 8, 16, 32, 64, 128, 256 and then adding the assigned numbers of the occupied squares. The greatest number will be two to the ninth, or 511. The powers of 2 are stored in Q5(I) and appear in line 1080 of the data. It is more rapid and precise to look up the powers of two than to compute them each time they are needed.

Within the program the decimal coded binary information is used with the array B1(I,J). This matrix is dimensioned in line 60 of the program (B1(2,1023)). In the first dimension, 0, 1, and 2 are all used. The index of the second dimension is formed as Q in line 460 and consists of one plus the sum of coded occupancies by the first and second move players. The latter data is stored in BP(1) and BP(2) respectively, and updated during the play of the game.

The body of data within B1(I,J) is first generated for a specific value of Q in line 580. Note that P1(1,Q)=PB(1) and P1(2,Q)=PB(2). Finally, P1(O,Q)=PB(2). This is a logical OR. The data within P1(I,J) is later modified by OR and AND logic in lines 650 and 730 upon the events of losing or non-winning moves, respectively.

Throughout the program, arrays P1(I,J) and Q5(I) are queried by IF statements using Boolean operators to determine legal moves, prior similar moves, next allowable moves, and winning positions. These maneuvers occur in lines 280, 400, 450, 490, 510, 520, 550, 580, 590, 600, 620, 650 and 840.

The use of Boolean operators is elegant and rapid. You will have mastered a significant area of computer science by understanding them. Consider line 280. A specific player's combination of moves is stored in PB(Z1). A winning combination is stored in Q2(Z). The question is asked: Is the winning combination of Q2(Z) a member of the set of positions in PB(Z1)? If the answer is yes, the logical AND will return the number stored in Q2(Z), otherwise some other number or zero will be returned. In essence, the logical AND returns members, or in generalized terms, the binary bits which are in common.

The logical OR is illustrated in line 580. A similar move was not found. U1 was incremented. Next, P1(O,Q)=PB(1) OR PB(2). P1(O,Q) now contains a record of the occupancy of the board irrespective of first or second move. In essence, the

logical OR returns members, or in generalized terms, the binary bits, which occur in the first or the second or both tested configurations. Since occupancy of the tic-tac-toe board is mutually exclusive between players, simple addition would result in the same outcome in this situation, but not necessarily in the other uses of the logical OR within this program.

Evaluation of the Automated "Learning Process"

For any automated "learning" process, one would like to know the expected limits of performance and the time necessary to actually achieve a performance which is close to the expected limits. The beginning of such an evaluation consists of collecting the raw data which is displayed on the first line of the video screen. One could surmise that very little additional "learning" would occur if the level of play and the number of games won by the user did not change for many iterations.

In addition to these criteria, one can construct a standardized "learning curve" which will allow you to more easily compare various experiments you might conduct. The total number of trials (QG), the number of wins by the computer (QC) and the number of wins by the opponent (QH) could continue to increase.

The percentage of games won by the computer (QC/QG)*100, the percentage of games won by the opponent (QH/QG)*100 and the percentage of ties (1-((QC+QH)/QG))*100 will behave in a more controlled fashion. The percentages will sum to 100 percent. The percentage of games won by the opponent will approach zero. The percentage of ties appears to approach about 25 percent. The percentage of games won by the computer appears to approach about 75 percent. These limiting long-term values or asymptotes are only approximations, since they were not rigorously obtained by theoretical analysis of the problem, but rather determined by observation of two experiments, one running for 12,401 iterations and the other running for 67,626 iterations.

You can study your own computer's learning curve by tabulating the raw data at appropriate intervals, computing the percentages for each interval and plotting the percentages on the Y axis against the total number of trials on the X axis. This might even be a separate programming problem.

Expected performance values can be computed as a function of QT, the total number of trials. An empirical expected average learning curve for percentages of ties can be approximated by the function YT=100*(0.25*(1-EXP(A*QT))). For percentages of computer wins YC=100*(0.5+0.5*(EXP(B*QT))). For percentages of human wins YH=100-YT- YC. In the two preliminary experiments, the coefficient A was about -0.005 (-0.002 to -0.008). The coefficient B was about -0.007 (-0.006 to -0.008). A and B reflect in one number the rate of "learning."

With these procedures you are now ready to evaluate the "learning" process. The greatest action occurs during the first 2,000 iterations. Against a purely random opponent the experimental curve will not exactly follow the average expected curve but will develop plateaus. Different beginnings may be obtained by randomizing the seed of the random number generator during program initialization.

What is the effect of early or late training by a human opponent? What is the last "trick move" to be defensed by the computer after prolonged random learning? How many random trials are enough? How many human trials against an expert opponent are enough?

Divide your experiments into two states and call the information in the first stage the training set and the information in the second stage the test set. During the training set, tabulate the raw data at appropriate intervals in the usual manner. Continue the process during the test set but record the raw data at the end of the training set and subtract these numbers from the raw data displayed at the end of various intervals during the test set. Compute the performance percentages of the test set on the basis of these differences. Can human play as a training set provide all the information necessary for maximum winnings against a random opponent in a test set? How can the program be improved to make the percentage of computer wins against a random opponent approach greater than 90 percent and still maintain the philosophy of the program that the list of specific winning combinations is not directly accessible to the decision-making procedure?

This program was written very efficiently and runs at about 240 to 300 iterations per hour. Nevertheless, the generation of over 1,000 iterations becomes very tedious. The program can be compiled. The prolonged experiments described earlier were performed by a compiled program at a rate of 1,500 to 2,000 iterations per hour. The Boolean logic within the program is ideally suited for this maneuver. The result was a six to eight fold increase in the speed of execution of the program.

In summary, these analytic techniques begin to

provide methods for answering the following questions. How does one describe the behavior of an automated learning process? How can one study the effects of varied procedural interventions? How can one determine the "best" algorithm to accomplish the desired objective of the learning process? ■

This is the Tandy Model I/III version of Smart.Bas. It will also work on a Model IV running in Model III mode.

Additional notes: When it is your turn to move and you do not respond for about 45 seconds, the computer will start to play against itself using the random number generator. It learns better if you play it though, because the random generator makes some very funny moves. If you have let it play itself, you can jump back in by pressing the ENTER key. Also, when it is your turn, you can enter 9999 and the "smarts" will be saved to disk. When you run the program the next time you can load what it has already learned and continue.

```
10 'SMART3/TRS * FOR TANDY MODELS I & III *
20 'BY STEVE KELLEY
30 CLS:PRINT TAB(18);"SMART-TAC-TOE"
40 CLEAR 999
50 DEFINT P-Z: DEFSTR A-H
60 DIM P(9),Q1(9),Q2(8),Q3(2),Q4(8,9),Q5(9),A(2),P1(2,1023)
70 FORZ=0TO2:FORZ1=1TO10:READZ2:A(Z)=A(Z)+CHR$(Z2):NEXTZ1,Z
80 FOR Z=1TO9:READ Q1(Z):NEXTZ:E1=CHR$(30)
90 FOR Z=1 TO 8:READ Q2(Z):NEXT Z
100 FOR R2=1 TO 8:FORX=1 TO 9:READ Q4(R2,X):NEXT X,R2
110 FOR X=1 TO 9:READ Q5(X):NEXT X:QG=-1
120 PRINT:PRINT"USE DATA ON DISK ? 1=YES, 2=NO ";:GOSUB 880
130 IF VAL(A)<> 1 THEN QS=0:GOTO 160
140 OPEN "I",1,"DATATAC/TXT":INPUT #1,QG,QH,QC,U1
150 FOR R=0 TO 1023:INPUT#1,P1(0,R),P1(1,R),P1(2,R):NEXT R:CLOSE
160 PRINT:PRINT"ENTER"+CHR$(34)+"9999"+CHR$(34)+"AS MOVE TO RECORD
    NEW DATA TO DISK."
170 INPUT "HIT ENTER TO CONTINUE...";C
180 '                **NEW GAME**
190 QG=QG+1:Q3(1)=0:Q3(2)=0:FORX=1 TO 9:P(X)=0:NEXT X: PB(1)=0:
    PB(2)=0
200 '                **MAIN LOOP**
210 IF POINT(21,26) <> 0 THEN 250
220 CLS:FORZ=1 TO 60:SET (Z+20,14):SET(81-Z,26)' **DRAW CROSSHATCH**
230 SET(60,Z/1.9+5):SET(40,37-Z/1.9):NEXT Z
240 PRINT@183, "1 2 3";:PRINT@247, "4 5 6";:PRINT@311, "7 8 9";
250 FOR X=1 TO 9:PRINT@Q1(X),A(P(X));:NEXT X ' **PRINT PIECES**
260 PRINT@0,"GAME #:";QG;"   I WON:";QC;"   YOU WON:";QH;"   LEVEL:";
    U1;
270 FORZ1 = 1 TO 2:FOR Z=1 TO 8 ' **CHECK FOR WIN**
280 IF (PB(Z1) AND Q2(Z))=Q2(Z)THEN Q3(Z1)=Z
290 NEXT Z,Z1:ON SGN (Q3(1))+SGN(Q3(2))*2 GOTO 650,730
300 IF PB(1)+PB(2)=511 THEN 790'  **CHECK FOR DRAW**
310 IF QT=0THEN QT=1 ELSE QT=0:GOTO 430
320 '                ***HUMAN'S TURN***
330 C=INKEY$:IF C <>""THEN QS=0 '**RETURN CONTROL TO KEYBOARD?**
340 PRINT@896,E1+"MAKE YOUR MOVE  ";
```

```
350 S=RND(9):IF QS=0THEN GOSUB 880:S=VAL(A)
360 IF S <> 9999 THEN 390 ' *CODE FOR RECORD ON DISK?*
370 OPEN "O",1,"DATATAC/TXT":PRINT#1,QG;QH;QC;U1
380 FOR R=0 TO 1023:PRINT#1,P1(0,R);P1(1,R);P1(2,R):NEXT R:CLOSE
390 IF S<1 OR S>9 THEN 330
400 IF P(S)=0 THEN P(S)=1:PB(1)=PB(1) OR Q5(S):GOTO 210
410 PRINT@896,E1+"NO NO NO...";:FOR Z=1 TO 500*(1-QS):NEXTZ:GOTO 330
420 '                    ***COMPUTER'S TURN***
430 PRINT@896,E1+ "MY TURN  ";
440 R2=8
450 PB(1)=0:PB(2)=0:FOR X=1 TO 9:PB(P(Q4(R2,X)))=PB(P(Q4(R2,X)))OR
    Q5(X):NEXT X
460 Q=PB(1)+PB(2)+1:R1=2*Q+1'  **GENERATE HASH CODE**
470 IF PB(1)=P1(1,Q) AND PB(2)=P1(2,Q) THEN 500
480 IF P1(1,Q)=0 AND Q<> 1 THEN R2=R2-1:IF R2>=1 THEN 450 ELSE 580
490 Q=(Q+R1) AND 1023:GOTO 470
500 IF P1(0,Q)=511 THEN 650'  **CONCEDE IF NO MOVES**
510 '
520 FOR Z=1 TO 9:IF(P1(0,Q) AND Q5(Z))=0 THEN X=Z:Z=9
530 NEXT Z
540 P(Q4(R2,X))=2:Q3=Q:Q2=X'  *PUT O IN CURRENT POSITION**
550 PB(1)=0:PB(2)=0:FORX=1 TO 9:PB(P(X))=PB(P(X)) OR Q5(X):NEXT X
560 GOTO 210
570 '                *SIMILAR MOVE NOT FOUND*
580 U1=U1+1:P1(0,Q)=PB(1) OR PB(2):P1(1,Q)=PB(1):P1(2,Q)=PB(2)
590 '
600 FOR Z=1 TO 9:IF (P1(0,Q) AND Q5(Z))=0 THEN X=Z:Z=9
610 NEXT Z
620 P(X)=2:PB(2)=PB(2) OR Q5(X):Q3=Q:Q2=X
630 GOTO 210
640 '                *COMPUTER LOSE*
650 P1(0,Q3)=P1(0,Q3) OR Q5(Q2)'     **MASK OFF LOSING MOVE**
660 QH=QH+1
670 GOSUB 830
680 PRINT@896, E1+"I LOSE";:IF QH<>1 THEN PRINT" AGAIN";
690 IF U1>10 PRINT" -BUT I'M LEARNING";
700 IF QS=0 THEN GOSUB 880
710 GOTO 190
720 '                   *HUMAN LOSE*
730 P1(0,Q3)=(NOT(Q5(Q2))) AND 511'  *MASK OFF NON-WINNNING MOVES**
740 QC=QC+1
750 GOSUB 830
760 PRINT@896,E1+"I WIN";:IF QC<>1 THEN PRINT" AGAIN";
770 GOTO 700
780 '                     **KAT'S GAME**
790 IF QC>QH THEN C = "YOU" ELSE C= "I"
800 PRINT@896, E1+"AT LEAST "+C+" DIDN'T LOSE!!";
810 GOTO 700
820 '                    **DISPLAY WIN**
830 FOR Z = 1 TO 7*(1-QS)
840 FOR X = 1 TO 9:IF (Q5(X) AND Q2(Q3(1)+Q3(2)))<>0 THEN PRINT@Q1(X),
    A(0);
```

```
850 NEXT X:FOR X=1 TO 9:PRINT@Q1(X),A(P(X));:NEXT X
860 NEXT Z:RETURN
870 '                          **TIMED INPUT ROUTINE**
880 A="":PRINT CHR$(14);
890 FOR Z=1 TO 3000:B=INKEY$:IF B<>"" THEN Z=10000
900 NEXT Z:IF B="" THEN QS=1:GOTO 950
910 IF B=CHR$(8) AND LEN(A)>0 THEN A=LEFT$(A,LEN(A)-1):PRINT B;:GOTO
    880
920 IF B=CHR$(13) THEN PRINT:GOTO 950
930 IF ASC(B)>=48 AND ASC (B) <=57 THEN PRINT B;:A=A+B
940 GOTO 890
950 PRINT CHR$(15);:RETURN
960 '                    **DATA FOR CHARACTERS**
970 DATA 32,32,32,26,24,24,24,32,32,32
980 DATA 137,152,129,26,24,24,24,134,130,132
990 DATA 151,131,149,26,24,24,24,141,140,133
1000 DATA 142,152,162,398,408,418,654,664,674
1010 DATA 7,56,448,73,146,292,273,84
1020 '                      **DATA FOR MATRIX INVERSIONS**
1030 DATA 1,2,3,4,5,6,7,8,9,3,2,1,6,5,4,9,8,7
1040 DATA 1,4,7,2,5,8,3,6,9,3,6,9,2,5,8,1,4,7
1050 DATA 9,8,7,6,5,4,3,2,1,7,8,9,4,5,6,1,2,3
1060 DATA 7,4,1,8,5,2,9,6,3,9,6,3,8,5,2,7,4,1
1070 '                     *DATA FOR POWERS OF TWO**
1080 DATA 1,2,4,8,16,32,64,128,256
```

Smart.Bas for machines running GW BASIC and MSDOS.

```
10 'SMART.BAS * FOR MSDOS GW BASIC MACHINES *
20 'BY STEVE KELLEY
30 CLS:PRINT TAB(18);"SMART-TAC-TOE"
40 KEY OFF
50 DEFINT P-Z: DEFSTR A-H
60 DIM P(9),Q1(9),Q2(8),Q3(2),Q4(8,9),Q5(9),A(2),P1(2,1023)
70 A(0)=" ":A(1)="X":A(2)="O":'FOR Z=0 TO 2:FOR Z1=1 TO 10:READ Z2:
   A(Z)=A(Z)+CHR$(Z2):NEXT Z1,Z
80 FOR Z=1 TO 9:READ Q1A(Z):READ Q1B(Z):NEXT Z:E1=CHR$(30)
90 FOR Z=1 TO 8:READ Q2(Z):NEXT Z
100 FOR R2=1 TO 8:FOR X=1 TO 9:READ Q4(R2,X):NEXT X,R2
110 FOR X=1 TO 9:READ Q5(X):NEXT X:QG=-1
120 PRINT:PRINT"USE DATA ON DISK ? 1=YES, 2=NO ";:GOSUB 880
130 IF VAL(A)<> 1 THEN QS=0:GOTO 160
140 OPEN "I",1,"DATATAC.TXT":INPUT #1,QG,QH,QC,U1
150 FOR R=0 TO 1023:INPUT#1,P1(0,R),P1(1,R),P1(2,R):NEXT R:CLOSE
160 PRINT:PRINT"ENTER"+CHR$(34)+"9999"+CHR$(34)+"AS MOVE TO RECORD
    NEW DATA TO DISK."
170 INPUT "HIT ENTER TO CONTINUE...";C
180 '                **NEW GAME**
190 QG=QG+1:Q3(1)=0:Q3(2)=0:FOR X=1 TO 9:P(X)=0:NEXT X: PB(1)=0:
    PB(2)=0
```

```
200 '                **MAIN LOOP**
210 REM check for game board' IF POINT(21,26) <> 0 THEN 250
220  CLS'FORZ=1 TO 60:SET (Z+20,14):SET(81-Z,26)' **DRAW CROSSHATCH**
221 'GOTO 245
230 LOCATE 12,41:PRINT"     |   |";
232 LOCATE 13,41:PRINT" ---+---+---  " ;
234 LOCATE 14,41:PRINT"     |   |     ";
236 LOCATE 15,41:PRINT" ---+---+---";
238 LOCATE 16,41:PRINT"     |   |  ";
240 '
242 '
245 LOCATE 12,20:PRINT "1 2 3";:LOCATE 13,20:PRINT"4 5 6";:LOCATE 14,
    20:PRINT "7 8 9";
250  FOR X=1 TO 9:LOCATE Q1A(X),Q1B(X):PRINT A(P(X));:NEXT X '
     **PRINT PIECES**
260 LOCATE 1,1:PRINT"# GAMES: ";QG;"   I WON:";QC;"   YOU WON:";QH;"
     LEVEL OF PLAY:";U1;
270 FOR Z1 = 1 TO 2:FOR Z=1 TO 8 ' **CHECK FOR WIN**
280 IF (PB(Z1) AND Q2(Z))=Q2(Z)THEN Q3(Z1)=Z
290 NEXT Z,Z1:ON SGN (Q3(1))+SGN(Q3(2))*2 GOTO 650,730
300 IF PB(1)+PB(2)=511 THEN 790'  **CHECK FOR DRAW**
310 IF QT=0 THEN QT=1 ELSE QT=0:GOTO 430
320 '                ***HUMAN'S TURN***
330 C=INKEY$:IF C <>""THEN QS=0 '**RETURN CONTROL TO KEYBOARD?**
335 LOCATE 22,2:PRINT "                              ";
340 LOCATE 22,2:PRINT "MAKE YOUR MOVE  ";
350 S=INT(RND(9)*10):IF QS=0 THEN GOSUB 880:S=VAL(A)
360 IF S <> 9999 THEN 390 ' *CODE FOR RECORD ON DISK?*
370 OPEN "O",1,"DATATAC.TXT":PRINT#1,QG;QH;QC;U1
380 FOR R=0 TO 1023:PRINT#1,P1(0,R);P1(1,R);P1(2,R):NEXT R:CLOSE
390 IF S<1 OR S>9 THEN 330
400 IF P(S)=0 THEN P(S)=1:PB(1)=PB(1) OR Q5(S):GOTO 250
405 LOCATE 22,2:PRINT "                             ";
410 LOCATE 22,2:PRINT "NO NO NO...";:FOR Z=1 TO 500*(1-QS):NEXT Z:
    GOTO 330
415 STOP
420 '                    ***COMPUTER'S TURN***
430 LOCATE 22,2:PRINT "                             ";
435 LOCATE 22,2:PRINT "MY TURN  ";
440 R2=8
450 PB(1)=0:PB(2)=0:FOR X=1 TO 9:PB(P(Q4(R2,X)))=PB(P(Q4(R2,X)))OR
    Q5(X):NEXT X
460 Q=PB(1)+PB(2)+1:R1=2*Q+1'  **GENERATE HASH CODE**
470 IF PB(1)=P1(1,Q) AND PB(2)=P1(2,Q) THEN 500
480 IF P1(1,Q)=0 AND Q<> 1 THEN R2=R2-1:IF R2>=1 THEN 450 ELSE 580
490 Q=(Q+R1) AND 1023:GOTO 470
500 IF P1(0,Q)=511 THEN 650'  **CONCEDE IF NO MOVES**
510 '
520 FOR Z=1 TO 9:IF(P1(0,Q) AND Q5(Z))=0 THEN X=Z:Z=9
530 NEXT Z
540 P(Q4(R2,X))=2:Q3=Q:Q2=X'  *PUT O IN CURRENT POSITION**
550 PB(1)=0:PB(2)=0:FOR X=1 TO 9:PB(P(X))=PB(P(X)) OR Q5(X):NEXT X
560 GOTO 250
```

```
570 '                  *SIMILAR MOVE NOT FOUND*
580 U1=U1+1:P1(0,Q)=PB(1) OR PB(2):P1(1,Q)=PB(1):P1(2,Q)=PB(2)
590 '
600 FOR Z=1 TO 9:IF (P1(0,Q) AND Q5(Z))=0 THEN X=Z:Z=9
610 NEXT Z
620 P(X)=2:PB(2)=PB(2) OR Q5(X):Q3=Q:Q2=X
630 GOTO 210
640 '                  *COMPUTER LOSE*
650 P1(0,Q3)=P1(0,Q3) OR Q5(Q2)'    **MASK OFF LOSING MOVE**
660 QH=QH+1
670 GOSUB 830
675 LOCATE 22,2:PRINT "                                                ";
680 LOCATE 22,2:PRINT " I LOSE ";:IF QH<>1 THEN PRINT" AGAIN ";
690 IF U1>10 THEN PRINT" -BUT I'M LEARNING";
700 IF QS=0 THEN GOSUB 880
705 CLS
710 GOTO 190
720 '                    *HUMAN LOSE*
730 P1(0,Q3)=(NOT(Q5(Q2))) AND 511'  *MASK OFF NON-WINNNING MOVES**
740 QC=QC+1
750 GOSUB 830
755 LOCATE 22,2:PRINT "                                                ";
760 LOCATE 22,2:PRINT " I WIN";:IF QC<>1 THEN PRINT" AGAIN";
770 GOTO 700
780 '                      **KAT'S GAME**
785 LOCATE 22,2:PRINT "                                                ";
790 IF QC>QH THEN C = "YOU" ELSE C= "I"
800 LOCATE 22,2:PRINT"AT LEAST "+C+" DIDN'T LOSE!!";
810 GOTO 700
820 '                     **DISPLAY WIN**
830 FOR Z = 1 TO 7*(1-QS)
840 FOR X = 1 TO 9:IF (Q5(X) AND Q2(Q3(1)+Q3(2)))<>0 THEN LOCATE
    Q1A(X),Q1B(X):PRINT A(0);
850 NEXT X:FOR X=1 TO 9:LOCATE Q1A(X),Q1B(X):PRINT A(P(X));:NEXT X
860 NEXT Z:RETURN
870 '                          **TIMED INPUT ROUTINE**
880 A="":
890 FOR Z=1 TO 3000:B=INKEY$:IF B<>"" THEN Z=10000
900 NEXT Z:IF B="" THEN QS=1:GOTO 950
910 IF B=CHR$(8) AND LEN(A)>0 THEN A=LEFT$(A,LEN(A)-1):PRINT B;:GOTO
    880
920 IF B=CHR$(13) THEN PRINT:GOTO 950
930 IF ASC(B)>=48 AND ASC (B) <=57 THEN PRINT B;:A=A+B
940 GOTO 890
950 RETURN
960 '                  **DATA FOR CHARACTERS**
1000 DATA 12,43,12,47,12,51,14,43,14,47,14,51,16,43,16,47,16,51
1010 DATA 7,56,448,73,146,292,273,84
1020 '                    **DATA FOR MATRIX INVERSIONS**
1030 DATA 1,2,3,4,5,6,7,8,9,3,2,1,6,5,4,9,8,7
1040 DATA 1,4,7,2,5,8,3,6,9,3,6,9,2,5,8,1,4,7
1050 DATA 9,8,7,6,5,4,3,2,1,7,8,9,4,5,6,1,2,3
1060 DATA 7,4,1,8,5,2,9,6,3,9,6,3,8,5,2,7,4,1
1070 '                   *DATA FOR POWERS OF TWO**
1080 DATA 1,2,4,8,16,32,64,128,256
```

CodeWorks 1st Year Programs on Diskette

All the programs published during the 1st year of CodeWorks magazine are available on the following 5¼ soft-sector formats:

- ☐ PC/MS-DOS GW-BASIC 40 track DSDD
- ☐ CP/M MBASIC specify format and computer type:
- ☐ TRSDOS Model I 35 track SDSS
- ☐ TRSDOS Model III 1.3 40 track SSDD
- ☐ TRSDOS IV 6.2 40 track SSDD

If your format is not covered by the above, please inquire and we will try to accommodate you.

Only 50¢ per program!

Check "Disk Order" in the order form below and return this page or a photocopy to us. Diskettes will be sent 1st Class postpaid.

Special Subscriber price $20.00

Note: We cannot provide hard-sector formats. Non-subscriber price $30.00

Subscription ORDER FORM

187

NOTE: The entire set of (7) first year issues is still available at $24.95. Please specify "First Year" if you are ordering this set.

Computer type:

Comments:

Please enter my one year subscription to *CodeWorks* at $24.95. I understand that this price includes access to download programs at NO EXTRA charge.

- ☐ New subscription
- ☐ Renewal subscription
- ☐ Check or MO enclosed.
- ☐ Bill me later.
- ☐ Charge to my VISA/MasterCard # _____ Exp. date _____
- ☐ **Diskette Order**

Please Print Clearly:

Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to: ***Codeworks***
3838 South Warner St.
Tacoma, WA 98409

Charge card orders may be called in (206) 475-2219 between 9 AM and 4 PM weekdays, Pacific time.

Download

What's Happening on the Download

The download system seems to be settling down to a smooth purr. We just completed a survey of what happened during the month of September 1986. Just over 1,200 calls were logged for that month. Usage was markedly higher at night and on week-ends. Usage also peaks every other month, when the new issue comes out. During September, the most often downloaded programs were the NFL programs and since then it has been the stats for NFL.

The number of downloads is going up compared to the problem messages being left, which indicates that something must be working out. There are still some of you who think that you will be issued a "secret password." Most of you already know that you assign your own password.

Here is a cross-section of the comments that have been left on the board:

"I have really enjoyed downloading programs. Yours is one of the easiest boards I've used. Stephen Hancock."

"Help! Either I have not remembered the password or I've been deleted. How do I clear my record and start with a new password? *(When we receive a message like that we first check to see that you are on our subscriber list, then we reset you so that the next time you log on it will ask you to assign yourself a new password. As of right now, no one has been deleted from access to the board. However, those who have not renewed their subscriptions by January 1st will be removed then.)*

"Tried to renew subscription but could not seem to get to the DEMO. Is there any way to determine when subscriptions expire from the address label?" *(Golly, it used to be that people couldn't find their way to the subscriber side of the board, now it seems to be the other way around. To get to the DEMO side, just enter anything when you are asked to log on other than your correct name and number. No, there is no way to tell expiration from your label. Everyone's subscription runs out with the Sep/Oct issue of each year.)*

"Would you please reset me so that I can continue to run up my phone bill."

"Thank you for the program Check.Bas...that program was worth the cost for the entire year. Are you considering a disk for sale?" (Thank you, and yes, the disk is available. See the other side of this very page.)

"Thought you might like to know that there is a typo in NFL86, line 1340 LPRINT ...B(X1,5) should be B(X1,6). Ron Chiodo" *(When we first ported NLF86 over to the download it somehow missed line 1340 entirely. We then entered it manually, on the download system itself, and made that transposition. It was fixed as soon as we read your message.)*

"For a good time call....." *(Well, I suppose we had to get that one sooner or later.)*

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
US Postage
PAID
Permit No. 774
Tacoma, WA

CodeWorks

Issue 10 March/April 1987

CONTENTS

CodeWorks

Issue 10 Mar/Apr 1987

Editor/Publisher
Irv Schmidt
Associate Editor
Terry R. Dettmann
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Advisor
Al Mashburn

Editor's Notes

We're almost half way through the second year of CodeWorks. Perhaps it's time for a "State of the Magazine" address. Here goes.

Readers, subscribers, followers of CodeWorks: During the past few months renewals have continued to trickle in. We now have 70 percent of the readers we had at the end of our first year. There are now 1,600 of you who have shown your loyalty for a second year and we deeply appreciate that. Your enthusiasm is contagious; there is no mistake about it. Since we are a small group, we feel we can almost address each of you individually. In most cases, when you write or call, you receive at least somewhat prompt attention. We are sure you appreciate that, and we enjoy doing it.

The reasons given by many for not renewing their subscriptions varied from "too simple" to "too complicated." This, at least, tells us that we are reaching the majority of you between those two extremes. It is not that such criticism goes unheard. Some of our beginner material was fast leaving them behind. We are working on that now. On the other hand, we take the "too simple" criticism as a compliment: good code that does something well always looks simple.

Diskette sales, and the end of the NFL season, seem to have caused a dramatic decrease in the use of the Download System. Although we hadn't planned it that way, it turned out that revenue otherwise received by the phone companies came to us instead. The Download System (phone line and equipment leases) costs us about $200 per month to maintain. Even though some subscriptions, renewals and requests for sample copies are received on it each month, it does not begin to pay its own way. In all fairness, however, the Download System was acquired for a project that never got off the ground before CodeWorks was ever started. So the Download stays.

Given the present level of subscribers and the revenue received through them, it is questionable we will be able to publish for a third year. That sounds terrible! So we are determined not to let it happen. We will mount at least one (maybe two, finances permitting) promotion drive between now and November. We know that potential readers are out there, it's just very expensive to find them.

Now, as they say in the TV ads:

But Wait! There's more!

You can help. It's easy and simple, and there's something to be gained for you too (aside from continuing the magazine.) Here is how it works:

On our order form on page 39 you will find a new line added that says "Referred by _____" and a place for your subscriber number. You put your own name and number there, make photo copies of the order form and give it to people who might be interested. When we get just two such orders with your name on them, we will reserve a FREE diskette with all the second year programs on it for you. It will automatically be sent to you during November 1987. If you didn't get the first year diskette and want that instead, then we can do it that way too. And, since this is not a one-shot deal, the two people you sign up will also have a chance at their own free disk.

Give it some thought. By working on this from both ends, we can keep a good thing going and make it better for everyone. Thank you again for your continued support.

Irv

Forum

An Open Forum for Questions & Comments

Had enough NFL statistics? We have too, and look forward to letting it rest for awhile. Meantime, however, here are just a couple more to sort of wrap up the season.

Figure 1 is a list of the teams and how many times we picked them correctly (to either win or lose.) It turns out that the NFL86 Oracle only picked 63.2 percent, not quite the 66 percent we had hoped it would do.

Figure 2 is a ranking of the teams based on the average number of points the team scored over its opponents divided by the average number of points the opponents scored over them. The second column is the number of games actually won (or tied.) We found this one to be quite interesting and after a respectable breather, will try to incorporate something like this into the projections for next year, provided it shows some significance as the season progresses.

Many of you have asked for some way to project the playoffs. This would be difficult because of the totally unpredictable schedule. Who knows who will be there, and when they play, ahead of time? This does not mean it is impossible or that we won't try it. It could even end up as a separate playoff program, which seems to make more sense right now than trying to incorporate it into the existing program. Any ideas you may have on this subject would be appreciated.

Saints	11
Colts	11
Bears	11
Patriots	10
Packers	10
Oilers	10
Niners	10
Chargers	10
Cards	10
Bucs	10
Steelers	9
Jets	9
Cowboys	9
Seahawks	8
Redskins	8
Giants	8
Vikings	7
Rams	7
Lions	7
Falcons	7
Eagles	7
Dolphins	7
Bills	7
Bengals	7
Browns	6
Chiefs	5
Broncos	5
Raiders	4

Figure 1

Giants	14	17.88
Bears	14	09.68
Niners	11	05.23
Vikings	9	03.31
Patriots	11	03.10
Browns	12	02.88
Redskins	12	02.71
Rams	10	01.84
Seahawks	10	01.78
Broncos	11	01.70
Chiefs	10	01.54
Dolphins	8	01.26
Bengals	10	01.14
Cowboys	7	01.09
Saints	7	01.01
Falcons	8	01.00
Jets	10	00.84
Steelers	6	00.76
Raiders	8	00.69
Oilers	5	00.55
Chargers	4	00.53
Eagles	6	00.49
Lions	5	00.47
Bills	4	00.29
Packers	4	00.22
Cards	5	00.17
Colts	3	00.11
Bucs	2	00.04

Figure 2

...Ever since the early days ... there has been a serious irritant regarding published programs. More often than not, several months after the program has appeared in the magazine, a letter to the editor or some one of the columns would contain a correction to the published program. When the magazines are filed away for later use, there is never a connection between the original program and the later, published corrections.

I note that you are getting into that situation in CodeWorks. Recent issues have contained some corrections to previously published programs. To assist in seeing that the corrections are known by your readers, why not print a page or pages of corrections in every issue? These should be cumulative and should cover corrections to all programs published in the previous year. Even though the same correction might be repeated for several months, keep it in there until the original program is at least a year old. How much better this would be than to have the frustration of entering a program and having problems with it, not knowing that a correction was buried somewhere in a letter to the editor in some other issue.

Please give this consideration.

Charles B. Steele
La Jolla, CA

We have run into that exact situation ourselves, not only with our own magazine, but with others as well. Trying to find that one-line fix can be frustrating, we agree. We hate to give up even one

page of our (already small) magazine to such continued corrections. But fear not, there is another solution. The program Qkey.bas, in this issue, will do the job. And why not let the computer do the work? Also in this issue you will find the cumulative index for all issues published through Issue 9. In each issue following this one, we will print only the new additions to the index for the previous issue. The complete index will be on the download for each issue and we will also include it with our yearly diskette.

...I've watched great ideas go down the tubes in exchange for glossy paper and fancy advertising for over-priced software. I am an advanced level programmer (everyone else is an expert these days.) Typing, debugging and eventually improving on my favorites has done much for my skill level. Your magazine is addressing all my needs, including Beginning BASIC; even us know-it-all types forget there is life outside of hard drives and sub-directories occasionally.

Gilbert J. Thompson
Marlborough, CT

...after several years "vacation" from BASIC (and all other programming) I find that you've rekindled the old flame that had dwindled away to naught. Oh yes, my computer has been busy enough (even though I'm the only one at the throttles), but its use was restricted to my trusty word processor program. At least that's the way it was, prior to receiving your magazines. Since then, however, I've keyed in most of your programs; had to debug a few coding errors (both yours and mine); and have learned more about BASIC in the past month than I had planned on...

...Obviously, you must recognize the familiar format of your Plist.bas program in the enclosed printouts, and I can assure you that I wouldn't leave home without that program. But, seeing what it does, I have to ask a dirty question. Why don't you use it on your printouts of your programs? As anyone can plainly see - my programs listings are very disciplined and don't infringe upon the sacred territory of the line numbers. Yours, however, are unruly and should receive a five yard penalty for encroachment...

...On a more serious note, I continue to be amazed by the reactions of so many readers to keying in programs. Most of them will do anything to avoid the "drudgery" of keyboarding. Hence, the popularity of programs on disks, downloads, and bar codes, among other things. Most of those folks don't realize the overall benefits to be gained from such "drudgery." But, in all fairness, they're really no different from me, or so many other computer addicts. In my case, I couldn't even touch-type until the computer came into my life...

Glenn W. McMillan
Fairfax, VA

Ever since Issue 6, when Plist.bas was first published, we have been using it to run off the programs in the issues. Because the pages in the magazine and the listing pages don't lay out well together, we removed the page heading and page eject features of Plist.bas. Then we run out the programs in one long strip and cut to fit the issue. So, since that issue, there has been no encroachment of the line numbers. We agree that typing programs in from listings is a very good way to learn. Knowing how to type is almost something that comes with the territory when you get a computer. It's hard to even imagine someone typing in programs with two fingers and trying to keep eyes on the keyboard.

Do diskettes wear out after using them daily over a period of time? Does this explain how sometimes a program won't work - or parts of it won't work?

Is it standard practice to renew an operating system disk from a master?

I notice some diskettes stored in covered containers seem to lose their programs after months of not using them. Is this normal?

J. L. Lopez
Pasadena, TX

Does the Spearmint lose it flavor...? Yes, it does. So can disks. Like most things, there is no absolute answer. We once used an operating system disk daily for over two years. You could almost see through the directory track when you held it up to the light, and it still worked fine. In general, there are three things diskettes do not like: Dirt, heat and magnetic fields. Of these, heat is probably the least offensive. Even a magnetized paper-clip laying on a diskette can re-arrange the data on the diskette. Storing a box of diskettes near an electric fan motor, for example, is asking for trouble. Dirt is a real killer, not only because of the abrasive qualities but because it can keep the head of the drive micro-inches away from the surface of the diskette. That small amount of space is usually sufficient to get an "iffy" read because the magnetized spots on the diskette are not that strong to begin with and the strength of the spot decreases as the square of the distance away from it. Get a disk drive cleaning kit and use it occasionally. Keep your diskettes in a cool, dry place and in their jackets, and make backup

diskettes of all your important media. It's easy to do and saves so much frustration. Just last week I tried to load BASIC from a system disk I've been using for over a year and it reported an error during read. I went back to the backup master and copied BASIC from it to the same diskette I'd been using and it works fine again.

I like your magazine and I'm learning each day. My copies get more dog-eared each week. I think I've learned enough to comment about a program. This is about Math.bas (Issue 4).

The permutation formula seemed to me to have some very high values for P. Your formula P=N!/(N-R)! is not complete. According to a reference I have the formula should read: P=N!/(R!*(N-R)!). For example, I listed the possible combinations of six items taken two at a time and came up with 15 different arrangements. Please check the attached modified program listing and let me know if I goofed...

Walt Handberry
El Paso, TX

No, you didn't goof. Your formula is correct for ***combinations*** *while ours is correct for* ***permutations****. There is a difference between permutations and combinations which is easy to overlook. Our math book says that combinations are groupings of things without regard to their order, while permutations are arrangements of things in the sequence of particular orders. The book goes on to illustrate the difference by saying that when the baseball coach picks 9 men out of a squad of 20 to form a team, he is making a combination of 20 "things" taken 9 at a time. Then when the coach assigns the members of such a team the sequence of a particular batting order, he is making a permutation of the same 20 "things" taken 9 at a time. The book we are referring to is "Mathematics made Simple" by A. Sperling, Ph.D. and Monroe Stuart, Published by Doubleday & Co, New York. They give the permutation formula as: P=n!/(n-t)! where n is the total number of things and t is the number of things being considered within the number n. It goes on to give the combination formula as: C=n!/t!(n-t)! where n and t again have the same meaning as in the permutation formula. Thanks for bringing this up. Until you called our attention to it we weren't that clear on it either.*

How about an article on how to fill out pre-printed forms with the computer...?

Jack Barron
Seattle, WA

Good question. First of all, if the form is not set to accommodate six or eight lines per inch vertically, you may as well forget it. Even assuming it is, backspacing on many line printers is a problem. Making a generalized program to fill out any form would be very difficult. If you have one or two forms that you use repeatedly, the best way to handle it is to write a program specifically for that form and then just feed it the variable information. Last summer I worked with a Realtor trying to do this for contract forms. We finally gave up and went back to the typewriter, mostly because of the line spacing on the forms, and the little letter x that we needed to cross things out. We just couldn't get anything to line up or stay lined up.

I hear the term "compiling a program" but don't really understand how to do it. Could you discuss this in one of your future articles? Or could you cite some publication that is available that would give one an understanding of it?

Robert L. Dingle
Dayton, OH

Microsoft has two or three versions of their compilers. One of them was BASCOM, another the QuickBasic Compiler (which we use here.) The manual on either will give some background on how to use it. Space permitting, there will be such an article in this issue, and if not in this one then the next issue will carry it.

...Regarding the RENUM program in Issue 9. I find I must add two lines to get it to work on my Tandy Model III (DOSPLUS) and Tandy IV (LDOS 6.3.0). It seems the TRS-80 brings the lines of the program in with a leading blank after the first line. This causes S to equal 1 and LN to equal zero. The program runs fine with the addition of lines 505 and 805 (which are identical): 505 IF LEFT$(A$,1)=" " THEN A$=MID$(A$,2, LEN(A$)+1) Added line 805 is the same as line 505. I'm sure there are other ways to correct this, but this works.

Bob Montgomery
Englewood, OH

I just checked this out on our Model III/IV and found the program worked as written using TRSDOS 1.3 on the III and LDOS 6.2 on the Model IV. I don't think that the BASIC in the DOS versions you have are that different. Then again, maybe they are. In any case, we have mentioned this so that anyone else with the same problem can have a solution, although we're not sure there is a problem.

...I enjoy your magazine and look forward to its arrival. How about a BASIC program that would exercise the printer operating system?

Gene G. Wyman
Ketchikan, AK

It's coming up soon. We got a Star Micronics SG-10 and found all sorts of goodies in it. But setting all those CHR$ codes got to be real tiresome. So we wrote the One $ Electric Typewriter. We called it that because it uses the dollar sign for a control code. We designed the program to be highly modifyable, so that it would work with almost any printer. Expect it in the May or July issue.

...I cannot make Card.Bas work, despite checking my entries again and again and trying variations. At the moment I keep getting the error message "subscript out of range in 360" Can you help? I have a Kaypro 2X (CP/M) with BASIC-80 Revision 5.21.

Michael Cass
Macon, GA

Since you have a BASIC past revision 5.2 you will probably need to remove or remark line 105. The program needs to have the file initialized prior to the first time you run it. We gave a way to do that in the article, but many had problems with it. Here is another way to do it from the BASIC ready prompt:

Type: OPEN"O",1,"CARD.DAT" (then press Enter)
Type: PRINT #1,"ZZZ" (then press Enter)
Type: CLOSE 1 (then press Enter)

This should create an empty file on diskette called CARD.DAT. Now when you run Card.Bas, it should find that file and let you start entering data. ***Don't*** *do the above if you already have a file called CARD.DAT or it will be wiped out! The errors you mention in your letter are all probably due to the file not being there in the first place.*

I have four drives on my Tandy Model IV. How do I enable them on TRSDOS 6.2?

Edward J. O'Hara
Boynton Beach, FL

Take the write-protect tab off of your DOS disk, boot up, and at the DOS ready prompt:

Type: SYSTEM (DRIVE=2,ENABLE) and then ENTER
then type: SYSTEM (DRIVE=3,ENABLE) and ENTER again.
Then do a SYSGEN to save the new setup on your system disk.

You should now be able to access all four drives. Check your DOS manual on the SYSGEN command if in doubt.

At $4 per copy, you have the most expensive mag in the U.S.A. You boys better get a second way to make a living. You are going to need it!

Bob Dowling
Tallahassee, FL

I originally thought that $24.95 was pretty expensive for a six-issue publication but then I realized that what I was actually receiving was free software, and we all know how much can be spent on that. Not only was I receiving free software, but also mini-lessons in BASIC programming. Software vendors are always so secretive about how their programs work and here was software that gave a line by line description of how all of the code worked. Amazing.

P. J. Beaulieu
APO, NY

At first glance SOX looks like a neat idea. But on second glance, it is far too limited and there is a better way to generate non-linear sequences. The major problem with SOX, in either its alphabetic or numeric versions, is its restriction on the size of the numbers generated. The string length grows out of sight if you want very large numbers. Here is a simpler way to generate a sequence such as 50, 275, 5, 314.

```
Q%=VAL(MID$("050035275005314",3*I%-2,3))
```

I prefer to define a special function to generate any sequence of 3-digit numbers I want:

```
10 DEF FNTD%=VAL(A$,3*J%-2,3)
```

No parameters are needed on the left side if one remembers to use the string A$ to define the sequence of numbers, and J% for the index number. If this doesn't work on all BASICs, then simply DEF FNTD%(A$,J%), where A$ and J% just become place holders for any string and any integer.

Here's a program to field a single buffer in two different ways at once. I assume that a random file has been opened with a record length of 45 bytes.

```
20 B1$="005010003027" 'Fielded as 5,10,3,27
30 B2$="027003005010" 'Fielded as 27,3,5,10
40 FOR I%=1 TO 2
50 IF I%=1 THEN A$=B1$ ELSE A$=B2$
55 T%=0
60 FOR J%=1 TO 4
70 FIELD 1,T% AS DU, FNTD% AS B$(I%,J%)
80 T%=T%+LEN(B$(I%,J%))
90 NEXT J%
100 NEXT I%
```

In generating the A$ string, it is important to pad the numeral on the left with zeros so that each numeral element is the same length - 3 digits in my example.

It may seem the hard way to field a buffer if one is only storing information in a few standard formats, but I store records in dozens of different formats in the same file so that they can be printed out as in a biographical directory. Consequently, one novel use I make of non-linear sequences is in properly fielding each record in accordance with a code embedded in the record itself. I also use non-linear sequences to generate the tab positions and printer codes to produce an "intelligent" formatter that knows when to repeat a heading, when to indent, etc. As you correctly pointed out having all this information coded in a single line makes for easy access.

...I want to see you continue your excellent pedagogical function, and would be happy to help whenever I can. I applaud your non-ad policy...

Jerry Bails
St. Clair Shores, MI

Corrections and Explanations

I am mortified to have made a mistake in my suggestion printed in the Forum of the Jan/Feb 87 issue re change to NFLSTAT.BAS. The change to line 890 should be:

890 INPUT"Enter correct team number-";Z$:IF Z$<>""THEN A(I,1)=VAL(Z$). I used = instead of <> in my letter. I apologize to all for this error without thinking. I'm sure others have found the right solution.

Larry Abbott
Wyomissing, PA

Don't feel bad, we missed it too.

Don Largent, of Shafter, California, called to point out that Rcard.Bas (Issue 8) bombed with no explanation on one job he was running. After a couple more phone calls he discovered that one of the addresses in his data file had the number sign (#) in it preceding an apartment number. It turns out that Rcard.Bas looks for that symbol to determine date, page, or field number to print. The most expedient way to avoid this problem is to not use the # sign in your data file. If that's a problem, you can change it to another, less used, character. If you do that, however, you will need to make the change in six places in Rcard.Bas: Lines 630, 640, 650, 890, 900 and 910. Then don't forget to use your new symbol when you make the report format file using Form.Bas.

While we are on the subject of Rcard, we haven't heard too much from you about it. Was it that easy to implement, or was it so intimidating that most of you didn't even try it? We would like to know, because there is no sense in spending that much time and space on something that very few of you use.

In Issue 9, in Beginning BASIC (of all places to goof!) the code in the second line from the top of the right hand column (page 15) should be: P=INSTR(I,A$,B$). Both I, and our usually sharp-eyed proofreaders, missed that one.

We got an inordinate number of phone calls during January asking where Issue 9 was. It had been put into the mail at the usual time (between Christmas and New Year), and the best we can figure, it got caught up in the glut of IRS tax booklets and the "You have already won 10 Million" sweepstakes mail.

On another note, a friend recently gave me a book entitled "Overcoming Indecisiveness", but I'm not too sure I want to read it. Oh, well.

Thanks for all your input. We appreciate it and it points the way. Tell your friends you saw it in CodeWorks. - Irv

"I'm sorry Captain, but the battle computer has joined the peace movement."

Random Files

How stacks work & stacks in Randemo4.Bas

Terry R. Dettmann, Associate Editor. In this installment, Terry incorporates the idea of stacks into the Random file program. Included is a separate program that illustrates graphically how a stack works. The addition of stacks updates Randemo3 to Randemo4. Both the merge lines and the complete, merged Randemo4.Bas programs are included here.

Discussion

Last issue we introduced the simple random data file editor which had limited capabilities to add, edit, and delete records from a random file. On its own, it seems like a pretty sorry little program. It isn't very sophisticated, has no real reporting capability, does nothing particularly useful, and really doesn't get us much into random files at all. But over the next several issues, just watch where that program will take us.

In this issue, we're going to start expanding the simple random data file editor (we'll call it RANDEMO for short). Each issue we'll be introducing some new feature all built on the same basis. Some will be optional and some will be essential. This particular one will be essential.

The problem facing us here is one we pointed out in the last article when we were talking about deleting records. At that time, we noted that simply deleting a record leaves a hole in the data base. We put the word DELETED there to mark the record, but we had no way of getting the record back when we added a new one. Each time we added a record, the routine that determined where to put a record (subroutine 1300) always put them at the end of the file.

This might be OK for someone who never deletes anything, but most people want to delete things, even if it's only people they never want to hear from again. But how can we remember where the holes are?

Stacks as a way to keep track

One way to remember in a program is a device called a "stack". Stacks are a type of "data structure," a way of storing data. An array is also a data structure, one in which we find a piece of information by knowing the address in the array where the data is located.

In a stack, we have a special requirement. We want the first thing we find to always be the last thing we put into it. Just like a stack of trays in a cafeteria. The last tray we add to the stack is the first one that will be pulled out.

This type of data structure is called a "LIFO" or "Last In First Out" structure.

How do we make a Stack?

We have a number of ways that we can make a stack in a program. Since there is nothing that we can simply use in BASIC, we have to use the tools at hand to make them.

The standard way to make a stack is to set aside a stack array (let's call it "STK") which will hold the items to be stacked. An integer called the stack pointer (we'll call it "SP") will point to the array element which has the last thing added to the stack. Initially, we'll start SP at a value of '-1' which will mean that the stack is empty, nothing has been added.

Whenever we add something to the stack, we will set SP to the array location of the item (we have to be careful to not go beyond the size of the array so some protection is necessary). When we want to get something from the stack, we get it by taking whatever is at the location SP addresses, STK(SP). The sample program included here shows how this works in a visual manner.

The stack demo program

The stack program illustrates the principle of stack operation by creating a simple command processor for the two basic stack operations, PUSH and POP. When started, the stack program will display a 10 item stack on the screen and accept the commands 'PUSH' and 'POP' to illustrate how they work. For example, the command 'PUSH 10' pushes the number '10' onto the stack. An arrow (the stack pointer) will move to point at the number '10'. The command 'POP' will remove the top item from the stack and push the arrow down one.

The program initializes itself in lines 30-190. Variable "SZE" is the maximum size of the stack, "SP" is the stack pointer. Array "STK" is the actual stack while array "WD" is an array which will be used to break up the words in the command line.

In line 190, we call subroutine 1400 to clear and setup the screen and subroutine 1500 to display the stack itself.

Lines 200-240 are the command processor. A command line in the form of one or more words is input in line 210 into CD$. Subroutine 1700 converts all the characters in the line to upper case for processing and 1000 determines whether the first word is one of the allowed commands:

PUSH - add a number to the stack (next word on line)
POP - take a number off the stack
CLEAR - clear the stack
END - end the program

Every time a command is processed, the error location is cleared by subroutine 2050 so we know if we mess up when the line is processed.

When line 230 is executed, if subroutine 1000 didn't recognize the first word, then subroutine 2000 will be executed. Otherwise, the subroutines correspond to allowed commands as follows:

PUSH at 2100
POP at 2200
CLEAR at 2300
END at 2400

After executing whichever command is appropriate, the program clears the command area (subroutine 1900) and then returns to line 200 to start the command loop all over again. Let's look at the commands one at a time.

Error Handler (subroutine 2000)

If there is an error, this subroutine goes to a defined place on the screen and prints "OOPS" to indicate it didn't recognize the command word. We could have had it do something more sober and practical, but just between us, OOPS said it all.

PUSH (subroutine 2100)

To push something on the stack, we first make sure that the stack pointer (SP) hasn't gotten too large. If it has, we ignore the request. Otherwise, we increase the stack pointer by one and then put the number at that location on the stack. Notice that we clear the stack pointer "=>" on the screen so when we rewrite the stack, only one stack pointer will appear.

POP (subroutine 2200)

To pop something off the stack, we set the top back to zero and decrease the stack pointer by one. If the stack pointer is less than zero, then there's nothing on the stack and nothing to do, so we return.

CLEAR (subroutine 2300)

To clear the stack, we reset the original condition, all locations zero and the stack pointer at - 1.

END (subroutine 2400)

The end command simply clears the screen and says good bye before ending (thought it might be nice to be polite for once).

Let's play with stacks

Let's imagine that the stack display on the screen is the stack which remembers where deleted records are. When we start the program (create the data file) there are no deleted records, so we have nothing. Each time we delete a record, we "PUSH" its record number on the stack.

Let's assume we've created 47 records. If we delete record number 21, then there are 46 records active and a hole at location 21. When we delete the record, we "PUSH 21" to add the record number to the stack.

Now, let's assume that we delete record 43, so we "PUSH 43" to add that record number to the stack. Next delete record 10, so "PUSH 10." The stack should now look like figure 1 with 3 records (21, 43, and 10) on the stack from bottom to top.

When we want to add a new record to the file, we ask for a hole to put it in by getting the item at the top of the stack (10) and then using POP to remove it. Now the stack looks like figure 2, with only 21 and 43 on it from the bottom up.

Play with the stack program until you're sure you understand how the stack structure works. Then move on to add stacks into the Randemo program.

```
HOW THE STACK WORKS IN THE RANDOM FILE DEMO
Commands: Push ## or POP or Clear or END
          10:    0
           9:    0
           8:    0
           7:    0
           6:    0
           5:    0
           4:    0
           3:    0
       => 2:   10
           1:   43
           0:   21

         Command>
```

Figure 1
See text. The newest number "Pushed" onto the stack always goes in on top of what is already there.

```
HOW THE STACK WORKS IN THE RANDOM FILE DEMO
Commands: Push ## or POP or Clear or END
          10:    0
           9:    0
           8:    0
           7:    0
           6:    0
           5:    0
           4:    0
           3:    0
           2:    0
       => 1:   43
           0:   21

         Command>
```

Figure 2
Issuing the command to "POP" will always take from the stack the very last item that had been added to it. Notice that PUSH always needs to have a number following it, while POP, CLEAR and END do not.

A modification to the Random Files Editor

I have included a merge file as part of this article which adds a stack implemented as a file into the Randemo program. The way this stack works is a little different from the array in the stack program. Here, we use another random file to hold the stack entries. We can look at this file as an array holding integer numbers.

The first element of the array (record number 1) is the stack pointer in this case. Initially it is set to "1." Records 2 through the length of the file are the entries. As we delete records, we add the entries into the stack file (and update the stack pointer in record "1"). As we reuse deleted records, we find out where the top of the stack is at from record 1 (if the stack pointer is "1", then we know there is nothing on the stack).

Once we have the stack pointer from record one, we use that to locate the deleted record number and then decrease the stack pointer by one in record number 1.

In the merge file, line 125 adds the name of the stack file (extension .STK), line 130 opens it with a record length of 4 bytes and fields it, and line 135 initializes the stack if it doesn't already exist.

When we need to add a record, we go to subroutine 1300, so we've added line 1305 to make a check for an available deleted record before adding the record at the end of the file. Subroutine 1550 checks the deleted record stack for an available record. It returns record number (RN) of zero if there is no deleted record to fill.

When we're deleting a record, we go to subroutine 3100 to delete it. So while we're there, we have line 3120 call subroutine 1500 and save the record number on the deleted record stack.

With these lines merged in, you now have a slightly more useful program in the random file handler. But there's a lot more to come. We still have to have ways of sorting and searching for records in very large files, improving our storage so we have more useful data fields, printing reports, and much more. Among other things, we'll spend some time providing a different way of giving commands to the program (other than a numbered menu.) ■

Stack Operation Demo. This program does not in itself become a part of the Random file series. It is, instead, a demonstration to show how the stack operations work within Randemo4.Bas.

```
10 REM --- Stack Operation Demo
20 REM --- by Terry R. Dettmann for Codeworks Magazine
30 SZE=10:SP=-1
40 DIM STK(SZE),WD$(10)
190 GOSUB 1400:GOSUB 1500
200 REM --- Command Loop
205 GOSUB 1500
210 X=10:Y=15:GOSUB 1600:LINE INPUT"Command>";CD$
220 GOSUB 1700:GOSUB 1000:GOSUB 2050
230 ON CD GOSUB 2000,2100,2200,2300,2400
240 GOSUB 1900:GOTO 200
1000 REM --- parse command line
1005 GOSUB 1100
1010 IF WD$(0)="PUSH" THEN N=VAL(WD$(1)):CD=2:RETURN
1020 IF WD$(0)="POP" THEN CD=3:RETURN
1030 IF WD$(0)="CLEAR" THEN CD=4:RETURN
1040 IF WD$(0)="END" THEN CD=5:RETURN
1050 CD=1:RETURN
1100 REM --- break line into words
1110 I=0
1120 IF I>10 OR LEN(CD$)=0 THEN RETURN
1125    IF MID$(CD$,1,1)=" " THEN CD$=MID$(CD$,2):GOTO 1120
1130    L=INSTR(CD$," "):IF L=0 THEN WD$(I)=CD$:RETURN
1140       WD$(I)=MID$(CD$,1,L-1):CD$=MID$(CD$,L+1):I=I+1
1150 GOTO 1120
1400 REM --- Display Screen Lead
1410 CLS
1420 PRINT"HOW THE STACK WORKS IN THE RANDOM FILE DEMO"
1430 PRINT"Commands: Push ## or POP or Clear or END"
1440 RETURN
1500 REM --- display screen
1510 FOR I=0 TO SZE
1520    X=10:Y=13-I:GOSUB 1600
1530    PRINT USING"##: ####";I;STK(I);
1540 NEXT I
1550 GOSUB 1800
1560 RETURN
1600 REM --- goto screen position
1610 LOCATE Y,X:RETURN
1700 REM --- convert to upper case
1710 FOR I=1 TO LEN(CD$)
1720 C$=MID$(CD$,I,1)
1730 IF C$>="a" AND C$<="z" THEN MID$(CD$,I,1)=CHR$(ASC(C$)-32)
1740 NEXT I
1750 RETURN
1800 REM --- place stack pointer
1810 X=8:Y=13-SP:GOSUB 1600:PRINT"=>";:RETURN
1850 REM --- clear stack pointer
1860 X=8:Y=13-SP:GOSUB 1600:PRINT"  ";:RETURN
1900 REM --- clear command line
1910 X=18:Y=15:GOSUB 1600:PRINT"                    ":RETURN
```

Change line 1610 for your machine as follows:

Tandy I/III: 1610 PRINT@((X-1)*64)+(Y-1),;:RETURN

Tandy IV: 1610 PRINT@((X-1),(Y-1)),;:RETURN

CP/M BASICs: 1610 PRINT CHR$(27)+"Y"+ CHR$(31+X)+CHR$(31+Y);:RETURN

```
2000 REM --- error handling
2010 X=40:Y=15-SZE:GOSUB 1600:PRINT"OOPS":RETURN
2050 REM --- clear error line
2060 X=40:Y=15-SZE:GOSUB 1600:PRINT"    ":RETURN
2100 REM --- push instruction
2110 IF SP>= SZE THEN RETURN
2120 GOSUB 1850:SP=SP+1:STK(SP)=N:RETURN
2200 REM --- pop instruction
2210 IF SP<0 THEN RETURN
2220 GOSUB 1850:STK(SP)=0:SP=SP-1:RETURN
2300 REM --- clear stack
2310 GOSUB 1850:SP=-1:FOR I=0 TO SZE:STK(I)=0:NEXT I:RETURN
2400 REM --- end of program
2410 CLS:PRINT"Thank you for coming":END
```

Randemo4.Mrg These are the lines added to Randemo3.Bas (Issue 9) to make it Randemo4.Bas.

```
125 FD$=FF$+".dat":FS$=FF$+".stk"
130 OPEN "R",1,FD$:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
135 IF LOF(2)=0 THEN LSET SK$=MKI$(1):PUT 2,1
1305 GOSUB 1550:IF RN<>0 THEN RETURN
1500 REM --- Stack current record number
1510 GET 2,1:SP=CVI(SK$):SP=SP+1
1520 LSET SK$=MKI$(RN):PUT 2,SP
1530 LSET SK$=MKI$(SP):PUT 2,1
1540 RETURN
1550 REM --- Get top of stack or 0
1560 GET 2,1:IF CVI(SK$)<=1 THEN RN=0:RETURN
1570 SP=CVI(SK$):GET 2,SP:RN=CVI(SK$)
1580 SP=SP-1:LSET SK$=MKI$(SP):PUT 2,1
1590 RETURN
3120 GOSUB 1500:GOSUB 1200
```

Randemo4.Bas - the merged version upon which we will build in the next issue.

```
10 REM --- RANDEMO4.Bas - Random Files with stack
20 REM --- Terry R. Dettmann for Codeworks Magazine
30 DIM FP$(5)
40 DEF FNCTR$(X$)=STRING$((WD-LEN(X$))/2," ")+X$
41 DEF FNLF(X) = LOF(X)/128
50 WD=80:LN=24
60 FALSE=0:TRUE = NOT FALSE
100 REM --- file setup
110 CLS:PRINT FNCTR$("RANDOM FILE DEMO"):PRINT:PRINT
120 LINE INPUT"FILENAME: ";FF$
125 FD$=FF$+".dat":FS$=FF$+".stk"
130 OPEN "R",1,FD$:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
135 IF LOF(2)=0 THEN LSET SK$=MKI$(1):PUT 2,1
140 FOR I=1 TO 5
150    FIELD 1, (I-1)*25 AS ZZ$, 25 AS FP$(I)
160 NEXT I
200 REM --- main menu
210 CLS:PRINT FNCTR$("RANDOM FILE DEMO"):PRINT:PRINT
220 PRINT TAB(15)"1. Add Records to the File"
230 PRINT TAB(15)"2. Edit Records from the File"
240 PRINT TAB(15)"3. Delete Records from the File"
```

```
250 PRINT TAB(15)"4. Print the File"
260 PRINT
270 PRINT TAB(15)"0. End the Program"
280 PRINT:PRINT
290 PRINT TAB(15)"Command: ";
300 LINE INPUT CD$
310 CD = VAL(CD$)
320 IF CD<0 OR CD>4 THEN PRINT"Bad Command":GOTO 290
330 IF CD=0 THEN 500
340 ON CD GOSUB 1000,2000,3000,4000
350 GOTO 200
500 REM --- End of Program
510 CLS:PRINT"All Done":CLOSE:END
1000 REM --- Add Records
1010 CLS:PRINT FNCTR$("ADD RECORDS"):PRINT:PRINT
1020 FOR I=1 TO 5
1030    PRINT TAB(15)"FIELD(";I;"): ";:LINE INPUT LN$
1040    IF LEN(LN$)>25 THEN PRINT"TOO LONG, TRY AGAIN":GOTO 1030
1050    LSET FP$(I)=LN$
1060 NEXT I
1065 RN=0:GOSUB 1200
1070 LINE INPUT"Add More (y/n)? ";YY$
1080 C$=MID$(YY$,1,1)
1090 IF C$="Y" OR C$="y" THEN 1010
1100 IF C$="N" OR C$="n" THEN RETURN
1110 GOTO 1070
1200 REM --- save current record to data base
1210 IF RN=0 THEN GOSUB 1300
1220 PUT 1,RN
1230 RETURN
1300 REM --- assign a record
1305 GOSUB 1550:IF RN<>0 THEN RETURN
1310 RN = FNLF(1) + 1:RETURN
1400 REM --- get record from data base
1410 IF RN<1 OR RN>FNLF(1) THEN RETURN
1420 GET 1,RN
1430 RETURN
1500 REM --- Stack current record number
1510 GET 2,1:SP=CVI(SK$):SP=SP+1
1520 LSET SK$=MKI$(RN):PUT 2,SP
1530 LSET SK$=MKI$(SP):PUT 2,1
1540 RETURN
1550 REM --- Get top of stack or 0
1560 GET 2,1:IF CVI(SK$)<=1 THEN RN=0:RETURN
1570 SP=CVI(SK$):GET 2,SP:RN=CVI(SK$)
1580 SP=SP-1:LSET SK$=MKI$(SP):PUT 2,1
1590 RETURN
2000 REM --- Edit Records
2010 CLS:PRINT FNCTR$("EDIT RECORDS"):PRINT:PRINT
2020 LINE INPUT"SEARCH FOR: ";SF$
2030 GOSUB 2100
2040 IF FOUND THEN GOSUB 2200 ELSE PRINT"NOT FOUND"
2050 LINE INPUT"Edit More (y/n)? ";YY$
2060 C$=MID$(YY$,1,1)
2070 IF C$="Y" OR C$="y" THEN 2010
```

```
2080 IF C$="N" OR C$="n" THEN RETURN
2090 GOTO 2050
2100 REM --- search for record
2110 FOUND=FALSE
2120 FOR RN=1 TO FNLF(1):GOSUB 1400
2130    FOR I=1 TO 5:IF INSTR(FP$(I),SF$)<>0 THEN FOUND=TRUE:RETURN
2140    NEXT I
2150 NEXT RN
2160 RETURN
2200 REM --- edit record
2210 CLS:PRINT FNCTR$("EDIT RECORDS"):PRINT:PRINT
2220 FOR I=1 TO 5
2230    PRINT TAB(15)"FIELD (";I;"): ";FP$(I)
2240 NEXT I
2250 PRINT:PRINT
2260 LINE INPUT"FIELD TO CHANGE OR '0' TO END: ";FC$
2270 C$=MID$(FC$,1,1):IF C$="0" THEN GOSUB 1200:RETURN
2280 C=VAL(C$)
2290 LINE INPUT "CHANGE: ";LN$
2300 LSET FP$(C) = LN$
2310 GOTO 2200
3000 REM --- Delete Records
3010 CLS:PRINT FNCTR$("DELETE RECORDS"):PRINT:PRINT
3020 LINE INPUT"SEARCH FOR: ";SF$
3030 GOSUB 2100
3040 IF FOUND THEN GOSUB 3200 ELSE PRINT"NOT FOUND"
3050 LINE INPUT"Delete More (y/n)? ";YY$
3060 C$=MID$(YY$,1,1)
3070 IF C$="Y" OR C$="y" THEN 3010
3080 IF C$="N" OR C$="n" THEN RETURN
3090 GOTO 3050
3100 REM --- delete the current record
3110 LSET FP$(1)="DELETED"
3120 GOSUB 1500:GOSUB 1200
3130 RETURN
3200 REM --- delete record
3210 CLS:PRINT FNCTR$("DELETE RECORDS"):PRINT:PRINT
3220 FOR I=1 TO 5
3230    PRINT TAB(15)"FIELD (";I;"): ";FP$(I)
3240 NEXT I
3250 PRINT:PRINT
3260 LINE INPUT"Are you sure (y/n)? ";YY$
3270 C$=MID$(YY$,1,1)
3280 IF C$="Y" OR C$="y" THEN GOSUB 3100:RETURN
3290 IF C$="N" OR C$="n" THEN RETURN
3300 GOTO 3050
4000 REM --- Print File
4010 CLS:PRINT FNCTR$("PRINT FILE"):PRINT:PRINT
4020 FOR RN=1 TO FNLF(1):GOSUB 1400
4030    IF INSTR(FP$(1),"DELETED")<>0 THEN 4070
4040       FOR I=1 TO 5:PRINT "FIELD(";I;"): ";FP$(I)
4050       NEXT I
4060       PRINT
4070 NEXT RN
4080 RETURN
```

Beginning Basic

Graphing in three easy pieces

Even if you do not have high resolution graphics in your computer you can still make some respectable looking graphs, both on your screen and your line printer. You may be surprised to find how little code it actually takes to do it, and how easily that code can be changed to do different things.

Figure 1 shows a small program that will create a bar graph of ten data points. Only six BASIC statements are used (not counting the remark statement) to make this little program function. The values we are going to plot are in DATA statements in lines 30 and 40. These could represent almost anything: sales per unit time, number of items produced per month, etc. Data statements can be placed anywhere in a program; at the beginning, end or in the middle. In fact, they do not even need to be all in one place. In our example, for example, we could move line 40 to line 100 and it would still work. The data statements in a program are simply passed over as BASIC executes the lines of code. That is, until a READ statement is encountered. When BASIC comes to a READ statement, it goes and finds the first DATA statement in the program and starts to read there. In our example, in line 70, we READ D. We could have called the data anything we liked at this point, which is to say we could assign any variable not already used elsewhere in the program to contain the data we read.

Notice that we have ten data items, and that our loop which will read them is set to read all ten of them. We could read less than ten if we wanted to. But if we try to read eleven or more items an "out of data in line 70" error will occur. Try it. Change the 10 in line 60 to 11. Notice that the error occurs in the line trying to *read* the data, and not in the data line itself.

Now let's run through the program and see what happens. The loop between lines 60 and 80 will read the data items, one at a time. The "READ" takes place in line 70, and after we have read the first item we use STRING$ to print a line of asterisks that is as many character spaces long as the data items indicate. The first time through the loop there will be 15 asterisks printed, the next time through the loop it will print 20, and so on. STRING$ is an interesting function in BASIC. PRINT STRING$(20,"*"), for example, will print a row of 20 asterisks. You can put any valid ASCII character in place of the asterisk. STRING$(20," ") will give you 20 spaces. Nor does the symbol (ASCII character) to be printed need to be a literal, as we have shown with the asterisk. You can eliminate the quotes around the asterisk and put the ASCII value for it there instead, so that STRING$(20,"*") and STRING$(20,42) give the very same results. The number 42 is the ASCII number for the asterisk.

If we don't want a bar going across the screen (or printer) we can plot only the point representing the value by itself. To do this, change line 70 to: READ D:PRINT TAB(D);"*" and we will have something like Figure 2. It sort of leaves a bunch of asterisks hanging out there all by themselves, but you could manually connect them to show the plot line. This works because we can use the TAB function to move the cursor (or print head on the printer) to column position D, and once there print the asterisk.

Now we still have data points hanging out there all by themselves. How about putting some reference lines or points around the graph to dress it up? If we change line 70 to: READ D:PRINT I;TAB(D);"*" it will print the value of the loop counter (from 1 to 10) to the left of each plotted point. That will give it some reference. Or, you could change line 70 to: READ D:PRINT D;TAB(D);"*" and it will print the actual value of D to the left on each line. We have done this in Figure 3, and added a ruler line in line 90 as well. The ruler line is just a line of periods with an exclamation point thrown in every ten spaces for easy reference. To make this all print on the printer instead of the screen, just change the PRINT commands in lines 70 and 90 to LPRINT, turn on your printer and RUN. It's that easy.

Now let's get real for a while. You have probably guessed by now that the data values we chose fit very conveniently on the screen or printer. In the real world it doesn't happen that way, so what do we do then? In that case, we will need to do some scaling. Let's say your largest data value is 238. That would drive your TAB crazy and make it wrap around on the screen and the graph would look like a mess. If 238 was your largest value and your screen had 80 character positions, you could easily divide all values by three and keep them all on screen. In that case, line 70 would need to be changed to: READ D:PRINT D;TAB(D/3);"*" and you would have the actual value to the left but the plot would be scaled to fit the screen (or printer.) And yes, you can do arithmetic in a TAB as we just did. But when we start scaling, we lose resolution (you cannot tab to a fractional number, tabs are whole integers.) But, since all your values are scaled the same way, you still get a fair representation of the trend of the data.

How about putting more than one curve per graph, can that be done? Yes, with a lot of extra coding. You would have to check each point to see which value comes first (because you cannot tab backwards) and then print the first point (let's say it uses the plus sign) and then continue down the line and tab to the second (say it's the asterisk.) In this case, the data in the data statements would need to be interlaced too, and we would need to do two "reads" inside the loop. All in all, it would be sort of messy.

How about the fact that our graph is 90 degrees clockwise from the way we normally read graphs? Can we fix that? Yes we can. And we can find an easier way to get two or more curves on the same graph too, and put in some grid lines that don't interfere with the data as well. We are going to do it in the next issue, but in the meantime, here is a hint: We will "build" the entire graph in memory and then when it's all done we will print the whole thing out. Why? Because in memory we can go over the graph as many times as we want and, in effect, tab backwards and scroll up the page. By contrast, the graphs we show in this article are all printed once, from top to bottom and left to right, with no chance of doing much more than that.

Play with the programs in this article for a while. Try different data and different symbols. The lowercase letter "o" makes a nice plotting symbol. Add some numbers below every ten marks of the ruler line.

It's amazing what can be done with just six BASIC statements. ■

Figure 1

```
10 REM * BBasic1.Bas * Figure 1 for Beginning Basic Issue 10
20 '
30 DATA 15,20,34,38,45
40 DATA 40,36,40,46,54
50 '
60 FOR I=1 TO 10
70   READ D:PRINT STRING$(D,"*")
80 NEXT I
Ok
run
***************
********************
**********************************
**************************************
*********************************************
****************************************
**********************************
****************************************
*********************************************
**************************************************
Ok
```

Figure 2

```
10 REM * BBasic2.Bas * Figure 2 for Beginning Basic Issue 10
20 '
30 DATA 15,20,34,38,45
40 DATA 40,36,40,46,54
50 '
60 FOR I=1 TO 10
70   READ D:PRINT TAB(D);"*"
80 NEXT I
Ok
run
               *
                    *
                              *
                                   *
                                        *
                                             *
                                        *
                                   *
                                             *
                                                  *

Ok
```

Figure 3

```
10 REM * BBasic3.Bas * Figure 3 for Beginning Basic Issue 10
20 '
30 DATA 15,20,34,38,45
40 DATA 40,36,40,46,54
50 '
60 FOR I=1 TO 10
70   READ D:PRINT D;:PRINT TAB(D);"*"
80 NEXT I
90 PRINT".........!.........!.........!.........!.........!.........!
Ok
run
 15             *
 20                  *
 34                       *
 38                            *
 45                                 *
 40                            *
 36                       *
 40                            *
 46                                 *
 54                                      *
.........!.........!.........!.........!.........!.........!
Ok
```

Qkey.Bas

A versatile, quick, keyword search program

Staff Project. This was another of those programs that almost wrote itself. When we finally got done playing with it, we found it so useful that we are now keeping a half-dozen different files with it. Our cumulative index, included here, is just one example of its versatility.

Do you remember back to before you had a personal computer? Before the days of "User Friendliness?" We all had the dream of simply throwing all sorts of information into such a computer and then letting it find anything that we asked of it.

As we all have come to find out, it's not that easy. There are fields, sizes, alpha and numeric designations and menus, menus and more menus, from which we pick one from column A and another from sub-menu C. In fact, some of the current applications programs have as many as 150 different commands!

Qkey.bas is an attempt to get back to simplicity with utility. It cannot compete with today's databases. Yet, it has several things going for it that make it a useful, and simple to use, tool.

The program allows you to enter almost any kind of data into it, in any order, and then retrieve selected items by using keywords. Your information may be as free-form as you wish it to be. Any entry may be up to two screen lines long. You can scan the entire file or look for specific items. You can also use the "and" and "or" conjunctions in your search. Once found, you can delete the entry, replace it with something else, or skip on to the next record. The program will work with as many files as you wish to create, and if the file you specify does not exist, it will automatically create it for you.

One of the problems with menu-less programs, such as this, is that one must be able to differentiate between data input and command functions. The options are to get into esoteric CRTL-XX type functions for commands, which are difficult to remember, or to use plain English words, which may be misinterpreted as data instead of commands. Our solution to this problem was to use "dot" commands, plain English preceded by a period. This leaves the default mode, any input *not* preceded by a period, as valid data input.

Qkey.bas features a complete, on-line, help function. Issuing the command .help will bring up the list of commands available, and .help find, for example, will bring up information on the ".find" command. No matter what function you perform with this program, you are always returned to a unique prompt symbol (the double greater than, >>) from which you can perform any other function. Those of you who are familiar with MSDOS or UNIX will recognize the >> symbol, since it is used in those systems to append to an existing file. We have borrowed the concept here too, but use it a little differently.

Let's run through the operation of the program first, and then examine the program code later. When you run the program you are presented with some information about it. The first item of information says that for hardcopy output to use your screen print function. Almost all of the machines we cover have some facility to send the information on the screen to the printer. Those machines using NEWDOS80 can use the JKL keys for this purpose. Others use Shift and the S and P keys at the same time, while yet others (MSDOS) have a screen print key. The next bit of information is that if the file you specify does not exist, it will be created for you. Next there is a reminder that >> is the prompt for this program. This is followed by a note that data entry is the default mode for this program and that commands must be preceded by a period.

You are then asked for the name of the file to use.

If you press ENTER at this point the program will create a file called KEY.DAT. Otherwise, you may specify any file name, with or without an extension. The program does not append any type of extension for you. You are then presented with the >> prompt.

From this point on, anything you type (not preceded with a period) is taken as data input. The Enter key will signify the end of a record of data. Although single screen line records will work best, an entry may be up to two screen lines long. This limitation is imposed because of the maximum number of characters BASIC allows in one string. After you have entered any information you are always presented with the >> prompt. From this prompt, you can enter more records or any of several commands. Here is a summary of the commands available:

.find - searches through the file looking for a match to your input. If you use .find with no argument, the entire file will be displayed in the order in which you entered it. If you use .find horse, then only "horse" will be searched in the file and the record or records containing "horse" will be displayed. When records containing your keyword ("horse" in this case) are found they will be displayed on the screen with an asterisk before it. This was helpful to identify the beginning and end of records when they cover more than one screen line. After all records containing your keyword have been found, the screen will display information telling you how many there were. You may also use "and" and "or" in your search. To do this, you may use, for example, .find horse (and) cow, and only records containing both those words will be displayed. If you use .find horse (or) cow, then any record containing *either* word will be displayed. Notice that the words "and" and "or" must be put inside parentheses without spaces, otherwise, the program would look for the actual word "and" or "or." The spaces you use around keywords are important, as is capitalization. In the example, .find horse (or) cow, the program will not find "horsedrawn" because of the space following the word "horse." If you want to find keywords within other words, use (in this example) .find horse(or)cow. Now it will find "racehorse" or "cowgirl." The "or" connective is useful to find words that may or may not be capitalized, as in .find North (or) north. Another way to avoid the capitalization problem is to .find orth, i.e., omit the first letter of the word, which may or may not be capitalized. The program will not allow you to use both "and" and "or" in the same command.

.edit - used without an argument will display the entire file just as .find without an argument does. Everything we just said about .find, above, also applies to the .edit command, including the "and" and "or" conjunctions. This command allows you to delete a record, or delete it and replace it with a different record. It also allows for a skip option if you have not found the record you thought you wanted. You may .edit or .find entire phrases, as in '.edit in the summer (or) next Tuesday afternoon.' The more detailed your search keys are the more discriminating the search will be. In both the .find and .edit commands, records containing the search keywords will be displayed on the screen. If there are enough entries to more than fill the screen the program will stop when the screen is near full and display a prompt to Enter for more or use .quit to abort the search.

.save - will, at any time, save the file to disk and allow you to continue using it. During the save process, it will eliminate deleted records.

.quit - can be used to abort a search, to quit using a file and to quit the program altogether. When you use .quit at the >> prompt, the file will be saved to disk automatically and all deleted records will be removed and you will be at the program start again so that you can call for a different file. If you then enter .quit for the file name, the program will terminate and return you to the normal BASIC ready prompt.

.file - this command, like .save and .quit, requires no argument. This one simply tells you how many records are in the file and the name of the file you are currently using. It then returns you to the >> prompt. If you have deleted records and have not used .save, it will tell you how many records you have *including* the deleted records.

.help - can be used any time you are at the >> prompt. Typing .help will show you the commands that help is available for. Typing .help .edit will get you information on the .edit command, etc. Note that all commands may be used in either all upper case or all lower case, but not mixed case.

Program Details

The program proper starts at line 150, where machines using BASIC prior to version 5.1 should remove the remark at the start of the line to clear some string space. If you must clear string space, clear almost as much as you have memory left when the program is loaded. See the next paragraph, where we discuss the variable NR, the maximum number of records.

Line 160 sets the maximum number of records.

This is an arbitrary number, and what you use here will depend on how much memory you have left over when the program is loaded. To find out, load (don't run!) the program into memory, and at the BASIC ready prompt, enter: PRINT FRE(A$). This will tell you how much free memory remains for the program to use. On some machines you will need to enter PRINT MEM instead of FRE(A$). In either case, the machine will respond with a number between 16,000 and 50,000 depending on your machine and memory size. If you had to use the CLEAR in line 150, then subtract about 200 from this number and use the difference in the CLEAR statement. To get the variable NR for line 160, you must decide on what the average length of your records will be. Then divide the free memory space number you have by this average and set NR in line 160 to that number. If the average length of the records you enter are what you said then this number will be used to tell you how many more records you can enter. It's not absolute however, just a good guess.

In line 170 we set the number of columns on your screen and the number of lines in SW and SL, respectively. Some machines will need these to be changed to 64 and 16. The reason for setting these parameters is that the program needs to know where to stop scrolling when the information you display is more than one screenfull.

Every record you enter into this program is held in the B$ array. In line 180 we dimension that array to the maximum number of records that was set in line 160.

The following lines in the program print the usual CodeWorks heading and some information (mentioned earlier, at the beginning of this article) about the prompt, hardcopy output and data entry mode. Line 340 asks you for the name of the file to use. If you simply press Enter here, the program will use the default file KEY.DAT, and if it does not exist, it will create it. Note that using .quit or .QUIT here will cause the program to end. Note also that you can change the default file name to anything you like; it only appears in lines 340 and 360. Naturally, if you use any other than the default file name, the program will use that file instead, and again, if it does not exist it will be created. The ON ERROR GOTO 840 is placed at line 370 simply because it was a convenient place to put it. It needs to be anywhere near the beginning of a program where it can be read at least once before it can be acted on. We will get back to this statement in a short while.

The section of code from 390 to 450 is where we open the file on disk and read in the information already there. Note that we are not using any sentinel character for the file. The end of file (EOF) marker is sufficient to tell us where the end of the file is. It is checked for inside the M loop just prior to where we input each line from the file. When we reach EOF in the file, control jumps to line 450 where we set variable N to be the value of M less one. This is because the M count is already advanced one past the last item in the file. From here on in, N will tell us how many records are in the file, and will be updated each time we add another record.

If the file we requested does not exist, then when we try to open the file and read it in line 400 we will get an error. Because of the ON ERROR GOTO statement in line 370, however, our program is not going to bomb out on us; program flow will go to line 840 instead. Let's look at that section of code now to see what happens there.

The code from line 830 to 930 is where we create the file if it does not exist. The first thing we do when we get here is to check and see if the error code that brought us here is a "file not found" error. That's the one we want to act on. But, the error could just as well have been some other error, like a syntax error, and we still want our program to recognize such errors. The way we are using line 840 says that if the error was not 53 (file not found), then ON ERROR GOTO 0. In our case, ON ERROR GOTO 0 was executed inside the error trapping routine, and BASIC will now stop execution and print the error message for the error that caused the error trap to be executed. It doesn't make any difference that there is no line 0. This way, we can pick the "file not found" error out of all the other types of errors and act on it without impairing the function of the error handling inherent in BASIC.

If the error is 53 (file not found) then line 850 will tell us there is no file named whatever we named it and asks if we want one. If we say no, then the program flow takes us back to where we were asked for the file name in the first place. This is in case we have a file but just mistyped its name. If we say we want to create the file then line 910 will open a file with the name we already gave it, line 920 closes it (it's now an empty file on disk) and line 930 takes us back to line 400 where we need to read the file primarily to get the variable N set properly. All of this happens only when the file we ask for does not exist. Normally, the file is read in in lines 390 through 450 and the code never has to get to line 840.

We will leave line 480 for now and get on with lines 490 and 500. These two lines are the focal

point of the program, and just about everything else returns you to these lines. Variable CP stands for the "Command Processor" variable and lets us know at all times what kind of action has just taken place. Its range is from zero through six. In line 490 we set CP to zero and print this program's prompt, the >>, on the screen. Line 500 asks for input into A$. Everything we enter into this program goes into A$. After that, it only goes into the B$ array if it is data for the file. The second part of line 500 says that if A$ is a null string to go back to line 490. This prevents us from getting null records in the file by pressing Enter without there being any data on the line.

Assuming that we have just entered some data into A$, line 530 checks to see if the very first character of the line is a period. If it was a period it means that this is a command (and not data) and to go somewhere to process that command. If the line does not start with a period the program assumes that the entry was a data entry and goes directly to line 570, where our record count (variable N) is incremented by one, A$ is set into the B$ array at position N, and we go right back to line 490 to display the prompt and get further information into A$.

Processing Commands

In line 530, if the left-most character of our A$ input is a period, we need to go and find out what the command is and then do what it says. The various commands require different actions, and at the end of processing of each command we will set CP and return to line 540 to take the appropriate action. Line 530 says that if A$ was a command, to go to a subroutine at 960. Let's go there.

First of all, let's assume that the command typed was a mistake (.fond, for example, instead of .find.) In this case, the program flow will drop right through lines 960 to 1010 because no match was found and line 1020 will tell us that our command was invalid. At this point, flag CP will be set to 2 and we return to line 540, where the ON CP GOTO takes us right back, again, to display the >> prompt for a different input.

The .find and .edit commands are similar up to a point. With both, we need to find the records that contain our search keyword or keywords and display those records on the screen. With .find, we will count lines and display a screenfull at a time and a prompt to continue. With .edit, we will need to give the user the opportunity to change the record, delete it or skip to the next occurrence of the keyword. The CP flag for .find is 1 and for .edit it is 4. Both of these commands will send program flow to line 1030, where we strip off the command itself and leave only the keyword or keywords. In the lines immediately following, 1040 through 1070, we check to see if we have an "and" or "or" situation. If we do, then flag F is set to indicate which one it is and we go to 1090 to separate the keywords on either side of the "and" or "or" into P$ and Q$ and then return. If the situation is not "and" or "or" line 1080 will make sure that flag F is set to zero, we make P$ equal to A$ and we return to line 540. In line 540, if CP is 1 (.find) or CP is 4 (.edit), control will go to line 600.

The code from line 600 through 730 is where the records from the file are selected and displayed and where screen paging takes place. This section of code is essentially a loop that reads the entire file and looks for our keyword or keywords. In line 620 if the flag F is zero it means that we are looking for only one keyword or phrase, which is now called P$. In line 630 if flag F is 1 it means we are looking for an "or" situation and the keywords are in P$ and Q$. In line 640 we are looking for the "and" situation. Note that in 640 we cannot use the BASIC logical operator AND as we did the OR in line 630. Instead, we need to use a nested IF...THEN, which is equivalent to the AND operation.

If our keyword or keywords are found in any record, the record is displayed on the screen with an asterisk preceding it (see line 660.) Variables C and C1 keep track of the number of lines already printed on the screen so we can page when necessary. In line 660, if CP was 4 (.edit), we need to go and give the operator an opportunity to make the editing changes. To do this we go to a subroutine at line 1160. In the subroutine at 1160, E$ is our response to the prompt about what to do. If we delete the record, line 1220 sets the current B$ (at position K) to a null string and we return. If we decide to change the record, line 1230 asks for the new line (CE$) and line 1240 makes the current B$ equal to CE$ and then we return. If we decide to skip this record, line 1250 simply returns from this subroutine without change, and if we decide to quit this mode entirely, the last two references in line 1210 will take us to line 490 and the >> prompt. We got to this subroutine from line 660. Let's go back there.

Lines 670 and 680 do some calculations on both the length of the lines and the number of lines so that we don't overfill the screen. Variable C keeps track of how many screen lines were used, while variable C1 keeps track of the actual number of

records that were found, regardless of their length. In line 720 we attempt to correct for the bad grammar of most computer programs. How often have you seen the prompt "One records were found" or something similar? In line 720 we take care of that by sensing variable C1 and printing "*record was*" or "*records were*". Because of the improbability of a file with only one record, we didn't go through this same exercise in lines 710 or 1130.

The other commands are not nearly so involved. If the command was .help, then program flow goes to line 1280. If the command was .help without an argument, line 1280 will send us to 1300, where the commands are defined. In 1440, CP is set to 5 and we return. With CP at 5 we will again get our >> prompt. If the command was .help followed by any other command, line 1280 will send us to the appropriate section for more detail on help for that command. After each, CP is set to 5 and we return. The .find command is the only one that requires more than one page to display, consequently, it has an additional prompt after the first page.

The .quit command sets CP to 3 and returns. The ON CP GOTO in line 540 will then send control to line 760, where we open the file on disk and send all records that are not null to the disk file. After this, in line 810, we RUN 100. This will clear all variables and the file currently in memory and allow us to see the menu where we can select a different file to work with or enter QUIT for the filename and get out of this program entirely.

The .save command does almost the same as the .quit command does, except now CP is equal to 6 and line 810 will take us back to line 400. In 400, we read the file just saved back into memory and we get our >> prompt back. What for? Well, for one, it eliminates any deleted records and gives space for more. For another, it saves your input on disk and prevents loss if you do something wrong or there is a power failure.

The only other command is the .file command. This one is very simple and just tells you how many records and what file you are using at the time. This command uses CP equal to 5 and only works with line 1130.

Changes you can make

This program takes up almost exactly 10,000 bytes of memory without the data file. You can reduce the program size to 5,500 bytes if you do not want the detailed help on each command. To do this, change line 1280 to: 1280 REM. Then delete 1410, 1420 and 1430. Then delete lines 1450 through 2130. This will still leave you with the abbreviated help command and give you about 4,500 more bytes in free memory for your data file.

Changes for Tandy I and III

Determine the amount of memory to clear as described earlier in this article and set that amount in line 150. Set SW to 64 and SL to 16 in line 170. Change the .DAT in line 340 and 360 to /DAT. Change line 840 to: IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0. With the full-sized program loaded, you should have about 29,500 bytes left for data on a 48K machine, which should give about 450 records if each is 64 characters long.

Changes for Tandy IV and others

Tandy IV users should change the .DAT to /DAT in lines 340 and 360. No other changes are required except for setting the max number of records as explained earlier. It turns out that with the full-sized program loaded with a Model IV, there are only 21,500 bytes left in a 128K machine.

Generally, machines using Microsoft BASIC prior to version 5.1 should follow the suggestions for the Tandy I and III, above. Some MBASIC machines will need to change the clear screen (CLS) command in several places to their particular command. There are no PRINT@ or LOCATE commands in this program. The original version of Qkey.bas was written for GW BASIC under MSDOS.

We have compiled this program using the Microsoft Quickbasic compiler. It is not that terribly slow as is, but the compiled version did run faster. If you should want to use the same compiler, be aware that you will have to set the /E switch when calling the compiler (see your Quickbasic reference manual) because of the ON ERROR GOTO statements in this program.

User notes

This was an especially interesting program to write. In spite of the rather circuitous program flow, the whole thing (without the help screens) is just over 100 lines and is very easy to use. Thanks to our Editorial Advisor, Cam Brown, for suggestions concerning the development of the program.

Since we wrote it, we have been using the program to keep a running log of possible Programming Notes and another file on editorial ideas. The most obvious use, however, was

suggested by a letter to the editor about having an index of CodeWorks articles and notes. We have put one together through Issue 9, and it is included with this article. That file will also be included with the Issue 10 menu on the download and we will keep it updated as we go along. It will also be included on our yearly diskette. Its name is CWINDEX.DAT.

We are sure you will find other good uses for the program, just as we have. Keep in mind that it is still an in-memory data base and you can have more records only if you shorten the length of each record. Our currently running series on a Random File data base will work better for larger files, but this one works just fine for short and medium sized, quick reference files. We will appreciate hearing of any novel uses you find for it. ■

```
100 REM * QKEY.BAS * A free form query database program written
110 REM * for CODEWORKS magazine, 3838 S. Warner St. Tacoma, WA
120 REM * 98409 phones 206-475-2219 voice and 206-475-2356 modem
130 REM * (c)1987 80-NW Publishing Inc. Placed in Public Domain 1987
140 REM * Please do not remove these credit lines.
150 'CLEAR 10000 ' Use only if your machine needs to clear string
    space.
160 NR=550 ' Sets max # of records. See CodeWorks Issue 10.
170 SW=80:SL=24 ' set screen width and # of lines for your machine.
180 DIM B$(NR)
190 CLS
200 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
210 PRINT"      K E Y W O R D   D A T A B A S E   P R O G R A M
220 PRINT"            a free-form, in-memory query database
230 PRINT STRING$(60,45)
240 PRINT
250 PRINT"For hardcopy output, use your print screen function.
260 PRINT"If no file exists, one will be created.
270 PRINT"The ready prompt for this program is >>
280 PRINT"Data entry is the default mode, all other entries are
290 PRINT"commands and must start with a period.
300 PRINT
310 '
320 REM * request the file name from the user *
330 PRINT"Enter .QUIT or .quit for file name to exit this program.
340 INPUT"Use which file (default is KEY.DAT)";X$
350 IF X$=".QUIT" OR X$=".quit" THEN CLS:END
360 IF X$="" THEN FF$="KEY.DAT" ELSE FF$=X$
370 ON ERROR GOTO 840
380 '
390 REM * open the file and read in the data *
400 OPEN "I",1,FF$
410 FOR M=1 TO NR
420   IF EOF(1) THEN 450
430   LINE INPUT #1,B$(M)
440 NEXT M
450 N=M-1:CLOSE
460 '
470 REM * set up the command line and prompt *
480 IF CP<>6 THEN PRINT"Type .help for help, or enter a command or
    new data."
490 CP=0:PRINT">>";
```

```
500 LINE INPUT A$:IF A$="" THEN 490
510 '
520 REM * go and process the command *
530 IF LEFT$(A$,1)="." THEN GOSUB 960 ELSE 570
540 ON CP GOTO 600,490,760,600,490,760
550 '
560 REM * If no period, it's no command, so it's data by default *
570 N=N+1:B$(N)=A$:GOTO 490
580 '
590 REM * this is where the records are selected *
600 CLS:C=0:C1=0
610 FOR K=1 TO N
620   IF F=0 THEN IF INSTR(B$(K),P$) THEN 660 ELSE 690
630   IF F=1 THEN IF INSTR(B$(K),P$) OR INSTR(B$(K),Q$) THEN 660 ELSE
      690
640   IF F=2 THEN IF INSTR(B$(K),P$) THEN IF INSTR(B$(K),Q$) THEN 660
      ELSE 690
650   GOTO 690
660   C=C+1:C1=C1+1:PRINT"* "+B$(K):IF CP=4 THEN GOSUB 1160
670   LS=LEN(B$(K)):IF LS>2*SW THEN C=C+3 ELSE IF LS>SW THEN C=C+2
680   IF C=>SL-6 THEN C=0:PRINT"---- ENTER for more, Type .quit to
      abort. ----":LINE INPUT X$:CLS:IF X$=".QUIT" OR X$=".quit" THEN
      480
690 NEXT K
700 PRINT
710 IF A$="" THEN PRINT"---- There are ";N;" records in file: ";FF$:
    PRINT"---- There is space for ";NR-N;" more.":GOTO 730
720 IF CP=1 OR CP=4 THEN IF C1=0 THEN PRINT"There are no records with
    /";A$;"/" ELSE IF C1=1 THEN PRINT"Only one record with /";A$;"/
    was found." ELSE PRINT C1" records with /";A$;"/ were found."
730 GOTO 490
740 '
750 REM * output the file and eliminate deleted records *
760 OPEN"O",1,FF$
770 FOR M=1 TO N
780   IF B$(M)="" THEN 790 ELSE PRINT #1,B$(M)
790 NEXT M
800 CLOSE
810 IF CP=6 THEN 400 ELSE RUN 100
820 '
830 REM * make a file if none exists *
840 IF ERR<>53 THEN ON ERROR GOTO 0
850 CLS:PRINT"There is no file named ";FF$
860 PRINT"Do you want one (Y/N)?"
870 C$=INKEY$:IF C$="" THEN 870
880 AN=INSTR("YyNn",C$)
890 IF AN=0 THEN 870
900 IF AN=3 OR AN=4 THEN 330
910 OPEN "O",1,FF$
920 CLOSE
930 GOTO 400
940 '
```

```
950 REM * command line processor *
960 IF LEFT$(A$,5)=".FIND" OR LEFT$(A$,5)=".find" THEN CP=1:GOTO 1030
970 IF LEFT$(A$,5)=".HELP" OR LEFT$(A$,5)=".help" THEN 1280
980 IF LEFT$(A$,5)=".QUIT" OR LEFT$(A$,5)=".quit" THEN CP=3:RETURN
990 IF LEFT$(A$,5)=".EDIT" OR LEFT$(A$,5)=".edit" THEN CP=4:GOTO 1030
1000 IF LEFT$(A$,5)=".SAVE" OR LEFT$(A$,5)=".save" THEN CP=6:RETURN
1010 IF LEFT$(A$,5)=".FILE" OR LEFT$(A$,5)=".file" THEN CP=5:GOTO
     1130
1020 PRINT A$;" is not a valid command; try again, or try .help":CP=2:
     RETURN
1030 A$=MID$(A$,7,LEN(A$))
1040 IF INSTR(A$,"(or)") THEN F=1:GOTO 1090
1050 IF INSTR(A$,"(OR)") THEN F=1:GOTO 1090
1060 IF INSTR(A$,"(and)") THEN F=2:GOTO 1090
1070 IF INSTR(A$,"(AND)") THEN F=2:GOTO 1090
1080 F=0:P$=A$:RETURN
1090 P=INSTR(A$,"("):Q=INSTR(A$,")")
1100 Q$=RIGHT$(A$,LEN(A$)-Q)
1110 P$=LEFT$(A$,P-1)
1120 RETURN
1130 PRINT"You are working with ";N;" records in the file: ";FF$:
     RETURN
1140 '
1150 REM * the skip, change or delete subroutine *
1160 IF A$="" THEN RETURN
1170 PRINT"Delete, Change, Skip or Quit this mode (D/C/S/Q)"
1180 E$=INKEY$:IF E$="" THEN 1180
1190 AB=INSTR("DdCcSsQq",E$)
1200 IF AB=0 THEN 1180
1210 ON AB GOTO 1220,1220,1230,1230,1250,1250,490,490
1220 B$(K)="":RETURN
1230 PRINT"Enter the entire new record: ":LINE INPUT CE$
1240 B$(K)=CE$:RETURN
1250 PRINT:RETURN
1260 '
1270 REM * the help feature *
1280 ON INSTR("....HELPhelpFINDfindEDITeditSAVEsaveQUITquitFILEfile",
     RIGHT$(A$,4))/4 GOTO 1300,1300,1450,1450,1750,1750,1880,1880,
     1960,1960,2060,2060
1290 '
1300 CLS:PRINT TAB(15)" ----- HELP ----- "
1310 PRINT" The following commands are available:
1320 PRINT".FIND - searches the file for a match to your input.
1330 PRINT".EDIT - allows deletes or changes to the file.
1340 PRINT".SAVE - saves the file and allows continued use.
1350 PRINT".QUIT - saves the file and allows use of another or quit.
1360 PRINT".FILE - tells which file you are working with.
1370 PRINT" At the >> prompt, you may enter any command, either in
     all
1380 PRINT" upper or all lower case preceded by a period, or simply
1390 PRINT" type in new data. Limit each record to three screen lines
1400 PRINT" or less and terminate entry with ENTER.
```

```
1410 PRINT
1420 PRINT"At the >> prompt, typing .HELP FIND (for example) will
1430 PRINT"bring up more detail on that command."
1440 CP=5:RETURN
1450 CLS:PRINT TAB(15)"---- The .FIND command ----"
1460 PRINT"At the >> prompt, you may type .find or .FIND and the
1470 PRINT"entire file will be presented, paged, with prompts to
1480 PRINT"continue. To find an item or items in the file, type:
1490 PRINT".find item  and ENTER. Since 'item' is part of 'items'
1500 PRINT"any record containing those words will be displayed, as
1510 PRINT"well as 'itemized' and other words with 'item' in them.
1520 PRINT"To isolate 'item' put a space before and after it.
1530 PRINT"Your search of the file can be as broad or narrow as
1540 PRINT"you wish it to be by choosing your search string
     carefully.
1550 PRINT"Entries should all be in affairmative form to prevent
1560 PRINT"ambiguity. Use the (or) function, see following page,
1570 PRINT"when words could be capitalized, i.e., .find
     North(or)north
1580 PRINT"
1590 PRINT"  Press ENTER for .find with AND and OR options.";:LINE
     INPUT X$
1600 CLS:PRINT TAB(15)"---- .FIND with (AND) or (OR) ----"
1610 PRINT"The same rules that apply to single word or phrase
     searches"
1620 PRINT"apply to AND and OR searches. The conjunctions, AND or OR,
1630 PRINT"must be enclosed in parens without spaces. Some examples:
1640 PRINT"  .find lumber(or)wheat      .find John (OR) Mary
1650 PRINT"                .find lumber(and) wheat
1660 PRINT"                .find river (AND) Mississippi
1670 PRINT"                .find coal and oil (or) resources
1680 PRINT"The (AND) conjunction means that BOTH words or phrases you
1690 PRINT"specify must be in the record. The (OR) conjunction will
     return
1700 PRINT"all records that contain EITHER or BOTH of your words or
1710 PRINT"phrases. Isolating words with spaces fore and aft applies
     here
1720 PRINT"much as it does with single word searches. The AND/OR
1730 PRINT"conjunctions may also be used with the .EDIT command.
1740 CP=5:RETURN
1750 CLS:PRINT TAB(15)"---- The .EDIT command ----"
1760 PRINT"At the >> prompt you may type .edit or .EDIT followed
1770 PRINT"by a search string. Use a search string unique enough
1780 PRINT"to narrow your choice to one record if possible.
1790 PRINT"You may use the AND/OR conjunctions with the .edit
1800 PRINT"command. See .HELP FIND for information on AND/OR.
1810 PRINT"After the record is found, you have the option of deleting
1820 PRINT"it, changing it, skipping to the next record if there is
1830 PRINT"more than one or terminating the edit search. When
     deleting,
1840 PRINT"the record is not deleted on disk until the file is saved
1850 PRINT"(.SAVE) or you use .QUIT. When changing a record you will
1860 PRINT"need to re-enter the entire record.
1870 CP=5:RETURN
```

```
1880 CLS:PRINT TAB(15)"---- The .SAVE command ----"
1890 PRINT"At the >> prompt you may type .save or .SAVE (with nothing
1900 PRINT"following it) and the current file in memory will be saved
1910 PRINT"to diskette. The program will then return the >> prompt
     and
1920 PRINT"you may continue working with it. The .save function will
1930 PRINT"automatically eliminate records that have been deleted and
1940 PRINT"will release the space for other input.
1950 CP=5:RETURN
1960 CLS:PRINT TAB(15)"---- The .QUIT command ----"
1970 PRINT"At the >> prompt you can type .quit or .QUIT (with nothing
1980 PRINT"following it) and the current file in memory will be saved
1990 PRINT"to disk. Deleted records will be automatically eliminated.
2000 PRINT"After the save to disk, the program will start over
2010 PRINT"and display the standard heading. You can then either quit,

2020 PRINT"or load a different file to work with. This
2030 PRINT"program will work with any ASCII file, including BASIC
2040 PRINT"programs that were saved in ASCII.
2050 CP=5:RETURN
2060 CLS:PRINT TAB(15)"---- The .FILE command ----"
2070 PRINT"At the >> prompt you can type .file or .FILE (with nothing
2080 PRINT"following it) and you will get information about the file
2090 PRINT"you are currently working with. To change to another file,
2100 PRINT"use the .QUIT command, where the current file will be
2110 PRINT"automatically saved and you will be given the opportunity
2120 PRINT"to select a different file or quit.
2130 CP=5:RETURN
```

Programming Notes

There are times in computing when you may need to know whether a number is even or odd. The logical AND operator can be used to test for odd/even. Any binary number with bit one set is automatically odd. Since all numbers in your computer are in binary form, we can AND any number with 1 and find out if that number is odd or even. Try this code:

```
10 INPUT"Enter any number";A
20 IF A AND 1 THEN PRINT "Odd" ELSE
PRINT "Even"
```

The A AND 1 is a test for logical true or false. If the number you entered is odd then it and the number 1 will both have bit one set. The statement "A AND 1" will then be true and line 20 will print "Odd." Note that if you enter a number with a decimal fraction, the test will work only with the whole number, not the fraction.

It sometimes helps to know that a "null string" is not a non-existent string but one with a length of zero. If you say that A$="" then A$ will still continue to exist, but has a length of zero. It's similar to saying that B=0. But if you print B it will show a zero, while printing a null string will show nothing on the screen.

In BASIC, the For...Next, the While...Wend and the Gosub...Return all have error messages when the two are not paired. However, If...Then may also be nested, and there must be even pairs, but there is no specific error message when they are not paired. Generally the error will be a syntax error, but sometimes no error message will be shown and your results may be very wrong.

CWINDEX.DAT - Here is how we indexed all the items in all the issues of CodeWorks through Issue 9. Using Qkey.Bas, you can pull out any type of cross-reference you like. Issue 10 updates will be included with Issue 11, and so on. The complete index will also be on the download, and will be updated.

Plot.bas, main program, issue 1, page 4, plots histograms and trend
Search.bas, main program, issue 1, page 10, finds specific variables
Notes, relational operators AND and OR, issue 1, page 11
Notes, MOD function, issue 1, page 13
Notes, interactive versus compiled BASIC, issue 1, page 13
Extr.bas, main program, issue 1, page 12, extracts program lines
Writer.bas, main program, issue 1, page 17, analyzes your writing
Maker.bas, main program, issue 1, page 25, makes data statements easy
Writer.bas, reference, issue 2, page 3, various
Halley.bas, main program, issue 2, page 5, tracks Halley's Comet
Beginning BASIC, the print statement, issue 2, page 11
Notes, DOS jargon, issue 2, page 11
Cal.bas, main program, issue 2, page 12, calendar maker program
Norris.bas, main program, issue 2, page 16, convert file types
Notes, affix date to filename, issue 2, page 17
Notes, spaces between quotes, issue 2, page 17
Notes, sort utility in UNIX and MSDOS, issue 2, page 17
Notes, using BASIC to keep notes, issue 2, page 17
Card.bas, main program, issue 2, page 18, mini-database program
Puzzler, loop counter variable, issue 2, page 25
Sorting, article w/samples, issue 2, page 26
Notes, press any key, issue 2, page 28
Notes, STRING$ to LPRINT spaces, issue 2, page 28
Notes, variable assignment, issue 2, page 29
Notes, decrementing for...next loops, issue 2, page 29
Notes, using TRON and TROFF, issue 2, page 29
Writer.bas, reference, issue 3, page 3
Card.bas, reference, issue 3, page 3
Cal.bas, reference, issue 3, page 4
Writer.bas, reference, issue 3, page 5
Card.bas, reference, issue 3, page 5
Outline.bas, main program, issue 3, page 6, a programming aid
Notes, length of variable names, issue 3, page 7
Notes, random function w/o argument, issue 3, page 7
Notes, two ways to remark, issue 3, page 7
Random, article w/samples, issue 3, page 8
Beginning BASIC, the input statement, issue 3, page 12
Puzzler, a=b=c?, issue 3, page 13
Murray.bas, main program, issue 3, page 14, calculating accuracy
Notes, change locate to print@, issue 3, page 18
Download, issue 2, page 32
Download, issue 3, page 19
Wood.bas, main program, issue 3, page 22, best fit wood cutting
Notes, Tandy .Bas vs /Bas, issue 3, page 36
Sorting, article w/samples, issue 3, page 38
Plot.bas, reference, issue 4, page 3
Outline.bas, reference, issue 4, page 3
Card.bas, reference, issue 4, page 4
Check.bas, main program, issue 4, page 5, converts Card.bas
Precomp.bas, main program, issue 4, page 7, a BASIC precompiler
Notes, locate & print@ correction, issue 4, page 11
Notes, limits of numeric constants, issue 4, page 11
Notes, the FIND utility in MSDOS, issue 4, page 11
Norris.bas, main program continued, issue 4, page 12
Beginning BASIC, For...Next loops, issue 4, page 14
Sequential files, article w/samples, issue 4, page 15
Math.bas, main program, issue 4, page 20, algebra in action
Pay.bas, main program, issue 4, page 26, a payroll program
Puzzler, making a deck of cards, issue 4, page 39
Download, issue 4, page 40
Card.bas, reference, issue 5, page 3
Precomp.bas, reference, issue 5, page 4
Wood.bas, reference, issue 5, page 4
Pay.bas, reference, issue 5, page 4
Card.bas, reference, issue 5, page 5
Wood.bas, reference w/program, issue 5, page 5
Busmod.bas, main program, issue 5, page 6, a business model
Notes, input nulls previous value, issue 5, page 14
Notes, password ideas, issue 5, page 14
Shareware, article, issue 5, page 15, how to get free software
Notes, using the MERGE command, issue 5, page 16
Beginning BASIC, binary and ASCII, issue 5, page 17
Puzzler, finding larger of 2 values, issue 5, page 18
MGsort.bas, main program, issue 5, page 20, merge/sort files
VXref.bas, main program, issue 5, page 25, a variable ref program
Notes, errors in INT function, issue 5, page 33
Notes, inkey with 2 digits, issue 5, page 33
Notes, lprint" " vs lprint, issue 5, page 33
Conv.bas, main program, issue 5, page 34, converts data types
Download, issue 5, page 38
Notes, where is issue 1?, issue 5, page 39
Card.bas, reference, issue 6, page 2
Wood.bas, reference, issue 6, page 3
Cal.bas, reference, issue 6, page 4

Technique

Using pointers to string data

Staff Project. Sometimes we get bogged down by trying to do too many things at once. Here is a technique that allows you to forget all about moving strings around with integers until you really need them. It illustrates the use of tags or pointers.

More often than not, when we have to deal with numeric data, we would like to attach some kind of name to each record. This, of course, means that at least one of the data items in a set will need to be a string variable.

There is nothing wrong with having one item be a string and others integers - until you start to manipulate the integers, as in a sort. Then, lugging that string along gets to be annoying. But if all the integer data has been re-arranged, how do you keep track of which string name belongs to what integer data?

There are several ways to do this, and here is one of them. In our hypothetical program we are entering the name, height and weight for up to 15 people. We then intend to sort them on height, from the tallest to the shortest. At the time when we enter the data for each person, from lines 190 through 270, we *know* what data belongs to which person. The loop index number, I, tells us that all data for a particular loop count belongs to one person. Subscripted array A(I,x) will hold the height and weight of each person, in A(I,1) and A(I,2), while A$(I) will hold the name of the person. If we now create a third A(I,x) position, and let it be the same as the loop count, I, we will have effectively created a pointer that will point to the name of the person to whom that data belongs, and, it will be an integer. See line 240.

Now, we can do what we want with the data, as long as we carry the pointer integer along with the other integers. We no longer need to worry about the string variable name, in fact, we can forget all about it until it's time to print out results.

So what are the advantages, if any? For one, sorting integers is always much faster than sorting strings. Moving strings around in memory very quickly causes string space to fill up with left-over pieces of useless data, and causes the so-called "garbage man" to appear to clean things up. BASIC automatically takes a "time-out" when this happens and effectively locks you out while it cleans up. Integer manipulation is much cleaner, and does not cause this type of problem.

In our sample program, between lines 300 and 390, we sort the integer data into descending order by height, making sure to switch all three integers involved when such a switch is necessary. BASICs having the SWAP command can use it to good advantage here in lines 340, 350 and 360. The syntax is: SWAP A(I,x),A(L,x), and you can then also then dispense with the T1, T2 and T3 variables.

From line 420 on, we want to print out our results. Now we need to connect the string name to the appropriate integer data. The pointer to that string name is in subscripted variable A(I,3). If you will note line 460, we can now take that pointer subscript and make the entire expression of it a *subscript* of A$. Line 460 will now put the correct name to its corresponding data.

Our sample program is small with few data entries, and the advantages of using this method may not be too apparent. On larger data arrays, however, it can make a very significant difference.

As an aside, if you want to sort by height in ascending order, change the "equal to or greater than" in line 330 to "equal to or less than." Further, if you wanted to sort on weight instead of height, change the "1" in the subscripts in line 330 to "2". The sort in this program is our old friend, the Bubble Sort, which for the purposes used here is the easiest to implement and the fastest because there are so few items. ■

```
Name            Height          Weight          Index #

Mary            70              113             2
John            67              150             1
George          62              145             3
```

Figure 1

As you can see from this sample run, the names were entered in the order of the Index #, John, then Mary and then George. That index number is what re-attaches the correct name to the data.

```
100 REM * Point.Bas * for Technique in Issue 10 *
110 '
120 DIM A$(15),A(15,3)
130 CLS
140 PRINT"Input Name, height in inches and weight at the prompts."
150 PRINT"Enter .end or .END for name to quit."
160 PRINT
170 '
180 REM * Enter the data for up to 15 people.
190 FOR I=1 TO 15
200   INPUT"Name of person  ";A$(I)
210   IF A$(I)=".end" OR A$(I)=".END" THEN 270
220   INPUT"Height in inches";A(I,1)
230   INPUT"Weight in pounds";A(I,2)
240   A(I,3)=I
250   PRINT
260 NEXT I
270 N=I-1
280 '
290 REM * Sort the data by height
300 F=0
310 FOR I=1 TO N-1
320   L=I+1
330   IF A(I,1)=>A(L,1) THEN 380
340   T1=A(I,1):A(I,1)=A(L,1):A(L,1)=T1
350   T2=A(I,2):A(I,2)=A(L,2):A(L,2)=T2
360   T3=A(I,3):A(I,3)=A(L,3):A(L,3)=T3
370   F=1
380 NEXT I
390 IF F=1 THEN 300
400 '
410 REM * Display the sorted data with the proper name attached.
420 CLS
430 PRINT"Name","Height","Weight","Index #"
440 PRINT
450 FOR I=1 TO N
460   PRINT A$(A(I,3)),A(I,1),A(I,2),A(I,3)
470 NEXT I
480 END
```

Compilers

Using one is not that difficult

C. F. Mowery, Jr., Col USAF(Ret.), Brandon, FL. The Microsoft QuickBASIC Compiler is one of the most forgiving and easy-to-use. They have come a long way, as Col. Mowery shows.

I've only been using a compiler for about three months; so I'm not an expert - and I don't know whether all compilers function the same way. But for those who are contemplating an investment in one, my experience to date might be of some help, keeping in mind that my comments are limited to the Microsoft QuickBASIC compiler for use on MS-DOS machines.

I should also mention that I know nothing about assembly languages, but I get the impression from the compiler manual that, although a knowledge of assembly language procedures is not necessary, it would facilitate learning to use a compiler - or at least help you to use it more effectively.

I had hoped that a compiler would require nothing more than having a BASIC program, e.g., TEST.BAS, to which could be applied a command such as "COMPILE TEST.BAS", and that the compiler would then automatically produce a program that was shorter and faster, and that could be run from the DOS ready prompt without first loading BASIC. I was only partially disappointed.

Compiling is a two-step process: first you compile and then you "Link." Using TEST.BAS as our hypothetical BASIC program to be compiled, first you compile it into a second file that would be named TEST.OBJ (for TEST.object), and then you link that file (don't ask me to what it is linked - it gets too complicated), and you have a third, executable file called TEST.EXE.

Although TEST.EXE can be run from the DOS level by simply typing TEST and will run as much as 30 times faster than TEST.BAS (depending on the functions in the program), the .EXE file is much larger than the .BAS file - and the .OBJ file is even bigger. After "Link" you can delete the .OBJ file to save disk space, but you had better keep the .BAS file someplace, because there is no way, unless you know machine language, to modify the .EXE file.

My compiler program is called BCOM20.EXE. So the first thing I did was rename it to COMPILE.EXE. "Compile" I can remember - the other name is ridiculous. The result is that I can, in fact, begin compilation as simply as I had hoped, i.e., by simply typing COMPILE TEST. (The Compiler assumes that the extension will be .BAS.)

Well, ok, it's not quite that simple - but almost. The .BAS file has to be in ASCII format. But that means that the BASIC program can just as easily be written and edited with a text editor, instead of using the BASIC interpreter and having to remember to save it with the ,A option.

You can also add a semicolon after the filename - COMPILE filename; - and avoid having to answer some additional prompts, such as the name of the .OBJ file and the drive to write it to.

But here's where it gets a little complicated. There are some switches that you may have to add after the semicolon. The first one to consider is /O.

In order for the final .EXE file to be executable from the DOS level, you either have to have a "run-time module" (don't ask) on the same disk as the .EXE file, or the run-time module has to be incorporated into the .EXE file during the compilation process. Since incorporation results in a very large .EXE file, I see no reason for using it, except in certain cases. The run-time module is called BRUN20.EXE but, fortunately, you don't have to remember its name - just make sure it's on the same disk as the .EXE file. On the other hand, if you want to see your .BAS file grow to huge proportions, compile it using the initial command COMPILE filename;/O.

There are two other types of switches that you do need to concern yourself with. (There are others. For example, you could write a program without line numbers or with line labels, and use the /N switch, but let's keep it simple for this discussion.)

The first type applies if you have RESUME (with ON ERROR) in your program, and the choice is simple: if you have any form other than RESUME (line number), then you must use the /X switch. If all you have is one or more RESUME (line number) statements, then use the /E switch (which makes it easy to remember: E for error.)

If you have any event-trapping routines in your program (ON KEY, ON PLAY, etc.), then you must use either the /V or /W switch. If you want the program to check for the event after each line, use

/V. If you want it to check after each statement, use /W. Since I always use multi-statement lines, I find it easier to always use /W, if the program has any event trapping. (Note that event trapping does not include the normal ON A GOTO or ON B GOSUB type of statements.)

That's all you'll usually have to remember for the first step in compiling:

1. Put a semicolon after the filename.
2. Add /O or make sure the run-time module is on the same disk as the final .EXE file,
3. Add /E or /X if you have RESUME statements, and
4. Add /V or /W if you have event trapping.

For example, if you are compiling TEST.BAS and will use the separate run-time module, and have RESUME, RESUME NEXT or RESUME 0 in the program, and the program includes event trapping and has multi-statement lines, then the command line would look like this: COMPILE TEST;/X/W.

Before compiling a program, I use my text editor to search for any occurrences of RESUME and ON to determine which switches, if any, are required.

One final complication in connection with beginning compilation requires discussion. Some of the files required for compiling, including your BASIC program, take up a lot of disk space and you may find it necessary to have the compiler files on one disk and use another for the .BAS file and the .OBJ and .EXE files produced by the compiler. Since I have 2 floppy drives and enough memory to set aside part for a RAM disk, I use drive A: for the necessary compiler programs, drive C: (the RAM disk) for the .BAS and .OBJ files, and drive B: for the .EXE file. Using the RAM disk also speeds up the compilation process.

In any case, make sure that you have enough disk space for the large .OBJ and .EXE files to be created. Then begin compilation from the default drive to which you want the .OBJ file written. For example, using the configuration mentioned above, I begin from C: and enter the command A:COMPILE TEST;. The compiling programs are then read from A:, TEST.BAS is read from the default drive, C:, and the large TEST.OBJ file is written to C:.

Once compilation is completed, and the DOS prompt has returned to the screen, it's time to "Link", and here comes another small complication that may or may not be applicable to you. If you are only "linking" the single .OBJ file, you only have to type LINK TEST (our example filename). The compiler, however, includes a file called SMALLERR.OBJ, which reduces the size of the final .EXE file by using error numbers instead of error messages. I always use it. To do so, my command line looks like this: A:LINK TEST+A:SMALLERR. The LINK file and the error module are on drive A:, and TEST.OBJ is in the default drive, C:.

You are then prompted for four separate items. Unless you are doing something other than a routine compilation, you should only have to answer one or two of them with anything other than RETURN or ENTER.

The first prompt asks for the Object modules. Since you have already named them in the command line, just press RETURN.

The second prompt asks for the location and name for the RUN file, in our case, TEST.EXE. If you want it written to the default drive, just press RETURN. In my case, since the default drive, C:, is my RAM disk, I enter B: so that the file will be written to my second floppy drive. It's not necessary to enter the filename, unless you want the .EXE file to have a different name.

The third prompt asks about the List File. Just press RETURN. (Don't ask - I don't know.)

The fourth prompt asks about the libraries. Just enter the designation for the drive containing all the compiler files, in my case, A:, and press RETURN. Then the linker will do its thing and, when finished, your DOS prompt will return to the screen.

At this point you will have a fully compiled, executable file with the .EXE extension. You may now delete the TEST.OBJ file.

That's it. Here are the commands and prompt responses, which assume that you are compiling TEST.BAS from drive C:, that the program contains no RESUME statements, that it contains no event trapping, that you have renamed the compiler program as COMPILE.EXE and will use a separate run-time module on the disk with the compiled TEST.EXE file, that you begin from drive C:, that your compiler programs are in drive A:, that you don't have or won't use the error module, and that you want TEST.EXE to be written to drive B:

```
>A:COMPILE TEST; (RETURN)
>A:LINK TEST (RETURN)

Object Modules (.OBJ): (RETURN)
Run File (TEST.EXE): B: (RETURN)
List File (NUL.MAP): (RETURN)
Libraries (.LIB): A: (RETURN)
```

Simple, huh? ■

Lotto.Bas

For impatient potential winners

Staff Project. Now that you have paid your tax through the lottery, find out fast whether or not you have won anything.

If you only have one or two Lotto tickets it's easy to see if you have won or not. During the week between Christmas and New Year, 1986, the Lotto jackpot in our state had grown to 11.6 million. It was a natural thing then, to buy a few tickets to include with a card for everyone in the family, with the idea that any one winner would share with everyone else. Looking through 20 or 30 cards then, with two sets of numbers on each, it was easy to overlook a match of three out of six (which can be redeemed for a new ticket.) This was something the computer could do faster and more thoroughly, so we wrote Lotto.bas.

The program is short and easily modified. If your state doesn't count three of six as a winner, for example, you can easily change the code to reflect that. The program design is quite simple. You create a text file which contains all of your Lotto picks. Then, when the winning numbers are announced, you enter them and the program does a quick comparison to see if you have matched three, four, five or six of the winning numbers. The winning numbers need not be entered in any special order as long as they are all there, the program will find any matches. Try the program with dummy data first to see how it works before you throw away a winner!

You can use any file editor to make the data file (Lotto.Txt.) Because we are dealing with single and double digit numbers, all single digits should be preceded with a zero. This is true both when entering your numbers into the file and also when entering the winning numbers. See Figure 1 for a sample of how the data in the file looks. It contains 10 numbers, although the program will let you have as many as 200, and if that isn't enough, you can change line 270 to put in as many numbers as memory can hold. We made our file with WordStar, then tried it with MSDOS's EDLIN, and then we used Qkey.Bas (in this issue), and they all worked fine to create the data file. The important thing with all these editors was to press Enter after each group of six digits was entered.

You don't need to tell Lotto.Bas how many numbers there are; it reads the file and figures out how many, and then tells you so on the screen. (See Figure 2)

The program contains an error trap so that if you run it without a data file it will tell you so and quit so you can first make one. The error trap is at line 560, but if your BASIC is earlier than version 5.1, remark line 560 and un-remark line 570.

After the opening screen, the first thing the program does is open the data file and read in the numbers. It figures out how many there are in line 330. You are then prompted to enter the winning numbers using the loop between 380 and 400. The next section, from lines 430 through 530, checks every digit in the winning number against every digit in all of your picks and keeps track of how many matches there are. It uses three nested loops, I, J and L. Lines 500 and 510 keep you posted on the progress and show you how many of the six were picked (if any) for each number. If your computer has sound, you may want to add a couple of BEEPS after "you got the big one" in line 500. Variable CT is a simple counter. It counts the number of wins you have, and if there are none, line 540 will tell you so.

So what happened back at Christmas time? Well, five people split the 11.6 million. All we got were a few free tickets to the next week's drawing. Even though many people here refer to the Lotto as a "Fools Tax," it does help the state educational and highway systems. ■

```
100 REM * LOTTO.BAS * CodeWorks (c)1987 80-NW Publishing
110 REM * Placed in public domain Mar 1987
120 F=0
130 CLS
140 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
150 PRINT"                  L O T T O   P R O G R A M
160 PRINT"            find your winning Lotto numbers - fast
170 PRINT STRING$(60,45)
180 PRINT
190 PRINT"The numbers on your lotto cards should be entered
200 PRINT"into a file called LOTTO.TXT with two spaces for
210 PRINT"each digit, i.e., 010305102341
220 PRINT"You can create the file with any text editor or
230 PRINT"with the program Qkey.bas in this issue (#10),
240 PRINT"or with EDLIN or any other file editor.
250 IF F=1 THEN PRINT:PRINT"After you have done that, rerun this
    program":END
260 ON ERROR GOTO 560
270 H=200:DIM A$(H)
280 OPEN "I",1,"LOTTO.TXT"
290 FOR I=1 TO H
300   IF EOF(1) THEN 330
310   INPUT #1,A$(I)
320 NEXT I
330 H=I-1
340 CLOSE
350 PRINT
360 PRINT"Enter the winning numbers, using 01 for 1, etc."
370 PRINT
380 FOR J=1 TO 6
390   PRINT"What is winning digit #";J;:INPUT B$(J)
400 NEXT J
410 PRINT
420 PRINT"Examining ";H;" numbers for winning combinations."
430 FOR I=1 TO H
440   FOR J=1 TO 12 STEP 2
450     C$=MID$(A$(I),J,2)
460     FOR L=1 TO 6
470       IF B$(L)=C$ THEN Q=Q+1
480     NEXT L
490   NEXT J
500   IF Q=6 THEN PRINT I;"-";A$(I);" YOU GOT THE BIG ONE !":CT=CT+1:
      GOTO 520
510   IF Q=>3 THEN PRINT I;"-";A$(I);" ";Q;" out of six":CT=CT+1
520   Q=0
530 NEXT I
540 IF CT=0 THEN PRINT"You didn't get anything at all.."
550 END
560 IF ERR <> 53 THEN ON ERROR GOTO 0
570 'IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0 ' for BASICs prior to
    ver. 5.1
580 F=1:CLS:GOTO 190
```

Figure 1

```
Ø1Ø3Ø5Ø71Ø12
Ø5Ø71Ø232730
1Ø1827293144
Ø1Ø3Ø5Ø91138
Ø5Ø711324041
Ø7Ø911132340
Ø6Ø712182339
Ø2Ø3Ø7131926
Ø2Ø41Ø141827
Ø32127293438
```

The sample dump of a typical Lotto.Txt file showing ten lotto picks. It is important to precede single-digit numbers with a zero, both in the .Txt file and when entering the winning digits, as shown in Figure 1, above. Almost any file or text editor can be used to create the Lotto.Txt file. Note that when a match is found, the program will print out the placement number in the file of the matching lotto number. In our example above, it found the match in the first number in the file.

Figure 2

Shows a sample run of Lotto.Bas

```
---------------------- The CodeWorks ----------------------
                  L O T T O   P R O G R A M
            find your winning Lotto numbers - fast
-----------------------------------------------------------

The numbers on your lotto cards should be entered
into a file called LOTTO.TXT with two spaces for
each digit, i.e., Ø1Ø3Ø5Ø2341
You can create the file with any text editor or
with the program Qkey.bas in this issue (#1Ø),
or with EDLIN or any other file editor.

Enter the winning numbers, using Ø1 for 1, etc.

What is winning digit # 1 ? Ø1
What is winning digit # 2 ? Ø3
What is winning digit # 3 ? Ø5
What is winning digit # 4 ? 1Ø
What is winning digit # 5 ? 22
What is winning digit # 6 ? 39

Examining  1Ø  numbers for winning combinations.
 1 -Ø1Ø3Ø5Ø71Ø12  4  out of six
```

Card1.Bas

A field change to Card.Bas

Staff Project. In engineering there are always field changes (or Advanced Design Change Notes.) Here is our first official change to Card.Bas. It allows you to specify the file, and also creates it initially if it does not exist.

If we assume that the original Card.Bas program was version 1, then what is presented here will make it version 1.1. To keep from having conflicting filenames on disk, we will call this new version Card1.Bas.

This update to Card.Bas allows you to tell the program what file to work with. In addition, it gets rid of a problem which has been with the program from its beginning. That problem was how to get the data file established in the first place. We had more calls and letters on that one than for any other program.

The listing accompanying this article shows the lines to be added, changed or deleted. Basically what we needed to do was to prompt for the data file early in the program, before that file would be accessed. Added line 125 does that. Then, in case the file did not already exist, we had to add an error trap that would allow you to create it automatically. Line 127 sets up the error trap. The lines from 530 through 548 take care of finding the proper error to act upon and then, if the error was "file not found," go ahead and ask if you want to create the file. When you answer "Y" to this question it will take a moment or two while the file is created on the diskette.

For a more thorough discussion on this type of error trap, see the article in this issue on Qkey.Bas. The way the error trap is used there and here is virtually identical.

Our file name is now contained in FF$ instead of being "hard coded" as it was in the original program. This required lines 160 and 330 to be changed to reflect the new filename variable.

In the "nice but not absolutely necessary" department, we can now tell which file you are working with, as in lines 522 and 700. Where before there was only one file you could work with, now you could be working with files of various names, so the program lets you know what it is.

If you have incorporated Card.Bas into the Mini-Card Database System (see Issue 8) then you will need to replace added line 528 in the accompanying listing with: 528 RUN "MCARD.BAS" instead of the END that is there now.

Caveats, and more Caveats

The changes listed here will increase the size of the original Card.Bas by 433 bytes. If your data files were already pushing the limit of free memory, then the revised program and the data file may not fit in memory at one time. In that case, use the original version of Card.Bas to eliminate about four or five records. The exact number will depend upon how many characters you are using and the number of fields in each record.

By correcting the problem of file naming and initialization, we may be creating a new problem. For example, if you have several versions of Card.Bas with a different number of fields in each, this revision will give you quite unpredictible results. If the new Card1.Bas is set up for eight fields and the data file has only five fields, you would be reading the first five fields correctly but picking up three more fields from the next record in the file. In short, it would be out of whack. The number of fields specified in the Card1.Bas program must match the number of fields in the data file it will be using. You can, of course, have several different versions of Card1.Bas, each with the correct number of fields to correspond to your various data files.

Changes for various machines

In line 532 we used the PRINT CHR$(12) as a clear screen command to keep the changes consistent with the original program. Most machines will need to change this to CLS, others (notably CP/M BASICs) may need to change the CHR$ number to whatever they use. Incidentally, while checking this out, we found that GW BASIC (MSDOS) will respond correctly to either the PRINT CHR$(12) or the CLS - they both do the very same thing.

If you have a BASIC prior to version 5.1 you may need to change line 530 in the accompanying listing to: 530 IF (ERR/2)+1<>54 THEN ON ERROR GOTO 0. Some of the machines that would require this change are the Tandy I and III.

The Listing

You may be wondering how we got the format for the listing with this article. Several issues ago, we asked about programs that compare for differences. Several ideas were sent in, and we looked them over and then wrote our own version of a compare program. Its output is what you see in the listing here. It has turned out to be an extremely useful program, especially when additions or changes are made to an existing program, such as with Card.Bas. We will publish the compare program in a future issue because we are certain you will appreciate its utility as much as we do.

Other possible changes

There are several other changes that could be added to Card.Bas. One of them would be a two-level sort, so that you could sort names within cities, for example. Another change could be made to right-set numeric fields so that they would sort properly. You may already have noticed the the number 2 will sort *after* the number 113 because the leading spaces before the digit 2 are not padded with blanks.

Because of the fact that Card.Bas is an in-memory program, any changes that make it larger will take away space from the data file. If you are content with small files and want to see such changes, let us know. Otherwise, if file size is a problem, we urge you to follow our currently-running series on Random files by Terry. ●

```
Name of original file? card.bas
Name of revised file ? cardl.bas
Do you want line printer output too (Y/N)
Processing...
Added---> 125 INPUT"What data file do you wish to work with ";FF$
Added---> 127 ON ERROR GOTO 530
Changed-> 160 OPEN "O",1,FF$
Changed-> 330 OPEN "I",1,FF$
Added---> 522 PRINT FF$;" FILE IS SAVED, SESSION ENDED."
Added---> 524 PRINT
Added---> 526 PRINT"THE FILE NOW CONTAINS <";J-1;"> RECORDS"
Added---> 528 END
Changed-> 530 IF ERR <> 53 THEN ON ERROR GOTO Ø
Added---> 532 PRINT CHR$(12):PRINT"There is no file named ";FF$
Added---> 534 PRINT"Do you want one (y/n)?"
Added---> 536 CC$=INKEY$:IF CC$="" THEN 536
Added---> 538 AN=INSTR("YyNn",CC$)
Changed-> 540 IF AN=Ø THEN 536 ELSE IF AN=3 OR AN=4 THEN 528
Added---> 542 OPEN"O",1,FF$
Added---> 544 PRINT #1,"ZZZ"
Added---> 546 CLOSE #1
Added---> 548 GOTO 330
Deleted-> 550
Deleted-> 560
Changed-> 670 PRINT"          C A R D   F I L E   P R O G R A M   Ver. 1.1
Changed-> 700 PRINT"            You are currently working with ";FF$

There are  239  lines in the original program (card.bas)
There are  251  lines in the revised program (cardl.bas)

 14  Lines were ADDED in the revised program.
 6   Lines were CHANGED in the revised program.
 2   Lines in the original were DELETED in the revised program.

Done processing.
Ok
```

CodeWorks 1st Year Programs on Diskette

All the programs published during the 1st year of CodeWorks magazine are available on the following 5¼ soft-sector formats:

- ☐ PC/MS-DOS GW-BASIC 40 track DSDD
- ☐ CP/M MBASIC specify format and computer type:
- ☐ TRSDOS Model I 35 track SDSS
- ☐ TRSDOS Model III 1.3 40 track SSDD
- ☐ TRSDOS IV 6.2 40 track SSDD

If your format is not covered by the above, please inquire and we will try to accommodate you.

Only 50¢ per program!

Check "Disk Order" in the order form below and return this page or a photocopy to us. Diskettes will be sent 1st Class postpaid.

Special Subscriber price $20.00

Note: We cannot provide hard-sector formats. Non-subscriber price $30.00

Subscription ORDER FORM

387

NOTE: The entire set of (7) first year issues is still available at $24.95. Please specify "First Year" if you are ordering this set.

Computer type:

Comments:

Please enter my one year subscription to *CodeWorks* at $24.95. I understand that this price includes access to download programs at NO EXTRA charge.

- ☐ New subscription
- ☐ Renewal subscription
- ☐ Check or MO enclosed.
- ☐ Bill me later.
- ☐ Charge to my VISA/MasterCard # _____ Exp. date _____
- ☐ **Diskette Order**

Referred by _____ # _____

Please Print Clearly:

Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to: ***Codeworks***
3838 South Warner St.
Tacoma, WA 98409

Charge card orders may be called in (206) 475-2219 between 9 AM and 4 PM weekdays, Pacific time.

Puzzler

Puzzler 9 was a bit of code that generated numbers suspiciously looking like Pi. The question was, would it? could it? and why?

It appears at first glance that this algorithm can generate Pi to any number of decimal places. It does not. The procedure will result in a close approximation, but the result can never equal Pi. The SQR function in BASIC is limited to single precision (even if the variables are defined as being double precision.) Even assuming that the SQR function could be as accurate as you desired; the result will still not be equal to Pi.

Pi is a non-constructible number. It has been proven that it cannot be generated in any finite combination of the operations of +, -, *, /, or SQR. One of the classical problems in geometry is the "squaring of a circle" using nothing more than a straight-edge and a compass. A circle has been squared when you have constructed a square whose area is equal to that of a given circle. To be able to do so, you have to be able to construct the square root of Pi. That requires Pi to be constructible. In 1882, Lindemann finally proved that Pi was transcendental, therefore non-algebraic, therefore non-constructible.

For more information on Pi, we recommend *The History of Pi, Second Edition,* by Petr Beckmann, St. Martin's Press, New York, 1971. For a short discussion on the classical construction problems and the algebra of constructible numbers, see Appendix A in *Elements of Analytic Geometry and Linear Transformations,* by Paul J. Kelly and Ernst G. Straus, Scott Foresman & Company, 1970.

Robert R. Keegan, of Fayetteville, AR, wrote to say he didn't have an answer but that he had modified the code slightly (to show as much double precision as possible) and that he could now see the problem more clearly.

Richard Oberdorfer (the originator of the puzzler) wrote to say: "I am not into science fiction but I like mathematics and computers. I did some algebra on this to try to figure it out but it got so complicated and long that I quit. It is fascinating to me how a little computer program produces Pi to so many significant figures but seems to neither converge nor diverge to/from the value of Pi. What needs to be done is to carry X in the For...Next loop out to 100, 200 or even more. This requires more than the 16 significant places that BASIC is capable of. I think there is a BASIC that supports 50 or 100 places (ZBASIC?) or the program could be modified to use strings. Maybe I'll try that sometime."

Puzzlers will no longer occupy a place of their own in the magazine. When anything qualifying as a good puzzler comes along (either from you or from ourselves) you can look for it in the Forum section of CodeWorks. ■

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
US Postage
PAID
Permit No. 774
Tacoma, WA

CodeWorks

Issue 11 May/June 1987

CONTENTS

CodeWorks

Issue 11 May/June 1987

Editor/Publisher
Irv Schmidt
Associate Editor
Terry R. Dettmann
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Advisor
Al Mashburn

CodeWorks, 3838 S. Warner St. Tacoma, WA 98409

Telephones
(206) 475-2219 (voice)
(206) 475-2356 (modem download)

300/1200 baud, 8 bits, no parity and 1 stop bit

Authors: We constantly seek material from contributors. Send your material (double spaced, upper/lower case) and allow 4 to 6 weeks for editorial review. You may send IBM-PC compatible diskettes (please save your programs in ASCII format.) Also send a hard copy listing of the program and article. Media will be returned if return postage is provided. Compensation will be made for works which are accepted for publication. **CodeWorks** pays upon acceptance rather than on publication.

Subscription price: $24.95 per year (six issues.) A subscription year runs from Nov/Dec through Sep/Oct. Anyone subscribing during the current subscription year receives all issues for that year. Not available outside the United States Zip codes. **VISA and Master Card orders are accepted by mail or phone (206) 475-2219. Charge card orders may also be left via our on-line download system (206) 475-2356.**

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in the United States of America. Bulk rate postage is paid at Tacoma, Washington.

Sample Copies: If you have a friend who would like to see a copy of CodeWorks, just send the name and address and we will send a sample copy at no cost.

Editor's Notes

You have probably noticed that in this issue, and in several past issues, there are reader-written programs. We think it's refreshing to see the efforts of some of you and to pass them along to others. Not all of them conform to our style of programming - but then we will be the first to admit that ours is not the only style.

Somewhere, a while back, we said that CodeWorks was dedicated to problem solving through programming. It still is. If you have a program that solves a particular problem that seems applicable to the readership at large, we would be happy to review it and possibly include it in the magazine. Variety, of course, is the spice of life and we appreciate sampling the different flavors.

It's not that we are running out of ideas here at all. In fact, we are currently working on programs that may not appear until sometime next year. Terry's on-going series on random files promises to be exciting. Given a well-designed database, there is no end to the number of interesting application programs that can be written to work in conjunction with it, including, but not limited to: inventory, accounting and mailing list programs. In addition, of course, it will easily handle the video tape/record collection database so many of you have asked for. The Card.Bas program could handle those also, but may be too limiting because it is an in-memory program. For that matter, so could Qkey.Bas but it also is an in-memory program.

Those of you who have been with us since the beginning will probably remember the program Wood.Bas in Issue 3. Although we didn't bill it as such, it was an "expert" system in that it didn't work with an algorithm, but instead worked with an heuristic (rule of thumb approach.) If you will remember, it attempted to show the best way to cut plywood, paper, cloth or sheet metal. It was one of the toughest problems we tackled, and took about three man-months to complete.

Lately, we have been thinking about another such program, almost the reverse of Wood.Bas, that would take various size items and fit them into a specified space. You may well ask where such a program would find application. The first which comes to mind would be in composition and layout of newspapers and magazine articles, where you have several columns of type varying in length that must be fitted onto a standard page size. Fortunately, even though this sounds difficult enough, there are only two dimensions to work with. We don't even want to start thinking about a program that would tell how best to pack various sized smaller boxes into a larger shipping box!

At any rate, the type layout problem has us scratching our collective heads. How do we approach this problem? It sounds simple at first (just like Wood.Bas did) but when you get into trying it you find the obstacles. We haven't even found a good rule of thumb yet, but have decided that the problem boils down to an algorithm which will look at the various length sticks of type and decide what combination of them will just fit a page (or pages) of n column inches.

Somehow, bending over a light-table, it just seems to happen by itself. But trying to follow your thought processes in doing it is rather elusive, and makes you marvel at the super-computer which we call the brain.

Now there's a puzzler to work on.

On another subject, which I don't care to belabor: It is very encouraging to see the efforts you are making to help our circulation problem. Thank you, and keep up the good work.

Irv

Forum

An Open Forum for Questions & Comments

Corrections and Fixes

You are probably wondering about our funny terminology on page 36 of Issue 10 (last issue.) Figure 1 refers to figure 1, *above*. We originally had the two figures on that page reversed, and after proofing decided they would look better the other way around. So we switched them and the figure numbers and completely forgot to read the text that went with them. Oops!

Qkey.Bas will not allow you to delete the last record in a file if you are working with BASIC prior to version 5.1. This was called to our attention by Mr. G. L. Dailey of Jacksonville, Florida, and sure enough, when we tried it on a Tandy III it didn't. We then tried it on MBASIC under CP/M and it works as written, as well as under GW BASIC, where it was originally written. We haven't yet figured out why you can't delete the last record in a file on some machines, but in any case, here is a fix so that it will. In Qkey.Bas, move line 760 to 765 and then add a new line 760 that simply says: 760 KILL FF$. Killing the old file before writing out the new one seems a bit gross, but it works.

While on the subject of Qkey - did you know it can be used to keep track of all sorts of miscellaneous reminders? Things that need to be renewed periodically are especially easy to forget, like homeowner's insurance, driver licenses, life insurance premiums and, of course, magazine subscriptions. Then there are anniversaries and birthdays. "Jot" them down in Qkey along with the month and day, and then at the beginning of the month just .find May (for example) and see all the good stuff you need to do and remember in May. We have found that, as in all programs like this one, consistency is important. If you forgot how you entered a particular type of item just search for one like it and the form you used to enter it will be readily apparent. It even stays on the screen while you add information to the file.

After wondering out loud in the last issue about the use of the Mini-Card Database system we have heard from many of you and apparently there is some misunderstanding. Here is another quick rundown of how it works: **Card.Bas** still works like it always did. **Form.Bas** is used to make just one small file called the "Report Definition File" which is used to tell Form.Bas how to format the data file created with Card.Bas. See page 31 in Issue 8 for samples of the report definition file. It also lets you make changes to report definition files you have already made. **Rcard.Bas** is used to print the report. It uses the report definition file created by Form.Bas to tell *how* to print the records you created with Card.Bas. Form.Bas and Rcard.Bas together provide the means of printing reports in any manner you choose. **Mcard.Bas** is simply a small menu program which ties all the programs in the system together and makes it easy to go from one to another.

I finally received my long awaited Jan/Feb issue. Guess the Post Office got a tad behind. I downloaded Poker7.Bas and was wondering if you plan to list any patches for it so it would run on the Tandy IV? It looks like a well written program but appears to be for an MS DOS machine. I am really pleased with the ease and availability of your download. I have never had any problem what-so-ever with it. Enjoy your magazine.

Bill Baker
Independence, MO

The changes for Poker and the Tandy IV are in this issue, including all the fixes and additions to make it print the suits of the cards.

First, I'm delighted with the CodeWorks program listings and the clarity of the explanations in the program documentation. Even a BASIC novice of my age (66) can understand it. The programs work well on my NCR Decision Mate V using MS DOS 2.11 and GW BASIC.

Second, I have a request for a calendar program that can be annotated with recurring events (birthdays and anniversaries) and with one-time events such as an appointment for a haircut. The recurring events should be retainable by month and date so that they won't have to be keyed in for each printing or for each year. The one-time events should be automatically dropped when preparing next year's calendar.

Please keep up the good work.

John A. Jakab
Dayton, OH

Since yours is not the first such request for an appointment calendar, we dug through our files and came up with one - almost. It needs some revisions and updates but does just about what you ask for. It will probably require a 132 character line printer (or an 8" printer in condensed mode) and should appear in these pages in an upcoming issue. And thanks for the good comments - we try to aim at the "adult novice" with our material. Actually, I have a mythical character named Joe Magarac (Mag-R-Rack) who lives somewhere in mid-America to whom I address most of what I write. It's sort of like public speakers picking out someone in the back of the room to talk to when delivering an address. If Joe can understand it, anyone can.

Following up on a message left on the download, I find that every program that I download comes in with the error message of DIRECT STATEMENT IN FILE. I'm not sure that this is my inexpert use of my download program or your program.

Thanks for the time and consideration. You have a great magazine. Don't feel that you are wasting your space on those pieces for beginners. Some of us still need them.

G. Hill
Reno, NV

When downloading, many times you will get parts of the interchange between your and our computers into the file you save. These prompts do not have line numbers and cause BASIC to give you the error message. A blank line in the file can cause the same error message. The programs you download from us are all in ASCII format, which means they can be loaded with any text editor or word processor. By doing that, you can see the errant lines and, with your text editor or word processor, remove them and save the file back to disk. They should then load and run properly under BASIC.

I have a question you might be able to answer for me. When I was using the older TRS-80 Model III and later the Model 4, all my programs were written in BASIC, and the output was always directed to the screen with print statements. Whenever I wanted a hard copy of the output, all I had to do was to insert the SYSTEM"ROUTE *DO *PR" in the appropriate places in the BASIC program, and SYSTEM"RESET *PR" when I wanted the output to stop going to the printer.

I have now converted all my programs to MS-DOS and this BASIC does not support the above commands. In order to print anything I now have to change all the PRINT statements to LPRINT. There must be a comparable command, or even a POKE command to toggle the printer on and off from a BASIC program. Can you help?

One more thing, what is GW- BASIC?

Harry E. Guaspari
Rome, NY

We have not been able to find a similar command in GW-BASIC to route screen information to the printer. You probably know that the Shift/Print key will print the current screen to the printer. Also, the CTRL/Print keys will direct all screen activity to both the screen and the line printer. CTRL/Print is a toggle, and you must press them again to stop printing to both places.

GW-BASIC apparently stands for "Graphics/Windows BASIC" although many people call it "Gee Whiz BASIC."

I am having some difficulty on my Model III in your Poker.Bas program showing card suits as suggested by Robert Hood (This appears on page 21 of Issue 9.) I find that I get a toggling into kena characters on alternate runs of the program. There must be a way to control this problem.

W. P. Frakes
Cuyahoga Falls, OH

It shouldn't do that if you have the POKE statement in line 140. We couldn't make it happen on the Model III, but in getting the Model IV program ready for this issue we did run into the very same problem, mostly because we didn't know the correct place to POKE and were using the PRINT CHR$(21) statement to get to the alternate character set. Ours actually toggled three ways, first nothing, then Japanese characters and then the ones we wanted. The alternate character sets are nice to have, but it would also be nice if they stayed there once you select them.

...Everything in Check.Bas works OK on my PCjr except (sorting on number amounts.) Page 5 of the Mar/Apr 86 issue of CodeWorks states "You can then sort on any of these items and print out a neat list." I cannot find what I am doing wrong...

Howard W. Clother
North Java, NY

You are not doing anything wrong and it is not your computer. It's the program. Computers sort on ASCII values with the space being the lowest value. So, numbers of different lengths will not

*come out right. You could enter leading zeros in front of every number your enter, but why not let the computer do the work? Let's say that you want the fourth field (D$) to be numeric in either Card.Bas or Check.Bas and you will want that field to sort properly. Here I will refer to the original line numbers in Card.Bas. D$(I) is input at line 930. Add a line 935 that says: 935 D$(I)*=RIGHT$(" "+D$(I),8) which effectively moves the number you entered to the right end of a field that is 8 spaces long. The three blank spaces between the quote marks give the number space to grow to the left, otherwise they will always be blanks and when this field is sorted the numbers will sort properly. The number 8 is arbitrary, if you use larger numbers make it equal to or larger than the largest number you expect to use. Now, since you can also enter information in the edit mode, you will need to fix the corresponding edit-input line too. That's line 1480. Change it to 1480 D$(I)=U$:D$(I)=RIGHT$(" "+D$(I),8):GOTO 1530. After this, any number you enter in D$(I) will be justified right, with leading blanks and will sort according to the rules. You can make this change on as many numerical fields as you wish, we only used D$(I) as an example. This change has the same effect in sequential files as the RSET (right-set) command does in random files. Note also that if the number you enter is larger than the field length you allowed, it will truncate the left end of it.

...when I read the letter from Bob Dowling in Issue 10 I had to fire up the Word Processor! Even if CodeWorks were the most expensive magazine in the USA, it's worth every penny. Most computer magazines cost anywhere from three to four dollars per issue and they're full of ads. I work for a printer, so I have a good idea of the costs involved in putting out a publication like CodeWorks. Since you don't have advertising revenues to help meet these costs, the wonder is not that CodeWorks costs so much, it's that it costs so little! CodeWorks contains nothing but programs, and useful programs at that, how can you complain about the price? If you found only one program per year that was useful, you've gotten full value for your money. For instance, how much would you pay for a commercial program that would make program listings like Plist.Bas from Issue 6?

I subscribed to CodeWorks because the programs are well-written, and wonder of wonders there's not a single strand of spaghetti in the lot. I've often wondered how some of the magazines that claim to teach programming can keep a straight face when they expound on the virtues of well-structured, easy to read code when you only have to look at their examples to see they don't follow their own advice. It was the quality of the programs in (CodeWorks) that got me to subscribe. What keeps me coming back is the friendly attitude of the staff, and the speed with which problems are solved...

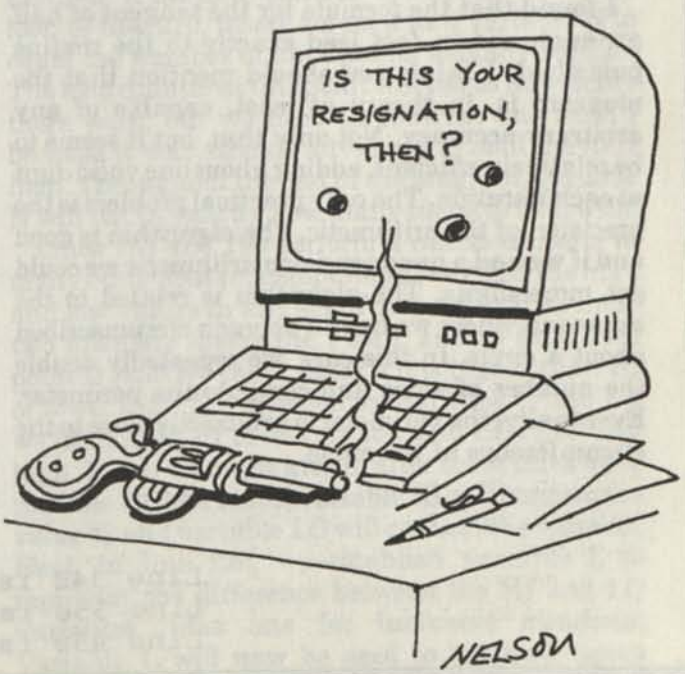
Irene P. Governale
T.S.U.G. News
Port Jervis, NY

Finding your letter on top of the stack on a Monday morning was a more than delightful way to start the week. Thanks. Irv

---

Did you know that you can get a
*** FREE ***
diskette of all the second year programs from CodeWorks?

It's simple. Just put your name on the order form on page 39 where it says "Referred by" and give a few copies of it to friends. When we receive just two such orders with your name on them you will go into our little black book of debts and you will receive the diskette when we issue them next fall.



Puzzler

More on Pi and a new one

Well, just when we wanted to relegate puzzlers to the Forum section, we got an idea for a puzzler of our own and an interesting letter in reference to Puzzler 9. That, and one page left over after making the dummy for this issue, was an irresistible combination. First, the letter.

Sirs:

I have been interested in your puzzlers. In the case of Puzzler 9, I did verify that it certainly was accurate to the extent of single precision BASIC and if we use a valid square root routine, it will produce more valid digits. At that point I was willing to let it lay until I saw what you had to say on the back of Issue 10. It just did not seem right that the program could get that close if it was not a true convergent process capable of arbitrary accuracy.

I mentioned the problem to one of my friends, Mr. Fred Hallden, of Huntington, New York. Fred seemed to agree that the routine must be one that is theoretically capable of arbitrarily close results. He seemed to think that it was based on polygons. He made several attempts to reproduce the result using formulas for the sine of half an angle. Finally, he aroused my curiosity to the point where I started playing with the problem.

I found that the formula for the tangent of half an angle did in fact lead exactly to the routine published. I think you should mention that the program is, in theory at least, capable of any arbitrary accuracy. Not only that, but it seems to be relatively efficient, adding about one valid digit at each iteration. The only practical problem is the precision of the arithmetic. The algorithm is good and if we had a quad precision arithmetic we could get more digits. The algorithm is related to the approach where we have a polygon circumscribed about a circle. In this case, we repeatedly double the number of sides and compute the perimeter. Eventually, the perimeter is arbitrarily close to the circumference of the circle.

Assuming you do not publish the (enclosed) writeup, I would appreciate it if you would send a copy to Mr. Oberdorfer.

Warren D. White
Ridge, New York

Mr. Oberdorfer has been sent a copy of the enclosed material which shows the development of the math and a sample printout, a total of five pages. Any reader interested can also have a copy by dropping us a note to that effect. - Irv

Puzzler 10

This is a programming puzzler we ran across one day recently. It came up during the development of a program flow reference program we were working on. Assume the program has already gone through the target program and found all the GOSUB references and put them into two arrays. The two arrays are then sorted with the sort being on the B(I) array. Given the two arrays, which contain the GOSUB line number references:

A(I)	B(I)
220	340
230	340
170	350
390	450
410	450
430	450
480	450
510	450

what BASIC code will read them and print output like that in figure 1? Does it sound simple? Give it a try. Use any method or command available in your version of BASIC and send us your solution. We will print the best of them. ●

Figure 1

```
Line 340 is called from: 220, 230
Line 350 is called from: 170
Line 450 is called from: 390, 410, 430, 480, 510
```

Beginning Basic

Graphing two sets of data at once

In the last issue we did some rather elementary graphs, using symbols and techniques available on any computer not having special graphics capability. In this issue we want to turn the graph 90 degrees counter-clockwise to make it read the way graphs are normally read. We also want to put some grid marks in it and be able to graph more than one variable. As you can see from the program listing, it takes considerably more code than our earlier attempt but the effort is well worth it and the few modules of code that work it all out are not difficult to follow.

The method we will use to build this graph is to create an array of strings, each one as long as the number of data points. Initially, these strings will be filled with blanks. After the array of blank spaces is made, we can then go through them and insert the proper symbols at the proper places. While we are making the array with blank spaces we can insert grid marks at the appropriate places. If we do that first, then any data point coinciding with a grid mark will overwrite that grid mark, and that's exactly what we want it to do.

While we read the data from the data statements, we will need to know the high and low values so that we can set the scale of the graph. This will also tell us how many blank lines there will be in the array, and that number will be determined by the difference between the high value and the low value. This will insure that our data will always fill the available graph space. Naturally, any widely divergent data point could throw the scale off in one direction or the other, but this is true of any type of graph and is not unique to this method.

Program description

The program is made up of six small modules starting at line 100. Each module contributes something to the completed graph. The program flow is essentially top-down and the order in which the modules are executed is important. Lines 100 to 130 do some necessary initialization. Line 110 sets the dimensions for three arrays. These are the D array, which will hold the first set of data points, the E array, which will hold the second set of data points and the A$ array, which will be our array of blank lines. Since we have less than 50 data points (actually there are 49) we dimension these arrays at 50.

Line 120 is a format line which we will use later, after the graph has been printed. It is a simple string line (FM$) with points from zero to 50. In line 130 we initialize variable HI to be as *low* as possible, and variable LO to be as *high* as possible. We'll see why in the next section.

Read data, find Hi/Lo

The lines from 160 to 220 perform three major tasks. They read the first set of data into the D array, find out how many points there were, and also find the highest and lowest values in the data set. Additionally, in line 220, we establish variable L to be equal to the difference between the high and low values, adding one so that we include both ends.

The loop between lines 160 and 200 reads the first set of data points. How do we know it will read the first set and not the second set? Because this is the first READ statement the program flow comes to and the data pointer will therefore point to the first line of the program containing a DATA statement (at line 12).

At the same time that we read D in line 170, we check to see if that data point is equal to -1, which will signify the end of that section of data, and if that data point is *not* -1, then we insert that data point into the D array at the loop counter (I) location. If the data point is -1, we jump out of the loop to line 210, where we set a new variable N to equal the number of data points actually read in. The loop counter at this point will tell us how many there were, but we need to subtract one from I because it was already advanced to read the *next* item. Throughout the rest of the program, variable N will tell us exactly how many items to deal with.

Lines 180 and 190 perform a function useful in many other areas of computing. They find the high and low values in any set of data. Note that these two lines are *inside* the I loop, and as each data point is read we check to see if it is higher than HI or lower than LO. Remember that back in line 130 we initialized HI to be very low and LO to be very high. In our example graph, after these lines have read the 49 data items, variable HI will contain the value 42 and variable LO will contain the value 23. Next, in line 220, we establish variable L to represent the difference between the HI and LO variables, plus one for inclusive numbers. Variable L will now be used to tell how many

horizontal lines (and consequently, blank A$ lines) the graph will contain. In our example case, there will be 19 lines, ranging from a high of 42 to a low of 23. This way there will be a space for every data point, but as we will see later, there may be a value between HI and LO that does not have an associated data item.

2nd Data set & Optional lines

BASIC keeps an invisible data pointer which you and I don't see. It keeps track of the *next* data to read, and as you may have guessed, it is now pointing to the data in line 30. (A BASIC "RESTORE" command would have reset the data pointer to line 12, but that's another subject for a later issue.) Lines 250 to 270 read the second set of data points and puts them into the E array. There may be fewer points in this array than in the first one, or there may be more. If there are more, the excess will be ignored because we are only going to use the first 49 of them. While the first data set will become the symbol "o"; this set will become "x" in the graph.

Line 300 is an optional print line to display the values we have found so far. It's nice as a check-out line, and can be removed if you like. If you remove it though, change the reference to line 300 in line 260 to 330.

Make blank lines and grid marks

We are now ready to make the blank A$ lines and put tick marks into them. The set of loops between 330 and 380 do it for us. Variable L is the number of lines we need. Variable N tells us how long the line of blanks must be. In line 340, each A$(I) is created by making it N+1 spaces long and filling it with blanks (STRING$(N+1," "). It had to be N+1 characters long to get that last column of dashes above the 5 in Figure 1.

As soon as we have created the first of these blank string lines, we go through it again with the J loop between lines 350 and 370. Here, we step down the blank line 10 spaces at a time and insert a dash. The mechanism we use to insert this dash is the MID$ statement in line 360. This statement simply says that going down the length of A$(I), at position J, for only one character, insert the dash at that point. The index J, of course, will only be 10, 20, 30, 40 and then 50, so the dash will only appear at those positions. Notice that the column of dashes above the zero in Figure 1 are not put there by this section of code, but are added later.

Inserting the data points

Now that we have the blank strings set up with tick marks every ten spaces, it's time to put the actual data points into the array. Lines 410 through 460 handle it for both data sets, using the same loop (and why not?). Let's look in detail at how this is going to happen. The X loop in line 410 counts from one to N (the number of data items.) The I loop, inside the X loop, counts from one to the number of blank lines we created. For every increment of X, I will count from one to L, in effect, scanning the entire graph from left to right and from bottom to top. Every time the count in X and I happens to coincide with either the data value for "o" or "x", then the appropriate symbol will be inserted at that position. If the "o" happens to be at the same point as a grid mark it will overwrite the grid mark, and if the "x" is at the same point as an "o" or a grid mark it will overwrite them. This is because of the *order* in which we enter these data points into the blank strings. The MID$ (mid-string) statement works here as described earlier. Going through line 430 one time should show how it works. Let's say that the X count is at one and I is at two. The D(X) array at D(1) contains the number 24. Now in line 430 it says that if I, currently at 2, plus LO, which is 23, less one, is equal to 24 then insert the "o" at that position. Well, 2 plus 23 less one is 24, so that position gets an "o". While we are counting index I from one to L, we may as well find the position for the "x" as well and insert it. Line 440 does that. Depending on the speed of your computer, it will take a noticeable amount of time to fill this array.

Printing it out

What we now have in memory is 19 lines of A$, each with tick marks, an "x" and an "o" in the proper place. We can add the other frills as we print it out. The first frill we want to add on printout is the actual value to the left. To do this, we first clear the screen to give us some work space. Then, in line 500, we set variable J equal to the HI value. Now we go into a loop that counts the lines from the highest line number to the lowest. We have to do this or the graph will be up-side-down. For each A$(I), in line 520, we first print the value of J, then at TAB 9 we print another reference dash, then starting at TAB 10 we print out the A$ for that loop count (I). After each, we decrement the J count and do it all again for the next A$(I), and presto! there is our graph.

When we reach the bottom of the graph we print FM$, which was defined way up in line 120, and it puts the bottom scale line on. Then we give the graph a name and we are done.

Additional Notes

This program automatically scales the vertical height of the graph. The top line will always be the highest data value you use, and the bottom line will always be the smallest data value you use. *All numbers between the high and low will be included in the graph scale.* Note that you may have data lines like this: DATA 20,22,,23 and the space between those two successive commas will count as a data point but will be blank, or a null data item.

If your data is in larger numbers, you may want to scale it upon reading it. To do that, in lines 170 and 260, right after the READ D (or READ E), put D=D/S, where S is the factor to divide by. Line 170, for example, may look like this: 170 READ D:D=D/3:IF D=-1 etc. If your graph scrolls on the video screen, you may want to send its output to your line printer.

The techniques used in this program all represent ways to get things done. They are not unique to this program. Finding HI and LO, for example, is used in many other programs and can possibly be "lifted" right out of this one and put into another. The exact syntax may not be the same but the *idea* is. Note that from the last issue, we have only added two more commands, MID$ and DIM. MID$ made all the difference in this program compared to the ones in the last issue. Play with this idea. Change one thing at a time and note the differences. Explore the reasons why things happen and change. You will learn without ever knowing you did, and that's fun. ■

```
10 REM * 1st set of data points *
12 DATA 24,25,26,27,26,25,24
14 DATA 25,26,28,30,32,34,35
16 DATA 33,32,31,30,31,29,28
18 DATA 27,26,24,23,23,24,25
20 DATA 27,28,29,30,32,33,33
22 DATA 34,35,36,37,37,36,36
24 DATA 37,38,39,40,41,41,42,-1
26 '
28 REM * 2nd set of data points *
30 DATA 38,39,40,40,41,42,41
32 DATA 40,39,38,39,39,39,38
34 DATA 37,37,36,36,35,34,34
36 DATA 33,33,33,32,32,32,31
38 DATA 31,31,30,30,30,30,29
40 DATA 29,29,30,30,31,32,33
42 DATA 34,35,36,37,38,39,40,-1
44 '
100 REM * Graph2.bas * for Beg Basic Issue 11 CodeWorks *
110 DIM D(50),E(50),A$(50)
120 FM$="0....!....1....!....2....!....3....!....4....!....5
130 HI=1E-09:LO=1E+09
140 '
150 REM * read 1st set of data to find how many and hi/lo values *
160 FOR I=1 TO 50
170   READ D:IF D=-1 THEN 210 ELSE D(I)=D
180   IF D(I)>HI THEN HI=D(I)
190   IF D(I)<LO THEN LO=D(I)
200 NEXT I
210 N=I-1
220 L=(HI-LO)+1
230 '
240 REM * read 2nd set of data (should be = or < than 1st set) *
250 FOR I=1 TO 50
260   READ E:IF E=-1 THEN 300 ELSE E(I)=E
270 NEXT I
280 '
```

```
290 REM * Optional print out to see values *
300 PRINT"hi=";HI;"lo=";LO;"total=";N;"hi-lo=";HI-LO
310 '
320 REM * make dummy strings with grid marks every 10 *
330 FOR I=1 TO L
340   A$(I)=STRING$(N+1," ")
350   FOR J=10 TO N+1 STEP 10
360     MID$(A$(I),J,1)="-"
370   NEXT J
380 NEXT I
390 '
400 REM * put the data into the dummy strings *
410 FOR X=1 TO N
420   FOR I=1 TO L
430     IF I+LO-1=D(X) THEN MID$(A$(I),X,1)="o"
440     IF I+LO-1=E(X) THEN MID$(A$(I),X,1)="x"
450   NEXT I
460 NEXT X
470 '
480 REM * print out the completed A$ array *
490 CLS
500 J=HI
510 FOR I=L TO 1 STEP -1
520   PRINT J;TAB(9);"-";TAB(10);A$(I)
530   J=J-1
540 NEXT I
550 '
560 REM * Add the bottom scale line (FM$) *
570 PRINT TAB(9);FM$
580 PRINT TAB(20)"Beginning Basic Graph Sample Run"
590 END
```

```
run
hi= 42 lo= 23 total= 49 hi-lo= 19
 42      -     x   -         -         -         -         -       o-
 41      -    x x  -         -         -         -         -      oo -
 40      -  xx   x -         -         -         -         -    o  x-
 39      - x      x-xxx      -         -         -         -   o  x -
 38      -x        x   x     -         -         -         -  o  x  -
 37      -         -    xx   -         -         -         - oo  o  x  -
 36      -         -      xx -         -         -       o -oo  x     -
 35      -         -  o     x-         -         -     o   -    x     -
 34      -         -  o       xx       -         -   o     -  x       -
 33      -         -    o     - xxx    -         -  oo     - x        -
 32      -         - o   o     -    xxx  -  o     -x                  -
 31      -         -       o o-       xxx         x                   -
 30      -         -o        o -         -xxxx   xx-                  -
 29      -         -           o         -o   xxx  -                  -
 28      -         o           -o        o        -                   -
 27      -   o     -           - o       o-       -                   -
 26      -  o o    o-          -  o      -        -                   -
 25      - o   o o -           -         o -      -                   -
 24      -o      o -           -   o  o  -        -                   -
 23      -         -           -    oo   -        -                   -
         0....1....1....1....2....1....3....1....4....1....5
                    Beginning Basic Graph Sample Run
```

Figure 1

Random Files

an input screen demo program for random files

Terry R. Dettmann, Associate Editor In this installment Terry shows how an input screen can be made. It also automatically fields the random file, based on two "configuration" files, which adds versatility to the program. The ideas presented in this issue will be installed in the Randemo.Bas series in the next issue.

In this issue we are going to prepare for some major changes to the Randemo.Bas series. The changes will be discussed in this issue and applied in the next. They will make it possible to create input screens and data formats from "configuration" files. First, let's look at what configuration files are.

Most programs are written to perform some task. They usually do that task well, but when you want something slightly different, the program itself must be either modified or rewritten. You then end up with two (or more) versions of the same program, each to do a specific job. It seems sensible then, to write a program in general terms and let it get its specific instructions from a small file that you can create and edit quickly. Such a file would be called a configuration file. The advantage, of course, is that the same main program can have many configuration files and therefore be able to perform in many specific ways instead of just one.

Configuration files take many forms. The most common form is simply a listing of parameters that the main program will use during its execution. For example, the next change to Randemo.Bas (and the sample Screen.Bas program in this issue) will use two configuration files. One of them lets you change the screen display any way you want *without rewriting the program*. Since the screen information tells what data will go into the file, the other configuration file will "map" the fields of information into the random file.

Both of these files will be created using any standard editor. You could use Wordstar, Microsoft Word, Scripsit, or some other editor you are familiar with, so long as you can create unformatted ASCII files. Call them TEST.MAP and TEST.SCN, and later, when you use Randemo5.Bas and it asks for the file name and you tell it TEST, it will use these two files to set up the program.

Let's take a look at the basic technique for this addition and see how it works.

Basic Technique

First let's deal with the MAP file. In order to lay out the map file, we should decide what we're going to need to know about the data in the data file. For each item, we'll need to know the following:

- A field designation for identifying the information
- The location of the field within the data record
- The length of the field in the data record

For convenience, we'll use a number to refer to the field within the program since it will make array references easy to work with, but we'll include a name for each field just so we can read the map.

To make the map file easy to work with, we'll set it up as a simple ASCII file with a line for each data field. We'll include the following information in each line:

Key word FIELD
Field Number
Field Name
Field Length
Field Location

Each piece of information will be separated from the others by a colon (:). As we read each line with the program, we can split it up using the colons to get out the information we need. The key word FIELD is included to allow us to add other information in the map file later. For now, this will be it.

The screen file will show the screen as it is to be displayed. We'll use it to locate the fields on the screen by coding the field number into the screen file. We'll need some sort of code to recognize that the number represents the location of a field. For that, we'll use an asterisk (*).

To represent field 3 in the screen, we'll put in the marker *3 at the location we want the field to appear. This makes it easy to create screens for the data entry without having to worry about the exact location of the fields. When creating screens with an editor, you have to be careful to not exceed the number of lines on the screen or the screen width. This is pretty easy since you can SEE what the screen will look like as you create it.

Let's do a simple example to illustrate how we can get a screen display from a MAP and SCREEN file.

The Screen.Bas sample program

The Screen.Bas program shows how we can make a screen display and control it using just the information in the screen and map files. Let's tour the program and see how it accomplishes this task.

Lines 10-50 Program Initialization
Here we set up the program for operation. We define the array SC$(24) to hold the lines of the screen display. Array XY(20,3) will hold display information on up to 20 fields for the screen display. These fields will be arranged as follows:

0 - the length of the field
1 - the X location of the field (row)
2 - the Y location of the field (column)
3 - the record location of the field

Next we define the characters corresponding to the arrow keys for up, down, left and right. As defined in the program, they are identical to the standard definitions for Wordstar (CTRL-E = UP, CTRL-X = DOWN, CTRL-D = RIGHT, CTRL-S = LEFT). Define them according to your computer's reference manual to match your keys or use the Keycode.Bas program to determine what the codes for the keys are.

On MSDOS or PCDOS machines the arrow key codes are *not* a single character as they are on other machines. We'll see how to decode them later in this series.

Lines 40 and 50 define some initial values for the variables:

CC - current cursor position
CL - last cursor position
CX - maximum cursor position
SL - screen length (number of lines)
SW - screen width (number of columns)

With these values defined, we can define the mapping and screen in line 190. Subroutine 2000 reads the file Test.Scn and decodes the lines one at a time to set up the screen display. As it reads lines, it calls subroutine 2050 to see if there are any fields defined in the screen display. If we find any (using the 'INSTR' function), we use the location found to enter them in the 'XY' array.

Subroutine 3000 reads the Test.Map file to determine how long the fields are and other information about them. The MAP file is built just as we indicated above. The subroutine at line 1400 decodes the lines by breaking them up using the colons (:) in the lines.

With the fields broken up, we call subroutine 3100 to decode the information about the field and store it in the 'XY' data array. Subroutine 3100 is laid out to allow defining other information to be decoded in the map file.

The main loop of the program (lines 200-250) controls the basic logic of the program. We start by displaying the screen and the empty data fields, and then waiting for character input. Just before inputting a character, we place a 2 character cursor on the screen using the combination of an equal sign followed by a greater than sign (=>).

If the character we input is CTRL-C (CHR$(3)), we end the program, otherwise, we call subroutine 2200 to decode the key stroke and then return to line 220 to keep going.

Subroutine 1000, which reads single keystrokes, has a special addition (line 1015), just for MSDOS machines. On these types of machines, arrow keys give a 2 character string in response to an INKEY$ function. Subroutine 1200 decodes this second character to determine which arrow key was pressed.

Subroutine 1300 places the cursor at a given row and column position on the screen. The rows are numbered from 1 to SL, the columns are numbered from 1 to SW. On some Tandy machines, this would be replaced with a 'PRINT@' statement.

The subroutine at 1400 breaks a line into up to 10 fields by locating the colons (:) in the line and placing each section in the array BL$. NB is the number of fields found in the line.

We've already mentioned subroutine 2000. Subroutine 2050 supports it by decoding the lines for field positions. Subroutine 2100 displays the raw screen lines, subroutine 2150 places the cursor on the screen by first clearing the last place it was and then placing it in the new place.

Subroutine 2200 interprets the keystrokes by defining the up or left arrow to go upwards on the screen and the down or right arrow to go down on

the screen. Subroutine 2300 changes the current cursor position to the previous field. If there is no previous position (CC < 1), then the position is set to the first field. Subroutine 2400 moves down on the screen until it gets down to the last defined field. These two subroutines make some very fundamental assumptions about the way the screen is defined:

1. Field number '1' is defined
2. All field numbers up to the last field are defined, there are no breaks

Subroutine 2500 goes through the screen positions and places periods on the screen for each position in the data field.

When we put all of these together, we find that the program will start up and display the data screen and place the cursor at the first field. Pressing the arrow keys will move the cursor up and down on the screen. The two configuration files, Test.Map and Test.Scn must be available for Screen.Bas to read. After you have had a chance to make the program work, try modifying the screen file to display the fields in other locations. Notice that while you can have them appear in any order on the screen, the program will always move to them in order by field number.

In the next issue we will incorporate the ideas presented here into the Random file series. ■

Demo Program: Screen.Bas

```
10 REM --- Demonstration Screen Display Control
20 DIM SC$(24), XY(20,3)
30 UP$=CHR$(5):DN$=CHR$(24):RT$=CHR$(4):LF$=CHR$(19)
40 CC=1:CX = 1:CL = 1
190 GOSUB 2000:GOSUB 3000
200 REM --- main loop
210 GOSUB 2100:GOSUB 2500
220 GOSUB 2150:GOSUB 1000
230 IF C$=CHR$(3) THEN CLS:PRINT"Bye Y'all":END
240 GOSUB 2200
250 GOTO 220
1000 REM --- input a character
1010 C$=INKEY$:IF C$="" THEN 1010
1015 IF LEN(C$)>1 THEN GOSUB 1200
1020 RETURN
1200 REM --- look for arrows
1210 C = ASC(MID$(C$,2,1))
1220 IF C=72 THEN C$=UP$ ELSE IF C=77 THEN C$=RT$
1230 IF C=80 THEN C$=DN$ ELSE IF C=75 THEN C$=LF$
1240 RETURN
1300 REM --- GOTO XY ROUTINE
1310 LOCATE X,Y:RETURN
1400 REM --- break line
1410 FOR K=1 TO 10:BL$(K)="":NEXT K
1420 JN$=IN$:NB=1
1430 K = INSTR(JN$,":"):IF K=0 THEN BL$(NB)=JN$:RETURN
1440    BL$(NB) = MID$(JN$,1,K-1)
1450    NB = NB + 1
1460    JN$ = MID$(JN$,K+1)
1470 GOTO 1430
2000 REM --- load screen display
2005 OPEN "I",1,"TEST.SCN"
2010 FOR I=1 TO 24:SC$(I)=""
2015    IF EOF(1) THEN 2030
2020       LINE INPUT#1,SC$(I):IN$=SC$(I):GOSUB 2050
2030 NEXT I
```

```
2035 CLOSE#1
2040 RETURN
2050 REM --- CHECK FOR FIELDS IN SCREEN LINE
2055 K = 0
2060 K = INSTR(K+1,IN$,"*")
2065 IF K<=0 THEN RETURN
2070 N = VAL(MID$(IN$,K+1)):XY(N,1)=I:XY(N,2)=K
2075 IF N>CX THEN CX = N
2080 GOTO 2060
2100 REM --- display screen
2105 CLS
2110 FOR I=1 TO 24:X=I:Y=1:GOSUB 1300
2115    PRINT SC$(I);
2120 NEXT I
2125 RETURN
2150 REM --- PLACE CURSOR
2155 X = XY(CL,1):Y = XY(CL,2) - 2
2160 GOSUB 1300:PRINT "  ";
2165 X = XY(CC,1):Y = XY(CC,2) - 2
2170 GOSUB 1300:PRINT "=>";
2180 RETURN
2200 REM --- interpret keystroke
2230 IF C$=UP$ OR C$=LF$ THEN GOSUB 2300
2240 IF C$=DN$ OR C$=RT$ THEN GOSUB 2400
2250 RETURN
2300 REM --- move up on page
2305 CL = CC:CC = CC - 1:IF CC<1 THEN CC = 1
2310 RETURN
2400 REM --- move down on page
2405 CL = CC:CC = CC + 1:IF CC>CX THEN CC = CX
2410 RETURN
2500 REM --- place blank fields on the screen
2510 FOR I=1 TO CX:X=XY(I,1):Y=XY(I,2):GOSUB 1300
2515    PRINT STRING$(XY(I,0),".");
2520 NEXT I
2525 RETURN
3000 REM --- read data map
3005 OPEN"I",1,"TEST.MAP"
3010 IF EOF(1) THEN 3035
3015    LINE INPUT#1,IN$
3020    GOSUB 1400
3025    GOSUB 3100
3030 GOTO 3010
3035 CLOSE#1
3040 RETURN
3100 REM --- decode map line
3105 IF BL$(1)="RECLEN" THEN RL = VAL(BL$(2)):RETURN
3110 IF BL$(1)="FIELD" THEN GOSUB 3200:RETURN
3115 RETURN
3200 REM --- define a field
3210 NF = VAL(BL$(2)):FL = VAL(BL$(4)):FP = VAL(BL$(5))
3220 XY(NF,0)=FL:XY(NF,3)=FP
3230 RETURN
```

Model changes

Tandy IV: Change the period to a slash in lines 2005 and 3005.
Change line 1310 to: PRINT@ ((X-1),(Y-1)),,:RETURN
The arrow key values for line 30 are: Up = 11, Down = 10, right = 9, left = 8.

Tandy I and III:
Change the period to a slash in lines 2005 and 3005.
Add line 15 CLEAR 1000
Change line 1310 to PRINT@ ((X-1)*64)+(Y-1),,:RETURN
In lines 2010 and 2110, change the loop limit from 24 to 16.
The arrow key codes for line 30 are: Up = 91, down = 10, right = 9, left = 8.
Note that on the Model I there is no control key and no way to simulate it. Use BREAK instead.

```
FIELD:1:NAME:26:1
FIELD:2:ADDRESS:26:27
FIELD:3:CITY:15:53
FIELD:4:STATE:2:68
FIELD:5:ZIP:9:70
FIELD:6:PHONE:12:79
FIELD:7:COMMENT:30:91
```

Test.Map

```
Screen Display Demo

          NAME:          *1
          ADDRESS:       *2
          CITY:          *3
          STATE:         *4
          ZIP:           *5

          PHONE:         *6
          COMMENT:       *7

UP/DOWN arrows move up/down
USE CTRL-C to quit
the program.
```

Test.Scn

Create the two files, Test.map and Test.scn, with your text editor or word processor exactly as above, including titles and prompts in Test.scn. Name the files as they are named here; Screen.bas uses both of them. The figure at the bottom of this page shows a screen dump of the output of Screen.bas and how it should appear if you used these two files.

Keycode.Bas

As a part of putting together the screen control program, it became necessary to learn how some of the keys returned ASCII codes to the INKEY$ routine. You'll need to do the same. The short program KEYCODE.BAS has been provided to help you find out.

```
10 REM - Keycode.Bas
20 C$=INKEY$:IF C$="" THEN 20
30 FOR I=1 TO LEN(C$)
40    PRINT ASC(MID$(C$,I,1));
50 NEXT I
60 PRINT
70 GOTO 20
```

The Keycode.Bas program is a very short program to tell you the ASCII code returned by the INKEY$ routine. It will also handle MSDOS machines where some keystrokes give more than one ASCII character for INKEY$.

The program starts in line 20 by waiting for you to type a character. As long as you type nothing, the routine will cycle here in what is called a 'BUSY WAIT' operation. It constantly checks for characters, eating time and doing nothing else.

Whenever a key is hit, the program cycles through all of the characters in C$ and prints the ASCII code for the character. After printing all of the characters returned by INKEY$, the routine moves to the next screen line and waits for another character. It's simple but effective.

Use the Keycode.Bas program to determine the codes returned for your arrow keys, your backspace key, and so forth. These are needed in Screen.Bas in this article and in Randemo5.Bas which will appear in the next issue.

```
Screen Display Demo

          NAME:          =>.........................
          ADDRESS:         .........................
          CITY:            ..............
          STATE:           ..
          ZIP:             .........

          PHONE:           ............
          COMMENT:         ..............................

UP/DOWN arrows move up/down
USE CTRL-C to quit
the program.
```

Ira.Bas & Retire.Bas

Build and then deplete your nest egg

Staff Project These two programs show first how funds accumulate for retirement and then how easy those funds are to deplete. The sidebar, provided by Mr. Al Hurst, gives an historical look at how IRA started and has developed.

Next to the concept of compound interest the Individual Retirement Account (IRA) is probably one of the best ideas that has come along in a good long time. It is a triple-threat idea in that it encourages you to save up for retirement, accumulates tax-free interest that is compounded and allows you to reduce the adjusted gross income you declare on your income tax each year by the amount of your annual contribution. In spite of the so-called "sweeping" changes in the 1986 Tax Reform, if you are married and your adjusted gross income is less than $40,000 you can contribute and deduct just as much as in past years. Between $40,000 and $49,999 you can contribute lesser amounts than before, and above $50,000 you can no longer utilize the IRA (see the sidebar to this article.)

The program, IRA.Bas, is designed to show you what your contribution will be worth in future years. Figure 1 shows a hypothetical run (you can use it to check the operation of the program after you have typed it in) of a single person who "rolled over" an initial amount of $50,000 into his IRA and contributes $2,000 per year for ten more years. The assumption, of course, is that he maintains an interest earning rate at 10 percent. If he "self-directed" his IRA and was in a stock Mutual Fund for the past few years he would probably have exceeded that 10 percent figure. As you can see from figure 1, time and compound interest really act together after a few years to produce a rather sizable sum. Try age 30, with no initial payment and 34 years at $1,000 at 10 percent and see what happens.

The little summary at the end of the program is for information only and is not as factual as it seems. There are definite rules about how you can take payments from your IRA after you reach retirement age. See your local financial expert for details on this.

The Ira.Bas Program

We checked out several "future value" type of programs and found that most of them wouldn't handle initial deposits, or *not* handle regular deposits. So we worked with the formula until we got it to work with or without an initial deposit and with or without regular deposits. Therefore, the program can also be used to figure the future value of various investments, not just IRA's.

There are two techniques used in this program (and in Retire.Bas as well) that you can use to good advantage in other programs. The first of these is a method of entering a number that can be either a whole number or in decimal form, as in entering an interest amount. Check line 220 in Ira.Bas. It asks the user to input the interest. If the interest is 20 percent you can enter either 20 or .2 - it makes no difference to the program. Line 230 then takes that input and if it is less than one it lets it be, otherwise, it divides it by 100. The same idea is used three times in Retire.Bas. It makes your input value work correctly no matter how you enter it.

The other technique, also used in both programs, shows how to make formatted print statements line up with headings. Look at lines 300 and 310 in Ira.Bas. Line 300 is a formatted string that will be used with a PRINT USING command later. It tells where the numbers will all go and will round dollar values to two places and add the dollar sign. By making a variable of the headings that go with this statement, and placing it directly above or below the formatted string in line 300, you can make all your headings come out exactly where you want them. Now when you want the heading, all you need to is to PRINT X1$ (as in line 430.) The list of items to print using X$ is in line 400. Now, if you need to move an item over to one side or the other a little bit, you can do it easily in lines 300 and 310 and keep everything lined up nicely. Simple things, to be sure, but nice when you need them.

Neither program employs a line print function. You can use your screen print function for that or, if you have it, re-route your screen to the printer. That can be done on MS-DOS machines by pressing the CTRL and Print keys at the same time. (Press them again to discontinue sending to

the printer - it's a toggle. By contrast, the Shift and Print keys will only send the current screen to the printer and is not a toggle.)

The Retire.Bas Program

Now that you have accumulated your nest egg for all these many years the big question is how far will it go? There are so many variables connected with that question it is difficult to answer. Retire.Bas, however, will give you some idea of what to expect. You can run it many times with different values. This program takes taxes and inflation into consideration, and tells how much you can draw per month for how many years before your funds are depleted. Figure 2 starts with the amount we found in figure 1. Our optimistic hypothetical person decides that he will get 15 percent interest on his investment (maybe he knows something we don't) and that the inflation rate will be 9 percent. He also figures he will be paying 10 percent of his income to taxes.

Here is how to read the output (see figure 2). Let's take the top item, starting at the left with $1100. What this says is that he can draw $1100 per month to start, increasing to $6165 (the third column) in 20.6 years and that at the end of that time there will be a residual amount in his account of $4509. That third column (in our example) shows the effect of 9 percent inflation for the last month of those 20.6 years.

There are, of course, several fuzzy decisions being made here. The first is: How long does he expect to live? That's a tough question to answer. The second is: Won't his interest rate increase as inflation rises? Probably, but we don't know if one will offset the other. The other question is taxes: Who knows what the tax bite will be in 20 years?

These programs are certainly not meant to be a retirement planning exercise. But, if they get you thinking about it you may want to see a qualified expert in that field. ■

```
100 REM * IRA.BAS * Written for CodeWorks Magazine (c)1987
110 REM * 3838 S. Warner St. Tacoma, WA 98409 (206)475-2219
120 REM * Placed in public domain 1987
130 'CLEAR 1000 ' Use only if you need to clear string space.
140 CLS
150 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
160 PRINT"          I R A  P R O J E C T I O N  P R O G R A M
170 PRINT"            Projects the future value of your IRA
180 PRINT STRING$(60,45)
190 PRINT
200 PRINT"The program assumes annual compounding."
210 INPUT"What is your current age ";AG
220 INPUT"What is the annual rate of interest you expect ";I
230 IF I<1 THEN 240 ELSE I=I/100
240 INPUT"What is the initial balance in your account ";SV:FV=FV+SV
250 INPUT"How much will you add to your investment each year ";P
260 INPUT"For how many years do you wish to project figures ";Y
270 CLS
280 '
290 REM * format the print image *
300 X$= "###     ###      #.###     $####,###.##      $######,###.##"
310 X1$=" YR     AGE       INT%     AMT INVESTED        FUTURE VALUE"
320 PRINT X1$
330 '
340 REM * do the future value calculation *
350 AI=SV
360 FOR J=1 TO Y
370   FV=FV+P
```

```
380    FV=FV+(FV*I)
390    AI=AI+P
400    PRINT USING X$;J;J+AG;I;AI;FV
410 NEXT J
420 PRINT
430 PRINT X1$
440 PRINT
450 '
460 REM * summarize the findings *
470 PRINT"You are now ";AG+J;" years old and you can withdraw the"
480 PRINT USING "$##,####.##";FV*I;:PRINT" interest per year and
    maintain"
490 PRINT"the principal at ";USING "$####,####.##";FV
500 PRINT"These figures are not adjusted for taxes or inflation,"
510 PRINT"and assume the earned interest is constant at ";I*100;"
    percent."
520 END 'of program
```

Figure 1

```
run
---------------------- The CodeWorks ----------------------
          I R A   P R O J E C T I O N   P R O G R A M
            Projects the future value of your IRA
-----------------------------------------------------------

The program assumes annual compounding.
What is your current age ? 54
What is the annual rate of interest you expect ? 10
What is the initial balance in your account ? 50000
How much will you add to your investment each year ? 2000
For how many years do you wish to project figures ? 10
 YR    AGE      INT%     AMT INVESTED        FUTURE VALUE
  1     55     0.100    $  52,000.00      $    57,200.00
  2     56     0.100    $  54,000.00      $    65,120.00
  3     57     0.100    $  56,000.00      $    73,832.00
  4     58     0.100    $  58,000.00      $    83,415.21
  5     59     0.100    $  60,000.00      $    93,956.72
  6     60     0.100    $  62,000.00      $   105,552.40
  7     61     0.100    $  64,000.00      $   118,307.60
  8     62     0.100    $  66,000.00      $   132,338.40
  9     63     0.100    $  68,000.00      $   147,772.30
 10     64     0.100    $  70,000.00      $   164,749.50

 YR    AGE      INT%     AMT INVESTED        FUTURE VALUE

You are now  65  years old and you can withdraw the
$ 16,474.95 interest per year and maintain
the principal at $  164,749.50
These figures are not adjusted for taxes or inflation,
and assume the earned interest is constant at  10  percent.
Ok
```

```
100 REM * Retire.Bas * CodeWorks Magazine, 3838 S Warner St
110 REM * Tacoma, WA 98409  (206) 475-2219
120 'CLEAR 1000 ' Use only if you need to clear string space.
130 DEFINT J
140 F1$="Init. Monthly   # of     Final Mo. Adj."
150 F2$="Withdrawal      Years    for inflation     Residual"
160 F3$="$$####           ##.#    $$#####       $$######"
170 CLS
180 PRINT STRING$(22,45)" The CodeWorks ";STRING$(23,45)
190 PRINT"                RETIREMENT REQUIREMENTS PROGRAM
200 PRINT"               or, how far will the nest egg go?
210 PRINT STRING$(60,45)
220 PRINT
230 INPUT"What is the amount of money accumulated at retirement ";P
240 INPUT"What interest rate do you expect to get then ";IR
250 IF IR<1 THEN 260 ELSE IR=IR/100
260 INPUT"What inflation rate do you expect at retirement ";FL
270 IF FL<1 THEN 280 ELSE FL=FL/100
280 FL=1+FL
290 INPUT"What tax percentage do you expect to pay then ";TB
300 IF TB<1 THEN 310 ELSE TB=TB/100
310 TB=1-TB
320 D=IR/12
330 E=P*D
340 A=100
350 FOR I=1 TO A
360   IF A>(.5*E) THEN I=A:A=A-100:GOTO 380
370   A=A+100
380 NEXT I
390 PRINT
400 PRINT F1$
410 PRINT F2$
420 PRINT
430 FOR I=1 TO A
440   A=A+100
450   C=A:B=P
460   FOR J=1 TO 480
470     B=(B*(1+(D*TB)))-C ' Take out the tax
480     IF B<0 THEN J=480:B=B+C:GOTO 510
490     M=J
500     IF INT(M/12)*12=M THEN C=C*FL ' Inflation factor
510   NEXT J
520   YR=M/12
530   IF YR=>40 THEN 560
540   PRINT USING F3$;A,YR,C,B
550   IF YR=<5 THEN 570
560 NEXT I
570 END 'of program
```

```
run
---------------------- The CodeWorks ----------------------
                RETIREMENT REQUIREMENTS PROGRAM
              or, how far will the nest egg go?
-----------------------------------------------------------

What is the amount of money accumulated at retirement ? 164749.5
What interest rate do you expect to get then ? 15
What inflation rate do you expect at retirement ? 9
What tax percentage do you expect to pay then ? 10

Init. Monthly    # of      Final Mo. Adj.
Withdrawal       Years     for inflation      Residual

 $1100           20.6        $6165             $4509
 $1200           17.8        $5193             $3884
 $1300           15.8        $4735             $3828
 $1400           14.2        $4678             $1261
 $1500           12.8        $4219             $2181
 $1600           11.8        $4129             $1880
 $1700           10.8        $4025             $2120
 $1800           10.1        $4261              $776
 $1900            9.4        $4127               $82
 $2000            8.8        $3985               $73
 $2100            8.3        $4184             $3188
 $2200            7.8        $4022             $1167
 $2300            7.4        $4204             $1217
 $2400            7.0        $4387             $4116
 $2500            6.7        $4193             $3605
 $2600            6.4        $4360              $791
 $2700            6.1        $4528             $4098
 $2800            5.8        $4308             $3767
 $2900            5.6        $4462             $4420
 $3000            5.4        $4616             $1764
 $3100            5.3        $4770               $32
 $3200            5.0        $4924             $4123
Ok
```

Figure 2

Programming Notes

If you use DEFSTR A-C near the beginning of a program, then all variables *starting* with A, B or C will be declared as string variables and you needn't show them as A$, B$ or C$ in the remainder of the program. This is handy and it saves a little space, but can make following a program, especially a long one, difficult in that you must remember what variables were declared as strings. In the interest of clarity, we usually do not use it.

It is interesting to note that POKE is a BASIC statement, or command, while PEEK is a function which will return a value. The syntax for POKE is: POKE #####,NN where # is a memory location and N is a number to put there. The syntax for PEEK is: PEEK (#####) or PRINT PEEK (#####) where # is a memory location and if you PRINT PEEK you will be shown the number stored at that location. The value returned will be an integer in the range 0 to 255 (one byte.)

What is an IRA?

The Employee Retirement and Income Security Act (ERISA) of 1974 first made the Individual Retirement Account (IRA) available to the public on January 1, 1975. This Act allowed workers who did not have a pension or retirement plan of any kind, regardless of the benefit size, to have their own individual qualified plans. The Act authorized such eligible employees to contribute 15 percent of their income, not to exceed $1,500 annually, to such an arrangement. Later a "Spousal IRA" became available which authorized an additional $250 deductible contribution for a married couple where the spouse was not receiving employment compensation.

Since this situation was intended by Congress to encourage saving for retirement purposes, the participants enjoyed tax-deferred growth on the investments as well as a tax deduction for their contributions. The primary appeal for many people has been the tax deduction with less regard for the retirement aspects. Congress, having foreseen such bias, inserted a penalty of 10 percent of any amounts withdrawn before age fifty-nine and one-half. In addition, the full amount withdrawn becomes taxable income, including the penalty. This served to emphasize that the IRA's middle name is "Retirement." The idea of a short term tax break was effectively ruled out by this penalty. There are some exceptions to this rule, for example, in the case of total disability. Penalties are also levied for over-contribution to one's IRA.

The vehicles available for IRA investment during the accumulation period ranged widely. For example, banks offered certificates of deposit (CD's) of varying maturities. Insurance companies issued annuity products. Investment houses made available many forms of investments such as stocks, bonds, money market accounts, etc. Additionally, the so-called "Self-Directed IRA" was popular with people who enjoyed setting up and monitoring their own mix of investment vehicles. Originally, certain investments were not allowed, such as gold and silver bullion and certain collectibles, but some of these restrictions were later lifted.

One frequently misunderstood aspect of IRA's was the mandatory withdrawal requirement after age seventy and one-half. Several methods are available and include lump sum payments, annuities and scheduled withdrawal payments. Initially, the latter required that the funds be totally exhausted by the person's life expectancy. This time period was determined by the IRS tables of mortality. Life expectancy does not necessarily coincide with one's actual life span, though, and some people have had concern that they might outlive their income from their IRA. Thus, the inclination to "stretch" the income from the IRA accumulation or use only the interest has been expressed as desirable. Again, foreseeing this, the law extracted a fifty percent penalty for under-withdrawals from the schedule. This problem has somewhat been alleviated today by the ability to recalculate, on an annual basis, this life expectancy.

Another approach to overcome this problem is the lifetime annuity. Annuities, basically, are the systematic liquidation of a sum of money over the life spans of many participants, some of whom live very short lives and others who live longer. This plan has the advantage of providing an income that cannot be out-lived, but that has no residue to be passed on to heirs when death occurs. Because this concept is not acceptable to some people, many variations on this theme have been devised. They include, but are not limited to, joint-and-survivor annuities, lifetime annuities with a "period certain" (meaning that the annuity will pay for a specified minimum time - the period certain - after the annuity commences regardless of when a person dies) or for life if longer than the period certain.

The situation remained largely unchanged until January 1, 1982 when the Economic Recovery Tax Act of 1981 (ETRA) expanded eligibility to include anyone with "earned income." The percentage also increased to 100 percent of earned income, not to exceed $2,000 annually. The deductible amount was increased to $2,250 total contribution for both persons when the Spousal IRA was used. Each person in a Spousal IRA program must have a separate account for his and her contribution but, unlike the previous arrangement that required an equal split of contributions, now any combination can be used so long as neither contract input exceeds $2,000 annually. The requirement that the spouse have no income from employment during the year remained intact. If the spouse does have earned income, of course, he or she could have a regular IRA without being subject to the Spousal IRA rules.

Apparently the cycle has somewhat come back to the point of beginning by again limiting unrestricted deductibility of IRA contributions to those with no pension plans. Employees with pension plans can still fully deduct their IRA contributions if their adjusted gross income is less than $40,000 for a married couple filing jointly or $25,000 for a single employee. A married employee filing singly and others may have partial deduction available depending on income levels.

Non-deductible contributions are available for people who wish to have tax-deferred growth on their money. This concept is not new, however, in that tax-deferred annuities have been available for many years and effectively accomplish the same task.

This is not intended to be an exhaustive dissertation on the IRA concept nor advice regarding taxes. For advice relative to your particular situation, consult an expert in this field.

(Our thanks to Albert Hurst, CLU, of Tacoma, Washington for supplying this information on IRA's.)

Addressr.Bas

Print and log addresses and labels

Mark Gardner, North Hollywood, CA An easier way to put addresses on envelopes or one-wide labels. Lets you make notes about what you sent to whom for a permanent annotated record.

What the problem was

The computer was supposed to bring an era of assistance to mundane tasks that face us humans. Great! But getting a computer to do something simple (though useful) like addressing an envelope seemed to be really complex.

You had to have the address in a data base. You had to put the data base in some mode that would format the addresses for printing in the correct place on the envelope, etc., etc. If you didn't have the data base and just wanted a couple of addresses, maybe you signed onto BASIC and used LPRINT commands. Maybe you just used a pen. Addressr.Bas can change all that.

What the program does

Addressr.Bas can be used to address envelopes, continuous form labels, or 3 x 4 inch pre-printed labels. If the envelopes are pre-printed, it's easy to suppress the printing of your return address, which is built into the program. You can print self-addressed envelopes (or label sets) by merely specifying how many you want.

You type all the addresses before any are printed. You get a second (or third...) chance on every address, in case you make a mistake. While typing, you don't have to worry about centering or alignment spaces, and the program gives you messages on the screen as you work.

Once you have as many addresses in as you want (or 25, whichever is less), you enter a printing phase - the program prompts you for each envelope, and once it is in place and the printer is ready, you press return to print the address. The program then prompts for the next envelope. If you are printing continuous labels, the program will not prompt you, since that would be silly.

When the printing is complete, the program gives you the opportunity to type a line of text to annotate each address. The address and annotation are then printed together on regular paper as a permanent record. The program prompts you to put the paper in.

The program will run under MBASIC on any CP/M based computer, Toshiba, Kaypro, Micromation, converted Apple, and others; and under BASIC on the IBM PC, Compaq, etc. It will not run on an Apple under Applesoft.

It doesn't matter what kind of printer you have. Only one special printer command is included, and it is carefully separated to be visible and changeable. I'll point it out later, with the few other things that you may need to change for your preference or equipment. They're all together in a single line of the program.

The advantage to a program like this is from specialization. Instead of one-to-one alternation between the two principal activities of typing and feeding envelopes, you will type for a while (up to 25 addresses) and then feed envelopes for a while. And you will never have to type your own address, either to or from. The program does not maintain any files, everything happens from the keyboard and to the printer.

What to do with the program

After starting the program, you must first choose envelopes or labels, E or L. Answer in either upper or lower case. Next, you must choose to include or omit the return address, N or return. Omitting the return address on envelopes allows use of pre-printed envelopes or 3 x 4 inch labels. Omitting it on labels keeps the program from printing your return address on every other label.

Hitting S or C will yield self-addressed envelopes or labels, with or without return address. If you choose self-addressed sets, the program will ask how many, and proceed to printing. Otherwise, the program will prompt you to type the addresses.

Type the addresses onto the screen, using return to end each line. You may use either the backspace or delete key to correct errors in a line. End each address by entering an empty line (just press return.) You will be asked to verify the address. If it is not OK, hit N and you get to enter it again. Each address is limited to five lines - the program won't let you enter a sixth, but goes immediately to the

verify step.

To start printing, enter a period followed by a return as the first line of what would be the next address. The program prompts you to put labels or envelopes in the printer (in case you forgot what you asked for), and waits for a return, which is its signal that the printer is ready to go. If printing envelopes or 3 x 4 inch labels, the program prompts after each; if printing form labels, it just continues until all are done.

When all the addresses have been printed, the program will output "Done! Enter R to restart ... OR hit return to annotate addresses." If you just press return, you will be prompted to put regular paper in the printer (top of form, please), and to enter the date (any format is OK). Now the addresses will be put on the screen one at a time, and you can type a comment for each, reminding yourself what it was you sent to that address. The comment and address will be printed together. When each page gets too full, the printer will form feed to the next (if you have to manually insert individual sheets, the program will wait for you each time.)

It may take a few tries to know exactly how to place each envelope or label in the printer. The program is designed so that, generally, the top of the envelope is about one-third inch above the first print line. Form labels are placed level with the first print line.

How to prepare the program for use

Only six lines (290, and 380 through 420) may need changing to suit your needs. If your printer can print Elite (12 pitch), put the control sequence to change pitch in for CHR$(27)+":" in line 290. If your printer cannot, or if you prefer Pica pitch, set Elite="". (EDGE sets the left margin for labels. CENTER sets the spacing for the envelope address.) If your printer takes hand-fed sheets, set TRACTOR=0. Put in your return address between the quotes in lines 380 through 420. My address only requires four lines, as you see, so the last is left empty. Under CP/M you may have to delete the LOCATE instruction and set BS$=CHR$(8). And of course, when you type the program, you don't need to put in any of the comment lines. ■

```
100 ' ADDRESSR.BSC 8-15-83 MARK GARDNER v2.0 10-2-83 v2.1 8-28-86
110 '
120 '"ADDRESSR.BAS" COPR. 1983 MARK GARDNER
130 '
140 'Program to print envelopes with return address (fixed) and
150 'addressee address (input by hand).  Accepts up to 25 addresses
160 'and also provides annotation for users records.
170 '
180 'Version 2.0 adds better user interface, ability to print 1 -
    wide
190 'labels, and support for self addressed envelopes.
200 '
210 'Version 2.1 adds compatibility for IBM PC machines: cursor on by
220 'using LOCATE command, printer controls (ELITE$) to IBM PC
230 'compatible printer, and CHR$(29) for backspace.  Some cleanup
240 'of comments, moved BS$ definition to top, added signon.
250 '
260 'Initialize printer control string and backspace string
270 '**** THIS LINE CONTAINS THE ONLY CODE POSSIBLY REQUIRING CHANGES:

280 '
290  ELITE$ = CHR$(27)+":" : BS$ = CHR$(29) : LOCATE ,,1 : TRACTOR =
     1
300 '
310 'ELITE$ is command string to change the printer to 12 characters
320 'per inch.  BS$ is the character that causes a cursor backspace
330 'on the screen.  The LOCATE turns the cursor on.  Set TRACTOR = 0
340 'if you wish to feed single sheets to printer for the annotation.
350 '
360 '**** PUT YOUR RETURN ADDRESS IN THESE FIVE DATA STATEMENTS:
370 '
```

```
380 DATA "Mark Gardner"
390 DATA "EEmergent Consulting"
400 DATA "Box 3762 GCS"
410 DATA "Glendale, CA 91201"
420 DATA ""
430 '
440 EDGE = 4 : CENTER = 36
450 'Initialize the addresses and return address
460 DIM A$(25,4) : FOR I = 0 TO 4 : READ RETADR$(I) : NEXT I : PRINT
470 'Signon
480 PRINT "ADDRESSR -- envelope/label addressing program, v2.1." :
    PRINT
490 'Select Envelopes or Labels
500 RETFLAG = 1 : PRINT "Choose one: 'E' for envelopes"
510 PRINT "              'L' for labels"
520 PRINT "              'return' to exit the program "; : GOSUB  1710
530 PRINT ANS$ : PRINT : IF ANS$="L" OR ANS$="l" THEN GOTO  560
540 GOTO  590
550 'EXELAB:
560 GOSUB  1630
570 GOTO  630
580 'NEXTIF:
590 IF ANS$<>"E" AND ANS$<>"e" THEN END
600 GOSUB  1670
610 'Give Instructions
620 'GETADR:
630 PRINT "Choose One: 'return' for regular operation "
640 PRINT "            'N' for no return addresses"
650 PRINT "            'S' for self-addressed sets"
660 PRINT "            'C' for combination of 'N' and 'S' ";: GOSUB
    1710
670 PRINT ANS$  : PRINT
680 IF ANS$="N" OR ANS$="n" OR ANS$="C" OR ANS$="c" THEN  RETFLAG = 0
690 IF ANS$="S" OR ANS$="s" OR ANS$="C" OR ANS$="c" THEN GOTO  1540
700 PRINT "       "+WHAT$+" Addressing Program; Enter Addresses --"
710 PRINT "       Terminate each address with one empty line." :
    PRINT
720 'Set up counters for addresses
730 ADDRESS = 0 : ALINE = 0
740 'Input line of address (use INKEY$ to allow commas)
750 'GETLIN:
760 GOSUB  1780
770 A$(ADDRESS,ALINE) = C$ : PRINT
780 'Case line (empty, period, neither)
790 'Condition line is empty
800 IF A$(ADDRESS,ALINE) <> "" THEN GOTO  940
810 'USRCHK:
820 LINES(ADDRESS) = ALINE - 1
830 'If user wishes to re-enter the address
840 PRINT "       Hit return if address ok, 'N' if not";
850 GOSUB  1710
860 IF ANS$="N" OR ANS$="n" THEN GOTO  900
870 'Move to next address
880 ADDRESS = ADDRESS + 1
890 'REDOIT:
```

```
900 PRINT CHR$(13)+"     Enter address (#"ADDRESS+1") or a '.' to
    start printing."
910 ALINE = 0 : PRINT :  GOTO  760
920 'End if (user wishes to re-enter address)
930 'CHKPER:
940 'Condition line is period
950 'Move to printing envelopes
960 IF A$(ADDRESS,ALINE)="." THEN GOTO  1060
970 'Condition line is neither empty nor period
980 'Store address line and move to next
990 '(If the five allowed lines are full, pretend end)
1000 ALINE = ALINE + 1 : IF ALINE = 5 THEN PRINT : GOTO  820
1010 GOTO  760
1020 'End Case line
1030 '
1040 'For all stored addresses
1050 'ENVELS:
1060 MAXADR = ADDRESS - 1 : ADDRESS = 0
1070 'Instruct to insert envelope
1080 'NXTENV:
1090 PRINT "Insert "+WHAT$+"in printer, hit return" : INPUT DUMMY
1100 'Print return address
1110 LPRINT ELITE$;
1120 'NXTLAB:
1130 'Put in correct spacing after return address
1140 IF RETFLAG=1 THEN FOR I = 0 TO 4 : LPRINT SPC(MARGIN1)
     RETADR$(I) : NEXT I
1150 IF RETFLAG=1 AND LABELS=0 THEN LPRINT
1160 IF RETFLAG=0 AND LABELS=0 THEN FOR I = 1 TO 7 : LPRINT : NEXT I
1170 IF RETFLAG=1 THEN LPRINT
1180 'Print address
1190 FOR I = 0 TO LINES(ADDRESS) : LPRINT SPC(MARGIN2) A$(ADDRESS,I) :
      NEXT I
1200 'End for all addresses
1210 '(Eject envelope, or space to next label)
1220 IF LABELS = 1 THEN GOTO  1250
1230 FOR I = 0 TO 3 : LPRINT : NEXT I : GOTO  1280
1240 'FLUSHL:
1250 EXTRA = 5 - ((LINES(ADDRESS)+1) MOD 6)
1260 FOR I = 0 TO EXTRA : LPRINT : NEXT I
1270 'NXTEN1:
1280 ADDRESS = ADDRESS + 1 : IF ADDRESS > MAXADR THEN GOTO  1320
1290 IF LABELS=0 THEN GOTO  1090
1300 IF LABELS=1 THEN GOTO  1130
1310 'ITSDON:
1320 PRINT   : PRINT "Done! Enter 'R' to restart ...OR"
1330 PRINT "Hit return to annotate addresses ";: GOSUB  1710
1340 PRINT ANS$ : PRINT : IF ANS$="R" OR ANS$="r" THEN RUN
1350 IF SELFFLAG = 1 THEN PRINT "SAE's -- skipping annotation" : RUN
1360 PRINT "Put paper in printer (top of form) and enter date - " ;:
     GOSUB  1780
1370 PRINT   : LPRINT "        Mail sent -- notes for " C$ : LPRINT :
     LINES = 0
```

```
1380 FOR J = Ø TO MAXADR
1390   FOR I = Ø TO LINES(J) : PRINT SPACE$(10)+A$(J,I) : NEXT I
1400   PRINT "Give annotation to go with the above address:" :  GOSUB
        1780
1410   LPRINT : LPRINT SPACE$(10)+C$ : LINES = LINES + 2 : PRINT :
       PRINT
1420   FOR I = Ø TO LINES(J) : LPRINT SPACE$(20)+A$(J,I)
1430     LINES = LINES + 1 : NEXT I
1440   IF LINES < 55 THEN GOTO  1480
1450   LPRINT CHR$(12) : LPRINT : LPRINT : LINES = Ø
1460   IF TRACTOR = Ø THEN INPUT "Put in new page, hit return.";
       DUMMY
1470   'NEXTJ1:
1480 NEXT J
1490 'END OF PROGRAM
1500 RUN
1510 '
1520 '*SLFONL, in-line code to setup for multiple self-address prints
1530 'SLFONL:
1540 INPUT "How many sets do you want"; NUMSLF
1550 IF NUMSLF > 25 THEN NUMSLF = 25
1560 FOR J = Ø TO NUMSLF-1
1570   FOR I = Ø TO 4 : A$(J,I) = RETADR$(I) : NEXT I
1580   LINES(J) = 4 : NEXT J
1590 MAXADR = NUMSLF-1 : SELFFLAG = 1 : GOTO  1090
1600 '
1610 '*LABSET, routine to set up for label printing
1620 'LABSET:
1630 LABELS = 1 : WHAT$ = "LABELS " : MARGIN1 = EDGE : MARGIN2 = EDGE
     : RETURN
1640 '
1650 '*ENVSET, routine to set up for envelope printing
1660 'ENVSET:
1670 LABELS = Ø : WHAT$ = "ENVELOPE " : MARGIN1 = 1 : MARGIN2 =
     CENTER : RETURN
1680 '
1690 '*GETANS, routine to return single keystroke from keyboard
1700 'GETANS:
1710 ANS$ = INKEY$ : IF ANS$ = "" THEN GOTO  1710
1720 IF ANS$=CHR$(13) THEN ANS$ = ""
1730 RETURN
1740 '
1750 '*GETSTR, routine to get a string from the keyboard
1760 '
1770 'GETSTR:
1780 C$ = ""
1790 'GETCHR:
1800 B$ = INKEY$ : IF B$ = CHR$(13) THEN RETURN
1810 IF B$<>CHR$(8) AND B$<>CHR$(127) THEN GOTO  1850
1820 PRINT BS$+" "+BS$;
1830 IF LEN(C$)>Ø THEN C$=LEFT$(C$,LEN(C$)-1) : GOTO  1800
1840 'TAKCHR:
1850 PRINT B$; : C$ = C$ + B$ : GOTO  1800
```

Diff.Bas

A file comparison utility

Staff Project When you have more than one version of a program it's always confusing as to which one does what. This handy utility can be used to show the differences and print a neat, formatted output.

Several issues ago we ruminated somewhat over what it would take to make a file compare program. Several of you sent in programs that already exist to do that. Not wanting to re-publish previously printed material, we decided to attack the problem on our own. Diff.Bas is the result, and we have found it to be a rather handy utility.

The problem, simply stated, is: How can you compare two similar programs to see what the difference between them is? Since we want to compare both line numbers and content of the lines it was obvious at the start that a completely renumbered program probably wouldn't work. The idea is to pick up additions, deletions and changes to an existing program; the kind you would make in putting the final touches to a program before it is finished or trying to make it work on more than one machine.

Our plan of attack was to read the line numbers of both programs each into a separate array. Then pad the shorter of the two with zeros, and set the total number of lines equal to the longest program. Then, by stepping through both of these arrays at the same time (each with its own counter) we could read in a line from each of the two programs and compare them.

To illustrate the procedure, we have used a sample program called ROLL1.Bas and then made some changes to it and called it ROLL2.Bas (see figures 3 and 4, respectively.) Figure 1 shows the two arrays, A(I) and B(J), which are created when comparing the two ROLL programs. The ROLL program, by the way, is an actual working program. See the sidebar for what it does. Here, we are only interested in finding what differences, if any, ROLL2 has compared to ROLL1.

Program Details

The program starts at line 130, where machines with BASIC prior to version 5.0 should remove the remark to clear some string space in memory. In line 140 we dimension the two arrays, A() and B() at 600. This means that we can compare programs with as many as 600 lines in them. If you have longer programs, these can be changed to accommodate them. Make sure though, that both arrays are dimensioned to the length of the longest program you wish to compare.

Following the heading lines we ask the user to input the filenames of the two programs to compare. These become F1$ and F2$. We then ask if line printer output is desired, and if so, set variable LP to one and print a heading line on the printer to show what programs are being compared.

The next section of code from lines 290 to 400 reads the original program (F1$) and creates the A(I) array which holds that program's line numbers. It does it by first opening the file for input in line 300 and then reading in a line at a time (checking all the while for the end of file in line 320) and stripping off the line number. The strip takes place in line 340, where we find the space after the line number and make variable S equal to that space position. Then in line 350 we make variable LN equal to the value of the left end of each line up to the space, which says that LN is now equal to the integer value of the line number. Line 360 then puts that number into the A(I) array at the proper I loop count. When we have gone through the original program line 380 closes the file and line 390 sets variable N1 equal to the loop count. In other words, variable N1 will tell us *how many* lines there were in the original program. Now, because we are going to use variables LN and S again for the revised program array, we clear them to zero in line 400.

Lines 420 to 520 do to the revised program what we just did to the original program. Notice that we are using buffer 1 again. We can do this because the two files were not opened *at the same time* and we closed the file after reading in the original program. The code here is almost identical to the

previous section. This time, however, we are filling the B(I) array with line numbers and setting N2 to hold the number of lines in the revised program.

Using our sample ROLL programs as the original and revised program files, the two arrays should now look like figure 1. You never see them like that when you run this program; we temporarily inserted some code in the program to make them appear so we could all look at them. Just like woodworking or metal machining (and many other things), the actual *doing* is nothing compared to the setup required. It usually works that way in programming, too. Everything we have done so far is setup. Now we must do some more. In computing, getting into and out of an operation is always more tricky than what happens during the operation. Most of the time, upon entering an operation, the first thing you need to do is figure out how you are going to get out later. With that little bit of philosophical preparation, let's get into the meat of the comparison code starting at line 550.

In line 550 we set N equal to the number of lines in the largest of the two programs. *Either* of the two could have more lines than the other, and in one case the extra lines would represent deleted lines and in the other case they would represent added lines.

Having done this, we now open both files for input at the same time (in lines 560 and 570.) Now we do a little more initialization. Line 580 sets both the I and J counters to zero. Variable I is going to count the A(I) array and variable J is going to count the B(J) array. They are going to act like a loop without using For..Next. The reason we don't use a For...Next here is that the two counts are going to be synchronized some of the time, and then advanced or retarded at other times, depending on how the line numbers in the two arrays match up to each other. The loop we are talking about here is between lines 590 and 650. Referring again to our little philosophical discussion earlier, lines 580 and 590 were "setup" to get into the loop. Now in the very next line we decide on what condition we will get out of the loop. That line is 600, and it says that if either the I or J count gets to the maximum lines set in variable N, earlier, we must be done and to go to line 660 where we summarize and quit. (This sort of reminds one of the gunslingers of the Old West, who always sat with their backs to the wall facing the door.)

During the comparison of the line numbers we have four cases to consider. Our example case will cover three of them, and if you will refer to figure 1 again, we will look at them one at a time as they come up.

The first line we will come to is line 650 because A(I) is larger than B(J). This says that the revised program has a line number that does not exist in the original program, so we input that line, print "Added" and print the line. Then we increment the J count by one (leaving the I count where it was), increment the added line count (A) by one, go to the subroutine at 770 to see if we need to print the line on the line printer and then go back to line 600 to see if we are done yet.

We are not done yet, and the next comparison of A(I) and B(J) will be equal (both 100), so we get to line 620. If they were not equal, we would be going to line 640 or 650 to see what they really were, but since they are equal we input line 100 from each of the two programs and make them A$ and B$. The next line, 630, will compare the actual lines (not just the line numbers) and if they are the same we go to 590 where the I and J counts are incremented and we go on. If they are different, line 630 will print "Changed" and the line in the revised

Figure 1

	A(I)	B(J)
1	1Ø0	9Ø
2	11Ø	1Ø0
3	12Ø	11Ø
4	13Ø	12Ø
5	14Ø	13Ø
6	15Ø	14Ø
7	16Ø	15Ø
8	17Ø	16Ø
9	18Ø	17Ø
1Ø	19Ø	18Ø
11	2Ø0	19Ø
12	21Ø	2Ø0
13	22Ø	21Ø
14	23Ø	22Ø
15	24Ø	23Ø
16	25Ø	24Ø
17	26Ø	25Ø
18	27Ø	26Ø
19	28Ø	28Ø
2Ø	29Ø	29Ø
21	3Ø0	3Ø0
22	Ø	31Ø
23	Ø	Ø

program that is different from that same line number in the original program. Then the changed counter (C) is incremented by one, we check if we need to print the line on the line printer and then go back to line 590 where the counts are both incremented by one and we go on to the next set of line numbers. Our printout, see figure 2, should by now say that line 90 was an added line and that line 100 was changed. It will then say that lines 220 through 250 are changed too, because we added indents in the revised program. As the I and J counts step down through the two arrays they will find corresponding numbers (even if they are offset one way or the other.) When the count gets to 270 in the I array there is no corresponding line in the J array for it. This will get us to line 640 because I will have line 270 in it and J will have line 280 already. Since A(I) is less than B(J), line 640 will cause "Deleted" to be printed, we will then advance the I count (leaving the J count where it is), increment the deleted count (D) by one, check for line printing and go back to 600 to see if we are done yet. When printing the deleted line we only print the line number, not what was in the line.

What about the other case - the one we haven't mentioned yet? It happens in line 610, where if A(I) is zero we input from the other file and use the "Added" terminology. We have to make a special case out of this condition, otherwise we will get an "Input past end" error. It occurs when we have added lines to the end of the revised file that weren't there in the original.

When I or J finally get to equal N we are done and go to 660 where we print the number of lines in both programs and then give a summary of the lines that were added, changed or deleted. The line printing subroutines at the end seem a little lumpy as far as clean coding goes, but the alternative was to stick them into lines 610 through 650, which would have made an even messier bunch of code there. One day, perhaps on a rainy Saturday, we'd like to take that whole section of logic and try to simplify it. Then again, it works as is, and may be just as well left alone.

Changes for some machines

For machines with BASIC prior to version 5.0 change line 390 to: 390 N1=I-1 and also change line 520 to: 520 N2=I-1. Also remove the remark at the start of line 130. Machines that will need these changes are the Tandy I and III, as well as some others. The program was designed with GW BASIC under MS-DOS. ■

What the Sample Program does

The sample program we chose to illustrate how Diff.Bas works is a little working program all by itself. It's nothing fancy, but does a calculation that will tell approximately what the length of any material is that is rolled up in a roll. This could be tar-paper, printing paper, carpet or whatever. The program assumes that the material is rolled up on some sort of core.

As an example: This magazine is printed on a Web (newspaper) press that uses very large rolls of paper. The rolls are about 34 inches wide, 41 inches in diameter and have a core about 4 inches in diameter. The paper has a thickness of about .005 inches. Two, three or four of these rolls (webs) are fed into the printing press at the same time. The paper is printed simultaneously on both sides. In the case of CodeWorks, eight pages are printed on one side while eight more are printed on the other. The press has the capability of printing on what they call a "half-web" of eight pages (four on one side and four on the other.) So a typical issue of CodeWorks is printed in one pass through the press on two and one-half webs, making 40 pages. The glue binding is applied as the paper passes through the last section of the press, after which the three webs are folded together. After it comes off the press all that is left to do is to run the copies through a three-knife trimmer to open the top, bottom and the right side. We then put a label on it and put it into the mail to you. Sometimes, the center pages of the issue are a little cocked because of the stretch in the paper. The whole operation is fast. The press can run up to 40,000 copies per hour once it gets going. As we said in the article on Diff.Bas, it's all set up — again.

If you want to check out this program, use 41 inches as the diameter of the roll, use 4 inches as the diameter of the core and .005 inches as the thickness of the paper. That represents a roll of paper for the press. You will be amazed to find that there are 261,597 inches of paper in such a roll or 21,799 feet. Actually, that all comes out to just over four *miles* of paper per roll!

You can then use this program (both versions of it) to check out Diff.Bas. If you do, you should get what is shown in the sample run in figure 2. Note that we routed all our output to the printer for the sample run. ■

```
run
---------------------- The CodeWorks ----------------------
          F I L E   C O M P A R E   P R O G R A M
   Compares ASCII program files and prints the differences
-----------------------------------------------------------

Name of original file? roll1.bas
Name of revised file ? roll2.bas
Do you want line printer output too (Y/N)
Processing...
Added---> 90 REM * ROLL2.BAS *
Changed-> 100 CLS
Changed-> 220    SU=SU+C
Changed-> 230    IF I=LA THEN 280
Changed-> 240    D1=D1+2*T
Changed-> 250    I=I+1
Deleted-> 270
Changed-> 290 PRINT"is: ";SU;" inches"
Changed-> 300 PRINT" or ";SU/12;" feet."
Added---> 310 END

There are  21  lines in the original program (roll1.bas)
There are  22  lines in the revised program (roll2.bas)

 2  Lines were ADDED in the revised program.
 7  Lines were CHANGED in the revised program.
 1  Lines in the original were DELETED in the revised program.

Done processing.
Ok
```

Figure 2

```
100 CLS:REM * ROLL1.BAS *
110 PRINT"This program will tell the length of material"
120 PRINT"which is rolled up on a core.
130 INPUT"What is the outside diameter in inches ";D2
140 INPUT"What is the core diameter in inches ";D1
150 INPUT"What is the thickness in inches ";T
160 PI=3.14159
170 LA=((D2-D1)/2)/T
180 LA=INT(LA)
190 I=1:SU=0
200 D1=D1+2*T
210 C=PI*D1
220 SU=SU+C
230 IF I=LA THEN 270
240 D1=D1+2*T
250 I=I+1
260 GOTO 210
270 PRINT
280 PRINT"The length of the material on the roll"
290 PRINT"is: ";SU;" inches or ";SU/12;" feet."
300 END
```

Figure 3

```
90 REM * ROLL2.BAS *
100 CLS
110 PRINT"This program will tell the length of material"
120 PRINT"which is rolled up on a core.
130 INPUT"What is the outside diameter in inches ";D2
140 INPUT"What is the core diameter in inches ";D1
150 INPUT"What is the thickness in inches ";T
160 PI=3.14159
170 LA=((D2-D1)/2)/T
180 LA=INT(LA)
190 I=1:SU=0
200 D1=D1+2*T
210 C=PI*D1
220   SU=SU+C
230   IF I=LA THEN 280
240   D1=D1+2*T
250   I=I+1
260 GOTO 210
280 PRINT"The length of the material on the roll"
290 PRINT"is: ";SU;" inches"
300 PRINT" or ";SU/12;" feet."
310 END
```

Figure 4

```
100 REM * DIFF.BAS * FOR CODEWORKS - 3838 SOUTH WARNER ST.
110 REM * TACOMA WA 98409 VOICE 206-475-2219 MODEM 206-475-2356
120 REM * (c) 1987 80-NW Pub Inc placed in public domain 1987
130 'CLEAR 5000 'Use only if your BASIC needs to clear string space.
140 DIM A(600),B(600)
150 CLS 'Clear screen command. Change to suit your machine.
160 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
170 PRINT"          F I L E   C O M P A R E   P R O G R A M
180 PRINT"   Compares ASCII program files and prints the differences"
190 PRINT STRING$(60,45)
200 PRINT
210 INPUT"Name of original file";F1$
220 INPUT"Name of revised file ";F2$
230 PRINT"Do you want line printer output too (Y/N)"
240 LP$=INKEY$:IF LP$="" THEN 240
250 IF LP$="Y" OR LP$="y" THEN LP=1 ELSE LP=0
260 IF LP=1 THEN LPRINT"Changes in ";F2$;" compared to ";F1$:
    LPRINT" "
270 PRINT"Processing..."
280 '
290 REM * Read the original file and make the A( ) array.
300 OPEN"I",1,F1$
310 FOR I=1 TO 600
320   IF EOF(1) THEN 380
```

```
330    LINE INPUT #1,A$
340    S=INSTR(A$," ")
350    LN=VAL(LEFT$(A$,S))
360    A(I)=LN
370 NEXT I
380 CLOSE 1
390 N1=I
400 LN=0:S=0
410 '
420 REM * Read the revised file and make the B( ) array.
430 OPEN"I",1,F2$
440 FOR I=1 TO 600
450    IF EOF(1) THEN 510
460    LINE INPUT #1,A$
470    S=INSTR(A$," ")
480    LN=VAL(LEFT$(A$,S))
490    B(I)=LN
500 NEXT I
510 CLOSE 1
520 N2=I
530 '
540 REM * Now read both files and compare them.
550 IF N1=>N2 THEN N=N1 ELSE N=N2
560 OPEN"I",1,F1$
570 OPEN"I",2,F2$
580 I=0:J=0
590 I=I+1:J=J+1
600 IF I=N OR J=N THEN 660
610 IF A(I)=0 THEN LINE INPUT #2,B$:PRINT"Added---> "+B$:A=A+1:J=J+1:
    GOSUB 770:GOTO 600
620 IF A(I)=B(J) THEN LINE INPUT #1, A$:LINE INPUT #2, B$:ELSE 640
630 IF A$=B$ THEN 590 ELSE PRINT"Changed-> "+B$:C=C+1:GOSUB 790:GOTO
    590
640 IF A(I)<B(J) OR (A(I)>0 AND B(J)=0) THEN LINE INPUT #1,A$:PRINT "D
    eleted->";A(I):I=I+1:D=D+1:GOSUB 810:GOTO 600
650 IF A(I)>B(J) OR (A(I)=0 AND B(J)>0) THEN LINE INPUT #2,B$:PRINT "A
    dded---> "+B$:J=J+1:A=A+1:GOSUB 770:GOTO 600
660 PRINT
670 PRINT"There are ";N1-1;" lines in the original program (";F1$;")"
680 PRINT"There are ";N2-1;" lines in the revised program (";F2$;")"
690 PRINT
700 PRINT A" Lines were ADDED in the revised program.
710 PRINT C" Lines were CHANGED in the revised program.
720 PRINT D" Lines in the original were DELETED in the revised
    program.
730 CLOSE
740 PRINT:PRINT"Done processing."
750 END
760 REM * Line print subroutines *
770 IF LP=0 THEN RETURN
780 LPRINT "Added--->"+B$:RETURN
790 IF LP=0 THEN RETURN
800 LPRINT "Changed->"+B$:RETURN
810 IF LP=0 THEN RETURN
820 LPRINT "Deleted->";A(I):RETURN
```

Poker Update

catching the cheat and Tandy IV changes

Our Poker.Bas program from Issue 8 apparently decided on its own that the uneven distribution of wealth was a good thing. With the help of Mr. Melanson, we have tracked the cheat to his lair. Also included are the changes for the Tandy Model IV version of Poker with seven players, Jacks or better, and visible card suits.

Poker Update
The case of the Cheating Poker Player

The poker program we published in Nov/Dec 1986 apparently has taken up cheating. "Detective" Arthur Melanson, of Audubon, New Jersey, volunteered for the assignment of tracking the cheat down. What follows is a condensed summary of his findings and our attempts to collar the culprit.

Alas! The Poker program cheats. I have suspected it for some time, but yesterday I caught it red-handed. I would have shot it on the spot except for two considerations: It was cheating in my favor and I really didn't want to put up with the terrible mess that results from an executed computer. I first noticed its unscrupulous ways when it would occasionally bet $30 or $35 for itself, while I was never permitted to wager more than $25. Because it got away with this transgression, it has taken further liberties - it awards the pot to whomever it feels like. On our last session, I was player 3 and I was losing. Most of my $1000 stake was gone. Then the showdown for the hands came, and the program (clearly it wished to prolong my agony) gave me the pot. I shudder to think of what new evil deeds it has planned...

Our effort to corner the cheat was the following: In line 3550, change the *28 to *42 and the *2 to *28. That gives more umph to the actual pairs and less to the kicker card and still lets the kicker determine the winner when the two pair in each hand are identical. *(Note that this fix didn't deter the cheat at all, it found ways of getting around it and we had to fix the fix later.)* As to the other problem of betting $30 or $35 from time to time, eliminate line 3860 and the problem will go away.

Thanks for your help with my cheating program. I have made the changes you suggested (increasing multipliers in line 3550) and all appears to be well. However, the pernicious attitude of the program has broken out in another area. It still cheats! This time the area of difficulty is with one-pair hands. I caught the program awarding the pot to a pair of Jacks with King and Queen kickers when it should have gone to the pair of Jacks with a nine and an Ace. The high pair determination code in lines 3260-3380 looks more complicated than any other determination (I suppose it would be with three odd cards to tally and the two-way Ace.) I'm not sure in my own mind if simply increasing the multiplier will do the job here. Please advise...

It turns out that the pair (no matter what pair) must be given a higher value than an Ace, King, Queen combination value. Then, in case of tie pairs, the next highest card must be given a higher value also. The merge lines (see figure 1) show what needs to be changed. The change to line 3300 was not necessary, but made to keep the code consistent - it didn't hurt to go to 5 here but there was no need to.

Since my last letter the same exact hands that I described to you in the faulty determination of pairs have appeared again. The program is not giving a true mix on the shuffle. What is being received is a variety of "belts" or series of cards. Once identifying a particular belt, or series, then every card to every player is predictable. I hope I don't weary you nor appear to be over critical. Poker is such a great game, and it is my wish to see it operate to the best of its potential...

Your problem with the random function can be taken care of quite easily. Simply add line 955 RANDOM and that will re-seed the generator every time a random number for the shuffle is required. The Tandy III, we found out, seeds the random generator from the system clock. The way the program is written, it only was seeded once per session. With this change it will be re-seeded every time a hand is dealt. (Note this change is applicable only to Tandy models I, III and IV.)

The egregious behavior of Poker.Bas continues. This time, the difficulty has appeared in an area where you have already taken remedial action - the determination of two pair. The new multipliers were installed in line 3550 but it still awarded the pot to the wrong player. Don't hold me to this - but I think I heard a faint snicker from somewhere inside the computer...

(The final fix for this is in line 3550 in figure 1.) Here is the fix for the two pair problem. Change line 3550 so that instead of multiplying by 42 and 28 we raise the high pair to the third power and square the other pair and then add the odd card.

The section of code from 3480 to 3590 works like this: For each of the players, in turn, see if he has folded (line 3500) and if so, go on to the next player. Then see if his hand is two pair, and if not, go on to the next player. If it is two pair, then find the position of the odd card (line 3530). With two pair, sorted the way we sort them, you know that position four will be one of the high pair and position two will be one of the low pair - always. So in line 3550 we can assign values to these two pair. Just in case there are two players with identical two pair hands, we add the odd card value in lines 3560, 3570 and 3580, depending on what value variable S was from line 3530. We found the odd card position in line 3530 by checking adjacent pairs of cards and adding the loop count to S only when the adjacent pairs were equal. Variable S can then only be a 4, 5, or 6, which will pick the appropriate line between 3560 to 3580 to go to.

Before Mr.Melanson received this last communication, he wrote to tell us the following:

Poker.Bas has now given the pot to the low *man in the showdown. This time there was a definite snide snicker as it stole the pot which was rightfully mine. Please rush the fix before I lose control and shotgun the program!*

By now Mr. Melanson should have received what we think is the final fix. We hope it stiffles that insidious snicker. The net result of the fixes is shown in figure 1. These are for all versions of the program, so you can type them in, save them in ASCII and merge them into your present program listing.

Case closed? ●

Changes to all versions of Poker.Bas
first published in Issue 8

Figure 1

```
3300    FOR J=1 TO 4
3330    IF S=1 THEN M(I,11)=(FNM(M(I,1))+33)^3+FNM(M(I,5))^2+FNM(M(I,
        4))+FNM(M(I,3))
3340    IF S=2 THEN M(I,11)=(FNM(M(I,2))+33)^3+FNM(M(I,5))^2+FNM(M(I,
        4))+FNM(M(I,1))
3350    IF S=3 THEN M(I,11)=(FNM(M(I,3))+33)^3+FNM(M(I,5))^2+FNM(M(I,
        2))+FNM(M(I,1))
3360    IF S=4 THEN M(I,11)=(FNM(M(I,4))+33)^3+FNM(M(I,3))^2+FNM(M(I,
        2))+FNM(M(I,1))
3370 S=0
3550    M(I,11)=(FNM(M(I,4)))^3+(FNM(M(I,2)))^2
3860 '
```

Poker for the Tandy Model IV

Back in November 1986 when we first published Poker we assumed that Model IV owners would use the Model III version. We assumed wrong. Because of the numerous requests for a Model IV version with seven players and visible card suits we are including it here. The following listing of changes and additions are to be applied to the original MS-DOS version published in the Nov/Dec 86 issue, starting on page 15. All of the changes mentioned earlier in this article are incorporated in these change lines as well as making the suits visible. The complete Model IV version (Poker7.TRS) will be available on the download, and of course, on the Model IV diskette at the end of the year.

We had quite a time trying to get the card suits to come up on the Model IV. To get them we had to select the special character set using PRINT CHR$(21) in line 140. But it turns out that this is a toggle, so we put in another PRINT CHR$(21) at line 5065. Now, if you enter and exit the program normally, you will always get the suit characters. But if you break somewhere in the program you may get nothing or maybe some Japanese characters. If this happens, you can PRINT CHR$(21) from command mode and then run again to get back into sync.

Also note that user responses must all be in upper case, something we overlooked in the original version and have not corrected.

```
Changed->100 REM * POKER7/TRS ** 5 CARD 7 PLAYER POKER FOR TANDY IV *
Changed->140 PRINT CHR$(21)' SELECT SPECIAL CHAR SET FOR SUITS
Changed->170 DIM B(74),M(7,13),B$(52)
Changed->190 '
Changed->230 PRINT"          J A C K P O T   P O K E R   P R O G R A M"

Changed->270 PRINT"You may watch 7 players or be one. Choose Ø to
             watch, 1 to"
Changed->280 PRINT"7 to sit in. The game ends when any player goes
             broke."
Changed->290 PRINT"There are no wild cards, a 3-card draw limit,
             jacks or better"
Changed->300 PRINT"to open. Chips are $5, $1Ø and $25. Bet in
             multiples of $5"
Changed->350 INPUT"Which position do you want to play: Ø = none or 1-
             7";U
Changed->360 IF U<Ø OR U>7 THEN 35Ø
Changed->410 FOR I=1 TO 7:M(I,1Ø)=DO:NEXT I
Changed->420 CLS:GOTO 46Ø
Changed->450 FOR T=1 TO 1ØØØ:NEXT T      'delay loop
Changed->460 '
Changed->470 FOR I=1 TO 7:IF M(I,1Ø)=<Ø THEN 5ØØØ
Changed->500 PRINT@122Ø,;:PRINT"Ante $1Ø          " '9 spaces
Changed->510 IF A=>8 THEN A=1
Changed->520 FOR I=1 TO 14 STEP 2:PRINT@I*8Ø,"PLAYER";I/2+.5:NEXT I
Changed->540 FOR I=2 TO 14 STEP 2:PRINT@I*8Ø,M(X,1Ø);
Changed->660 IF A1=>8 THEN A1=1:A=A+1:GOTO 43Ø          'IF NO ONE
             OPENED
Changed->860 FOR I=1 TO 7:M(I,7)=Ø:M(I,9)=Ø:M(I,11)=Ø:M(I,13)=Ø:NEXT
             I
Added--->955 RANDOM
Changed->98Ø   R=RND(C)
Changed->1Ø4Ø FOR I=1 TO 7:M(I,6)=1:M(I,12)=Ø:M(I,13)=Ø:NEXT I:DS=53
Changed->1Ø8Ø FOR I=1 TO 7
Changed->116Ø REM ** EVERYBODY ANTE UP 1Ø BUCKS **
Changed->117Ø FOR I=1 TO 7:POT=POT+1Ø:M(I,1Ø)=M(I,1Ø)-1Ø:NEXT I
```

```
Changed->1220 FOR I=1 TO 7
Changed->1360 FOR I=1 TO 7
Changed->1390 FOR I=1 TO 7
Changed->1420       IF FNM(M(I,J))=FNM(M(I,K))THEN V=V+1:M(I,
                    13)=FNM(M(I,K))
Changed->1460   IF V=<3 THEN M(I,8)=V ELSE IF V=4 THEN M(I,8)=14:M(I,
                13)=14 ELSE IF V=6 THEN M(I,8)=16:M(I,13)=14
Changed->1470   IF F1=0 AND S=6 THEN M(I,8)=4
Changed->1480   IF S=10 THEN M(I,8)=0:M(I,12)=1:M(I,13)=14
Changed->1530 FOR I=1 TO 7
Changed->1610   IF N(5)-N(1)=4 THEN M(I,8)=10:M(I,13)=14 '5 CARD
                STRAIGHT
Changed->1620   IF N(4)-N(1)=3 AND N(5)=14 AND N(1)=2 THEN M(I,8)=10:
                M(I,13)=14:AL=1:'ACE LO STRGT
Changed->1730 FOR I=1 TO 7
Changed->1850     IF FNI(M(I,J))<>S THEN B(DS)=M(I,J):M(I,J)=B(DK):
                  DK=DK+1:DS=DS+1:M(I,7)=1
Changed->1920 FOR I=1 TO 7
Changed->1980   IF S=1 THEN B(DS)=M(I,3):DS=DS+1:M(I,3)=B(DK):DK=DK+1:
                B(DS)=M(I,4):DS=DS+1:M(I,4)=B(DK):DK=DK+1:B(DS)=M(I,
                5):DS=DS+1:M(I,5)=B(DK):DK=DK+1:M(I,7)=3:GOTO 2070
Changed->1990 IF S=2 THEN B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:
              B(DS)=M(I,4):DS=DS+1:M(I,4)=B(DK):DK=DK+1:B(DS)=M(I,5):
              DS=DS+1:M(I,5)=B(DK):DK=DK+1:M(I,7)=3:GOTO 2070
Changed->2000 IF S=3 THEN B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:
              B(DS)=M(I,2):DS=DS+1:M(I,2)=B(DK):DK=DK+1:B(DS)=M(I,5):
              DS=DS+1:M(I,5)=B(DK):DK=DK+1:M(I,7)=3:GOTO 2070
Changed->2010 IF S=4 AND FNM(M(I,5))>12 AND U<>0 AND RND(3)=2 THEN
              M(I,7)=0:BL=1:GOTO 2070
Changed->2020 IF S=4 THEN B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:
              B(DS)=M(I,2):DS=DS+1:M(I,2)=B(DK):DK=DK+1:B(DS)=M(I,3):
              DS=DS+1:M(I,3)=B(DK):DK=DK+1:M(I,7)=3:GOTO 2070
Changed->2040 R=RND(3):IF R<>2 THEN 2060
Changed->2050 B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:B(DS)=M(I,2):
              DS=DS+1:M(I,2)=B(DK):DK=DK+1:M(I,7)=2:GOTO 2070
Changed->2060 B(DS)=M(I,1):DS=DS+1:M(I,1)=B(DK):DK=DK+1:B(DS)=M(I,2):
              DS=DS+1:M(I,2)=B(DK):DK=DK+1:B(DS)=M(I,3):DS=DS+1:M(I,
              3)=B(DK):DK=DK+1:M(I,7)=3
Changed->2110 FOR I=1 TO 7
Changed->2190 FOR I=1 TO 7
Changed->2350 FOR I=1 TO 7
Changed->2510 FOR I=1 TO 7
Changed->2520   PRINT@X*80+10,;
Changed->2620 FOR I=1 TO 7
Changed->2630   GOSUB 4450:GOSUB 2830:PRINT C$;
Changed->2660 FOR I=1 TO 7
Changed->2700 FOR I=1 TO 7
Changed->2730 FOR I=1 TO 7
Changed->2780 PRINT@1140,;STRING$(40,32):'CLEAR POT HAS LINE
Changed->2970 IF I<>U THEN D$="  "+CHR$(143):B$(J)=D$+" ":RETURN
Changed->3060 '
Changed->3070 IF FNI(M(I,J))=1 THEN SU=192:GOTO 3110
Changed->3080 IF FNI(M(I,J))=2 THEN SU=193:GOTO 3110
Changed->3090 IF FNI(M(I,J))=3 THEN SU=194:GOTO 3110
Changed->3100 IF FNI(M(I,J))=4 THEN SU=195
```

```
Changed->3110 B$(J)=D$+CHR$(SU)
Changed->3170 FOR I=1 TO 7
Changed->3270 FOR I=1 TO 7
Changed->3410 FOR I=1 TO 7
Changed->3480 FOR I=1 TO 7
Changed->3620 FOR I=1 TO 7
Changed->3700 I=A-1:A1=1:IF I=<0 THEN I=7
Changed->3710 GOSUB 4450:PRINT@X*80+Y-36,;"Dealer";
Changed->3720 I=I+1:IF I=>8 THEN I=1
Changed->3730 IF M(I,8)=0 OR M(I,8)=4 OR M(I,8)=5 OR M(I,8)=6 OR (M(I,
              8)=1 AND M(I,13)<11) THEN GOSUB 4450:PRINT"Passes";:
              GOTO 3760
Changed->3750 IF M(I,8)>1 OR M(I,13)>10 THEN BO=I:GOSUB 4450:
              PRINT@X*80+Y-36,;"Opener";:RETURN
Changed->3770 I=I+1:IF I=>8 THEN I=1
Changed->3780 A1=A1+1:IF A1=>8 THEN RETURN
Changed->3830 BT=RND(15)
Changed->3840 IF BT<>10 AND BT<>15 AND BT<>20 THEN 3830
Changed->3850 IF M(I,8)=>10 THEN BT=BT+5
Changed->3880 GOSUB 4450:IF BT=0 AND FB=1 THEN PRINT"Checks  "; ELSE
              PRINT"Bets ";BT;
Changed->3900 BO=BO+1:IF BO=>8 THEN BO=1
Changed->3970 R=RND(15)
Changed->3990 R=RND(25)
Changed->4040 I=I+1:IF I=>8 THEN I=1
Changed->4060 IF FL=>6 THEN RETURN
Changed->4130 IF HC=28 AND M(I,7)=0 AND BL=1 AND M(I,8)=1 THEN B1=1:
              R1=R1+1:GOTO 4320
Changed->4170 IF M(I,8)=0 AND BT>14 AND RND(5)=<2 THEN B1=3:GOTO 4360
Changed->4240 IF M(I,8)=3 AND RND(5)<>2 THEN B1=1:R1=R1+1:GOTO 4320
Changed->4250 IF R1<6 AND M(I,8)>2 THEN B1=1:R1=R1+1:GOTO 4320
Changed->4260 IF R1<4 AND M(I,8)=2 AND RND(4)<3 THEN B1=1:R1=R1+1:
              GOTO 4320
Changed->4270 IF M(I,8)=0 OR (M(I,8)=1 AND M(I,13)<11) THEN B1=3:GOTO
              4360
Changed->4280 IF M(I,8)<3 AND R1>5 THEN B1=3:GOTO 4360
Changed->4340 IF B1=1 THEN GOSUB 4450:PRINT R$;R;:M(I,9)=M(I,9)+(BT+R-
              M(I,9)):BT=BT+R:FL=0:GOTO 4040
Changed->4350 IF B1=2 THEN GOSUB 4450:PRINT"CALLS";BT-M(I,9);:M(I,
              9)=M(I,9)+BT-M(I,9):FL=FL+1:GOTO 4040
Changed->4360 IF B1=3 THEN GOSUB 4450:PRINT"folds    ";:M(I,6)=0:
              FL=FL+1:GOTO 4040
Changed->4400 IF M(I,6)<>0 THEN GOSUB 4450:PRINT"draws ";M(I,7);
Changed->4420 I=I+1:IF I=>8 THEN I=1
Added--->4502 IF I=6 THEN X=11:Y=36
Added--->4504 IF I=7 THEN X=13:Y=36
Changed->4510 PRINT@X*80+Y,;:RETURN
Changed->4520 PRINT@1280,;:RETURN:'STATUS LINE LOCATION
Changed->4530 PRINT@ 1220,;"POT HAS $";POT:RETURN
Changed->4540 FOR I=1 TO 14 STEP 2:PRINT@I*80+9,;STRING$(54,32);:NEXT
              I:RETURN
Changed->4570 PRINT@1440,;"How much do you bet";
Changed->4630 PRINT@1440,;STRING$(40,32);
Changed->4680 PRINT@1440,;"Discard how many (0 to 3)";
Changed->4690 HM$=INKEY$:IF HM$<"0" OR HM$>"3" THEN 4690
```

```
Changed->4720 IF HM=>4 THEN PRINT@1440,;STRING$(40,32):GOTO 4680
Changed->4740   PRINT@1440,;"Which ones (1 to 5, left to right)";
Changed->4750   WO$=INKEY$:IF WO$<"1" OR WO$>"5" THEN 4750
Changed->4780   PRINT@1440,;STRING$(40,32)
Changed->4810 PRINT@1440,;STRING$(40,32)
Changed->4850 PRINT@1440,;"Do you (F)old, (C)all or (R)aise";
Changed->4860 Q$=INKEY$:IF Q$<>"F" AND Q$<>"C" AND Q$<>"R" THEN 4860
Changed->4870 IF Q$="F" THEN B1=3:M(I,6)=0:PRINT@1440,;STRING$(40,32):
              GOTO 4360
Changed->4880 IF Q$="C" THEN B1=2:PRINT@1440,;STRING$(40,32):GOTO
              4350
Changed->4900 PRINT@1440,;"How much do you wish to raise the bet";
Changed->4950 PRINT@1440,;STRING$(50,32)
Changed->5040 FOR I=1 TO 7:PRINT"PLAYER ";I;" ";:PRINT USING "$$#,
              #####";M(I,10)-DO:NEXT I
Added--->5065 PRINT CHR$(21)
Changed->5100 FOR I=1 TO 7
```

Programming Notes

Most Microsoft BASICs have a shorthand way of expressing the logical operators AND and OR. You can use the plus sign (+) for OR and the asterisk (*) for AND. So if you find a program that says: IF (A=2)+(B=3), it simply says that if A equals 2 OR B equals 3. We seem to recall that one of the original BASICs (Palo Alto Tiny BASIC, back in about 1977) used the symbols exclusively and didn't have the words OR or AND.

We all know that to continue printing on the same line in BASIC we use the semicolon. In GW-BASIC, however, if what follows the semicolon is too long for the line, the entire portion after the semicolon is moved down one line, so you have part of a print statement on one line and the rest of it on the following one. Not nice, but here is a way of getting around it. Use the string concatenation symbol (+) instead of the semicolon. So instead of saying: PRINT "Here is the answer";B$ and B$ is too long for the line, you can say: PRINT"Here is the answer"+B$. In case you hadn't noticed, we used this convention in Qkey.Bas in the last issue and again in Diff.Bas in this issue.

MID$ (mid-string) in Microsoft BASIC is rather unique in that it can be both a *statement* or a *function*. A function always executes some internal code and returns a value. If MID$ is used on the left side of the equal sign, as in MID$(A$,5,3)="ABC", then it is a statement. If it is used on the right side of the equal sign, as in T$=MID$(A$,5,3), it is a function and the value it returns is contained in T$. In the first case, MID$ was used to replace three characters in A$, starting at position five, with the literal characters ABC.

If you must deal with Octal or Hexadecimal numbers in BASIC you needn't worry about converting them to decimal first. They can be represented directly, as &H7A for hex 7A or &O37 for Octal 37. The &H and &O tell BASIC you are dealing with hex or Octal numbers and makes whatever conversions that are necessary.

Most new BASICs support integer division. It uses the reverse backslash. In integer division BASIC rounds both operands to integers and truncates the result to an integer. It is much faster than standard division. Integer division is the complement of the MOD function: If you integer divide 10 by 3 you get 3, but if you take 10 MOD 3 you get 1, the remainder.

CodeWorks 1st Year Programs on Diskette

All the programs published during the 1st year of CodeWorks magazine are available on the following 5¼ soft-sector formats:

- ☐ PC/MS-DOS GW-BASIC 40 track DSDD
- ☐ CP/M MBASIC specify format and computer type:
- ☐ TRSDOS Model I 35 track SDSS
- ☐ TRSDOS Model III 1.3 40 track SSDD
- ☐ TRSDOS IV 6.2 40 track SSDD

If your format is not covered by the above, please inquire and we will try to accommodate you.

Only 50¢ per program!

Check "Disk Order" in the order form below and return this page or a photocopy to us. Diskettes will be sent 1st Class postpaid.

Special Subscriber price $20.00

Note: We cannot provide hard-sector formats. Non-subscriber price $30.00

Subscription ORDER FORM

587

NOTE: The entire set of (7) first year issues is still available at $24.95. Please specify "First Year" if you are ordering this set.

Computer type:

Comments:

Please enter my one year subscription to *CodeWorks* at $24.95. I understand that this price includes access to download programs at NO EXTRA charge.

- ☐ New subscription
- ☐ Renewal subscription
- ☐ Check or MO enclosed.
- ☐ Bill me later.
- ☐ Charge to my VISA/MasterCard # _____ Exp. date _____
- ☐ **Diskette Order**

Referred by _____ # _____

Please Print Clearly:

Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to: ***Codeworks***
3838 South Warner St.
Tacoma, WA 98409

Charge card orders may be called in (206) 475-2219 between 9 AM and 4 PM weekdays, Pacific time.

Index Update

Additions to CWINDEX.DAT

Here are the additions to bring CWINDEX.DAT up to date through Issue 10:

NFL86.bas, reference, issue 10, page 3
Qkey.bas, reference, issue 10, page 4
Plist.bas, reference, issue 10, page 4
Misc, diskette wear and care, issue 10, page 4
Math.bas, reference, issue 10, page 5
Misc, filling out pre-printed forms, issue 10, page 5
Misc, compiling a program, issue 10, page 5
Renum.bas, reference, issue 10, page 5
Card.bas, reference, issue 10, page 6
Misc, enable drives on Model IV, issue 10, page 6
Sox, reference, issue 10, page 6
NFLstat.bas, reference, issue 10, page 7
Rcard.bas, reference, issue 10, page 7
Form.bas, reference, issue 10, page 7
Beginning BASIC, correction, issue 10, page 7
Misc, program, random file demo 4, issue 10, page 12
Misc, program, stack demo for random files, issue 10, page 11
Beginning BASIC, 3 graphing programs, issue 10, page 15
Qkey.bas, main program, issue 10, page 18, keyword search program
Notes, find odd/even numbers easy, issue 10, page 27
Notes, null string explained, issue 10, page 27
Notes, nested if...then and errors, issue 10, page 27
Cwindex.dat, paper listing of this index, issue 10, page 28
Point.bas, main program, issue 10, page 30, using pointers
Misc, article, how to use a compiler, issue 10, page 32
Lotto.bas, main program, issue 10, page 34, find lotto winners
Card1.bas, changes to Card.bas to allow file name, issue 10, page 37
Card.bas, reference, issue 10, page 37
Mcard.bas, reference, issue 10, page 37
Puzzler, generating Pi, issue 10, page 40
Misc, program, double fielding buffers, issue 10, page 6

The changes for this issue will appear in the next. The updated index will also be available on the download under the current issue's menu. ■

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
US Postage
PAID
Permit No. 774
Tacoma, WA

CodeWorks

Issue 12 **July/August 1987**

CONTENTS

CodeWorks

Issue 12 **Jul/Aug 1987**

Editor/Publisher
Irv Schmidt
Associate Editor
Terry R. Dettmann
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Advisor
Al Mashburn

Editor's Notes

Computing, as you all have found out, is a very exacting exercise. If you don't do things the program expects you to it bombs. It seems that every time you build in code to trap for some unwanted input it creates yet another stumbling block for the user.

Take this case. A business man we know has a complete, and expensive, AT system. He uses Framework almost exclusively. One day, he decided to copy some files from his hard disk to a floppy to take to a conference. He kept getting the "invalid number of parameters" error, even though he was in the proper directory and was following the syntax correctly. After spending a considerable amount of time trying to figure out what was wrong, he called me over to look at it. I watched as he typed in a file name with a space in it. I told him that you cannot do that, that DOS would not allow a space in the file name. He told me that Framework let him do it all the time. The authors of Framework, apparently to make it easier for users, allow the space. But then they cleverly change it to an underscore so that DOS will not reject it. So, in making things easier, they created a new problem.

On another subject, I recently found a program listing in one of the most respected of computer magazines. Because I like to put myself in your place once in a while (and because the program looked like it would be interesting) I typed the program into my computer from the magazine. Doing this lets me find out what's good and bad about typing programs from a publication, and after all, that's what we are asking you to do most of the time. The program was billed as standard Microsoft BASIC, and it was.

The first thing that greeted me was the small type used in the publication. The next was the glare from the lights on the slick paper. Third, the magazine would not stay open and lay flat. I fixed two of those problems by making a photocopy of the two pages on the copy machine. That didn't help the small type, but at least it made the copy darker and easier to read.

When I ran the program it bombed - just stopped and wouldn't let me out of an input loop. I compared my listing to that in the magazine and, sure enough, I had mistaken an M for an N. Now it went part way through and then stopped and gave me the BASIC ready prompt. But there were sections in the program to load and save data from disk. Why didn't I ever get there? I sat down and took the program apart, line by line. I found a crucial quote missing, which made the following code look like a long variable name. I found two strings that were never defined. One of them may never have needed to be defined, since it was a null string in all cases. The other just hung out there, all by itself, only once in the program, and did what? Neither of these had anything to do with the disk access code I couldn't get to. Then I found a GOSUB in the program to a line number that didn't exist. The article with the program didn't shed any light on these problems either. I finally decided that I didn't want the program *that* badly.

I'm glad I went through that experience. It shows us what not to do in our own listings and articles. But we aren't perfect either. The other day I called up the original Card.Bas program to try some modifications to it and found that there were several lines at the end of the program that were repeats of earlier lines. They didn't hurt anything, but they shouldn't have been there. How they got there I simply can't imagine. Not only that, but they were in all the versions for all the machines on our first year diskette. Lines 2250 through 2360 should not be there! I'm still trying to figure that one out.

Irv

Forum

An Open Forum for Questions & Comments

Thanks for another great issue. The program that appealed to me was Addressr.Bas and I've spent a couple of days trying to translate it to TRSDOS 6 for my 4P. I have some of the modules working but not all of them, and I did find what I think is an error in your listing that has probably already been brought to your attention. Line 800 with GOTO 940 doesn't work since 940 is a REM line. Did you mean 910?

My printer is a Tandy DMP 200. It will accept ordinary #10 envelopes, not gladly, but it will take them, so I wrote the little gem enclosed which takes care of my needs.

I realize that with Tandy's change over to the Big Blue format there aren't going to be too many TRSDOS programs written in the future. So anything you include and any tips for using the programs you give us on Model III and IV will be greatly appreciated...

D. A. Crossley
Brownsville, TX

See the program listing Crossley. Bas for the program. Yes, you are probably right, it looks like line 800 should go to 910. And yes, we will continue to publish programs that will run with little or no modification on the Tandy machines. By the way, some printers will not let you insert envelopes because of the paper-out switch. If this is a problem, slide a thin strip of plastic down to make the switch and tape it into place. It's a kludge, but it works.

```
100 REM * Crossley.bas addressing program
110 CLS
120 PRINT "Envelope Addressing Program
130 PRINT "By Doc Crossley, Brownsville, TX
140 PRINT "Prints return address
150 PRINT "then address is input from the
    keyboard
160 LPRINT "D. A. Crossley"
170 LPRINT "PO Box 3816
180 LPRINT "Brownsville, TX 78520
190 LPRINT:LPRINT:LPRINT:LPRINT:LPRINT
200 LINE INPUT"NAME ";N$
210 LINE INPUT"STREET ";S$
220 LINE INPUT"CITY ";C$
230 LPRINT TAB(45);N$
240 LPRINT TAB(45);S$
250 LPRINT TAB(45);C$
260 END
```

...I just the other day downloaded Issue 9. I gave Qtext.Bas a whirl and liked it. I do have a real good commercial word processor but sometimes it's overkill. Though I haven't used many of the programs, I do hope CodeWorks will be around for awhile. Just between you and me I would pay a little more for it if necessary. (It) is the best thing since sliced bread, and you can quote me on that anytime.

This is my first letter to any magazine though I have considered one many times. Some programs are a little beyond me but the line by line explanations in CodeWorks have been and will continue to be a great teacher. It's the next best thing to having a live-in mentor.

Richard S. Burwell
Gorham, NH

Thanks for the nice comments, and see this issue for some changes to Qtext.Bas which we think will make it a little more useful.

I have had many hours of pleasure reading and using your programs, as well as the knowledge received in your programming notes.

My pet peeve has always been the time involved using PRINT@, so I have stayed away from them, and used the old TAB function instead. Enclosed is a method I now use that is similar to the LOCATE function in the high price computers now on the market...

Raymond J. Jasek
Spooner, WI

Mr. Jasek enclosed a short program works from LOCATE to PRINT@ but not the other way. Part of our reply to Mr. Jasek: "One of the problems with using a defined function is that you cannot put do *type commands in them, only functions. For example, you cannot put a PRINT@ inside a defined function. Which means that the PRINT@ still has to be in the body of the code. We think we have solved the problem by making a four-line set of subroutines, all of them remarked, and then you un-remark the one for your machine (you could also then delete the others.)" Both Typer.Bas and Speed.Bas, in this issue, use this technique.*

Concerning the letter in Issue 11 from Harry E. Guaspari, Rome, NY, on printing to the screen and line printer without having to use two sets of code or replace PRINT with LPRINT. I came across some code in a BASIC program that works with (MS-DOS 3.0). I have not used earlier releases and do not know if it will

work with them. The short program should be self explanatory. (See program listing Select.Bas) I sure enjoy the publication and the mix of useful programs. Keep up the good work.

Howard S. Scelsi
Augusta, GA

Darn! As soon as I saw your letter I remembered that MS-DOS BASIC has those capabilities. We just tried your code with version 2.0 and it works there too. Thanks also to Robert G. McSorley of New Bern, NC for giving us the same information on this.

```
100 REM * select.bas screen or printer by H S
    Scelsi
110 INPUT "<S>creen or <P>rinter or <E>xit ";
    SP$
120 IF SP$="S" THEN OPEN "SCRN:" FOR OUTPUT
    AS #1:DE$="Screen"
130 IF SP$="P" THEN OPEN "LPT1:" FOR OUTPUT
    AS #1:DE$="Printer"
140 IF SP$="E" THEN END
150 PRINT #1,"Selected the ";DE$
160 CLOSE #1
170 GOTO 110
```

Your magazine is a continuing source of pleasure for me, and I recommend it to anyone interested in programming. Your conversational approach to the simple basics of programming is just great.

The routine by Edward M. Roberts (Issue 8) to convert upper/lower case letters is excellent, although it still lacks a little. John Smith III, for example, comes out John Smith Iii - and of course there's John Smith IV to consider. Also, perhaps there's a way to expand the routine to work for street addresses.

The enclosed routine (see listing Names.Bas) is something I wrote a while back when I became annoyed at the way personal names were being entered into a file and I have used it in a couple of programs. It allows a personal "last name first," so that it may be written onto a data file for sorting, etc. Then when the name is brought out of the file for display, it is re-converted to "last name last."

This is probably a grossly inefficient routine, and I would appreciate any improvements you or your readers might suggest.

A. E. Barnett
Keller, TX

Although we have not exhaustively tested your program, we are including it here for those who may be interested.

```
90 REM * Names.Bas Demo for swapping last
   name by A E Barnett
100 LINE INPUT"Name ...";N$
110 GOSUB 160
120 PRINT:PRINT"Last name first is... ";N$
130 GOSUB 260
140 PRINT:PRINT"Last name last is... ";N$
150 PRINT:GOTO 100
160 REM * put last name first
170 JR$="SrJrIIIV VII"
180 GOSUB 320:IF INSTR(N$," ")=0 THEN RETURN
190 FOR I=1 TO 5:R(I)=INSTR((R(I-1)+1),N$,
    " "):IF R(I)=0 THEN I=1:GOTO 210
200 NEXT I
210 FOR I=5 TO 1 STEP-1:IF R(I)=0 THEN 250
220 Y$=MID$(N$,R(I)+1,2)
230 IF INSTR(JR$,Y$)<>0 THEN N$=MID$(N$,R(I-
    1)+1)+" "+MID$(N$,1,R(I-1)):RETURN
240 N$=MID$(N$,R(I)+1)+" "+MID$(N$,1,R(I)-1):
    RETURN
250 NEXT I
260 REM * put last name last
270 GOSUB 320:X=INSTR(N$," "):IF X=0 THEN
    RETURN ELSE X$=MID$(N$,1,X-1)
280 Y=INSTR(X+1,N$," "):IF Y=0 THEN 310
290 Y$=MID$(N$,X+1,2):IF INSTR(JR$,Y$)=0 THEN
    310
300 N$=MID$(N$,Y+1)+" "+MID$(N$,1,Y-1):RETURN
310 N$=MID$(N$,X+1)+" "+X$:RETURN
320 IF RIGHT$(N$,1)=" " THEN N$=MID$(N$,1,
    LEN(N$)-1):GOTO 320 ELSE RETURN
```

I just received the May/June 1987 issue and noted you were having trouble with the special characters vs. the space compression codes for the Model IV version of Poker.Bas.

If you use a POKE 2964,8 in line 140 instead of the PRINT CHR$(21) it will set the program to display the special characters. If you wish to return to the space compression characters (I do not know why anyone would want to return to such a useless function!) POKE 2964,0.

Just a word of caution, this same address is used to store the screen scroll protect for the first eight screen lines in bits 0-3 (decimal 9 to 15). If scroll protect is used, make sure to reset 2964 to 8, not 0, if you wish to retain the special character set.

Larry Meehan
Bremerton, WA

Well, we just tried it on our Model IV and it still doesn't work. We have

included your comments here though, because there may be different versions of the machine that we don't know about. Actually, we would much prefer the POKE instead of the way it is now. We would appreciate any further information on this.

I am new to programming and have a Tandy 1000A. What code would have to be added to the Poker.Bas program to add color for the different suits and also what changes are needed to subtract all bets from each player instead of just the ante? It looks funny to see more money at the end than when you start.

Dwight Shubert
Little Rock, AR

The poker program, as published, does update the cash for each player after each round of play. Doing it after every bet would slow the program down considerably, and as Kenny Rogers says in his song: "Never count your money while sittin' at the table, there'll be time enough for countin' when the playing's done!" As for color, you could insert color commands into the subroutines at 2960 through 3120. We haven't tried it because there are so many different ways (machines) that use different methods. We couldn't publish one method to cover all of them.

...What can I do, how do I do it? Protect disk from illegal or unauthorized use? I can use a code I know, but how do you protect against BREAK and then LISTing? I've run out of ideas...

Curtis Ainsworth
North Platte, NE

Check Issue 7, page 21, the third note. GW BASIC has the SAVE"filename",P option, which lets you save the program in protected mode. It can then be loaded and run, but not listed. Break still works, but when you try to list it gives you an illegal function call error. As the note says, be very careful that you don't save your only copy of the program with this method. You won't be able to go back to edit it yourself. Keep a copy that you can edit on another diskette which is not accessible to others. We know of no way to undo the P option. If there were one it would defeat the purpose of it in any case.

I have just finished trying out Diff.Bas in Issue 11. I found an error in line 640. The GOSUB 810 command should appear before "I=I+1". As printed, the trial program will list line 280 as deleted instead of line 270...

Robert L. Anderson
St. Albans, WV

You are absolutely right. To fix this, swap the positions of the GOSUB and the I=I+1 in line 640. We have already done this on our master diskette and on the download. Thanks for pointing it out to us.

I am having trouble with Graph2.Bas from Issue 11 on my Tandy Model III Tape loaded computer... When I attempt to run the program it displays the printout of values for line 300 and then prints 360 ?SN...

Cdr. Ralph W. Lindahl
Wenatchee, WA

Thank you for pointing out that you were using a tape Model III. We honestly didn't think that had anything to do with it but it has. The tape version BASICs on those machines do not allow the use of the MID$ function on the left side of an equal sign, although their disk versions do. It is ironic that in that same issue, in notes on page 38, we pointed out the versatility of MID$ used on both sides of the equal sign. As of yet, we have been unable to come up with substitute code for that function.

I once had Issue 3 of CodeWorks and a friend.

I loaned my Issue 3 to my friend.

I sought my Issue 3 from my friend.

I lost both my Issue 3 and my friend.

The loss of the friend was not so much, but I really miss my Issue 3. So if you have a spare copy available, would you please send it to me?

A. E. Barnett
Keller, TX

Such devotion!

That's all the space we have for this issue. Have a pleasant summer.

Irv

"The computer screened five million suspects and narrowed it down to two — you and me!"

Random Files

The merges to make Randemo 5 out of Randemo 4

Terry R. Dettmann, Associate Editor. In this installment, Terry adds the Screen.Bas program from last issue, as well as other changes, to make Randemo5 out of Randemo4. All the merge files, as well as a merged Randemo5 will be available on the download.

In the last issue, we presented the program Screen.Bas and gave some idea of how it works. In this issue, we will incorporate those ideas into the Randemo series.

The Merge for RANDEMO4.BAS

In order to create the program RANDEMO5. BAS, the newest version of the Random Files program including screen control and mapping, we need to build four merge files. You could, if you like, build one file with all of the features in and merge it, but I have split it into four files to make it simpler to understand what's going on. If you get them from the download, these files are:

RANDEMO5.MRG
S600.MRG
S5000.MRG
S6000.MRG

Let's start with S600.MRG and see what each file does. The S600.MRG file merges in utility subroutines from the SCREEN.BAS program in lines 600-999. In SCREEN.BAS, these routines were in lines 1000-1999. Since these lines are already in use in RANDEMO4.BAS, we'll assign them to another series of lines. The subroutines are now:

600 - input a single keystroke character
700 - decode the MSDOS arrow key codes
800 - position the cursor on the screen
900 - break a line into fields using colons

Each works exactly as explained for the SCREEN.BAS program in the last issue.

The merge file S5000.MRG is the group of lines we will be adding in line range 5000-5999. This is the same set of routines that we had in lines 3000-3999 in SCREEN.BAS. Once again, we've changed the line range for convenience since RANDEMO4.BAS already used lines in the desired range (3000-3999). Each routine is adapted to the new line range by adjusting GOTO's and GOSUB's within the lines for the new range.

The routines in line range 5000-5999 are now:

5000 - Read the data map file
5100 - Decode a map line
5200 - Define a field
5300 - Map the fields into the data base

There are some important differences here, beyond just changing lines. First, in subroutine 5000, we've changed the file number of the map file

to 3. If you look in RANDEMO4.BAS, you'll find that file numbers 1 and 2 are assigned to files associated with the data file already. We're also going to use the string FM$ for the data file name. As we'll see when we discuss RANDEMO5.MRG, this will be the name of the data base with a .MAP extension.

Subroutine 5300 is new. We didn't use it in SCREEN.BAS because we weren't dealing with the real data file, but here we need to translate between the MAP information we read in and the actual layout of the data file. If you remember, the actual layout is controlled by the FIELD statement in BASIC. Up to this point, we've left it set at a pretty useless, predetermined value. But now, we want it to be as laid out in the MAP file.

Since the MAP file tells us *where* each field is located and *how long* each field is, we can use that to locate the data within the FIELD statement. With all of the MAP and SCREEN data loaded, we simply pass through the data array (XY) and for each field listed, we execute the field statement to set up the corresponding field pointer (FP$). The field statement is formed in this case by actually defining a dummy field, and then the actual one.

The dummy field (X$) is defined to hold down all characters up to the data field we want to define, and then we define the desired field. If the field we want is at character position 5, then X$ is defined to hold down 4 characters (NL-1=5-1=4) and FP$ is then defined for its actual field length (say 10 characters).

```
          1         2         3         4
1234567890123456789012345678901234567890
--->X<--------->
  X$      FP$
```

We reuse the dummy variable each time through the loop to hold down a different area.

But *You can't do that!* Even as I sit here I hear the collective shout saying that. *It's not legal!* Well, if you thought that, you're wrong. Not only is there nothing in Microsoft BASIC which prevents doing this, this is simply a reassignment of a variable (X$). There is nothing in the reference manual which prevents me from having more than 1 FORMAT statement for a random file buffer. In this case, I have one format statement for *each defined field*. But let's not argue reference manuals and such. The ultimate reason for using this procedure is that *it works*.

The file S6000.MRG has the routines that were at lines 2000-2999 in SCREEN.BAS. The routines are now:

6000 - Load the screen display
6050 - Check for field locations
6100 - Display the screen
6150 - Place the cursor on the screen
6200 - Interpret the keystroke
6300 - Move up on the page
6400 - Move down on the page
6500 - Place blank fields on the screen

Each one works pretty much the same as we found in SCREEN.BAS. Once again in the loading (Subroutine 6000) we've changed the file number to 3 since 1 and 2 are already used. We've also changed the file name to FX$ which will be formed as the data base name plus the extension .SCN.

We've also created one special exception to SCREEN.BAS's operation on screen displays. If you look at each place where we place something on the screen (lines 6110, 6155, 6165, and 6510), the X or ROW coordinate on the screen is adjusted by adding 1. This moves the whole screen down one line compared to what it was in SCREEN.BAS. Why bother? This cuts down the number of lines on the screen to 23 for a 24 line screen and leaves a line blank. But we already have use for that blank line.

In RANDEMO4.BAS, each routine (ADD, EDIT, DELETE) begins by clearing the screen and writing a top line which indicates which routine we're in. I wanted to leave that line. So I shifted the screen down one, cut down it's number of lines by one, and we get a better display in the bargain.

The last merge file, RANDEMO5.MRG, has all the rest of the merges necessary to tie in the screen control and mapping operations into the program. Let's go down the lines one at a time and see what they do:

10 - New heading for the program
30 - Dimension for 20 input fields
51-53 - New variable definitions
140-170 - Define and load the map and screen files
1020-1050 - Replace the ADD command with screen editing (Note: 1050 REM gets rid of the old line 1050)
1060 - Puts the yes/no question on the bottom of the screen display
1600-1620 - Clears the Random Data Buffer for an ADD
1650-1690 - Displays the current Random Data

Buffer to the screen in the designated positions
1700-1740 - Basic screen control corresponding to the main loop in SCREEN.BAS
1800-1890 - Enters a single line of data
1900-1930 - Backspaces over a character
1950-1980 - Clears a data field on the screen if it already has something in it
2045 - Places the cursor on the bottom of the screen for the YES/NO question
2220-2310 - Replace the editing operation with screen editing (Note: lines 2270-2310 REM's are necessary to get rid of lines that are no longer needed)
3045 - Places the cursor on the bottom of the screen for the Yes/No question
3220-3240 - Replace the delete operation display with a screen display
3250 - Places the cursor on the bottom of the screen for the ARE YOU SURE question
4040 - Change the print routine to print all defined fields

If you go over the lines carefully, you'll see that we've reproduced the operations we did in SCREEN.BAS with suitable modifications for each of the basic RANDEMO operations (add, edit, and delete). By reusing the defined subroutines to give us similar results in each case, we've created a powerful screen based data editor.

To merge everything together correctly, get everything loaded onto your system using the file names I've specified, then go into BASIC. Once in BASIC, type:

```
LOAD"RANDEMO4.BAS"
MERGE"RANDEMO5.MRG"
MERGE"S600.MRG"
MERGE"S5000.MRG"
MERGE"S6000.MRG"
SAVE"RANDEMO5.BAS"
```

This will build the newest version of the Random Demo program for you from your old one. Have fun with it.

Where to NEXT?

All of the screen editing is pretty nice, but with any data base, the information you get in there is no good unless you can print it out. Right now, we've got pretty good management of the data and pretty flexible operation, but we need to be able to print it out better. Starting next issue, we'll deal with printing for a little while and get some powerful printing operations set up.

We're still not done with data base operations themselves, even though we do concentrate on printing for awhile. We still have to work up ways to sort the data, provide some protection for the operator, do more sophisticated file operations, learn to create multiple files data bases, and much, much more. Stay with us, "You haven't seen nothin' yet!" ■

Merge file S600.MRG

```
600 REM --- input a character
610 C$=INKEY$:IF C$="" THEN 610
615 IF LEN(C$)>1 THEN GOSUB 700
620 RETURN
700 REM --- look for arrows
710 C = ASC(MID$(C$,2,1))
720 IF C=72 THEN C$=UP$ ELSE IF C=77 THEN C$=RT$
730 IF C=80 THEN C$=DN$ ELSE IF C=75 THEN C$=LF$
740 RETURN
800 REM --- GOTO XY ROUTINE
810 LOCATE X,Y:RETURN
900 REM --- break line
910 FOR K=1 TO 10:BL$(K)="":NEXT K
920 JN$=IN$:NB=1
930 K = INSTR(JN$,":"):IF K=0 THEN BL$(NB)=JN$:RETURN
940    BL$(NB) = MID$(JN$,1,K-1)
950    NB = NB + 1
960    JN$ = MID$(JN$,K+1)
970 GOTO 930
```

Merge file S5000.MRG

```
5000 REM --- read data map
5005 OPEN"I",3,FM$
5010 IF EOF(3) THEN 5035
5015    LINE INPUT#3,IN$
5020    GOSUB 900
5025    GOSUB 5100
5030 GOTO 5010
5035 CLOSE#3
5040 RETURN
5100 REM --- decode map line
5110 IF BL$(1)="FIELD" THEN GOSUB 5200:RETURN
5120 RETURN
5200 REM --- define a field
5210 NF = VAL(BL$(2)):FL = VAL(BL$(4)):FP = VAL(BL$(5))
5220 XY(NF,0)=FL:XY(NF,3)=FP
5230 RETURN
5300 REM --- Map Fields
5310 FOR I=1 TO CX
5320    NL = XY(I,3)
5330    FIELD #1, NL-1 AS X$,XY(I,0) AS FP$(I)
5340 NEXT I
5350 RETURN
```

Merge file S6000.MRG

```
6000 REM --- load screen display
6005 OPEN "I",3,FX$
6010 FOR I=1 TO LN:SC$(I)=""
6015    IF EOF(3) THEN 6030
6020       LINE INPUT#3,SC$(I):IN$=SC$(I):GOSUB 6050
6030 NEXT I
6035 CLOSE#3
6040 RETURN
6050 REM --- CHECK FOR FIELDS IN SCREEN LINE
6055 K = 0
6060 K = INSTR(K+1,IN$,"*")
6065 IF K<=0 THEN RETURN
6070 N = VAL(MID$(IN$,K+1)):XY(N,1)=I:XY(N,2)=K
6075 IF N>CX THEN CX = N
6080 GOTO 6060
6100 REM --- display screen
6110 FOR I=1 TO LN-1:X=I+1:Y=1:GOSUB 800
6115    PRINT SC$(I);
6120 NEXT I
6125 RETURN
6150 REM --- PLACE CURSOR
6155 X = XY(CL,1)+1:Y = XY(CL,2) - 2
6160 GOSUB 800:PRINT "  ";
6165 X = XY(CC,1)+1:Y = XY(CC,2) - 2
6170 GOSUB 800:PRINT "=>";
6180 RETURN
6200 REM --- interpret keystroke
6230 IF C$=UP$ OR C$=LF$ THEN GOSUB 6300
6240 IF C$=DN$ OR C$=RT$ THEN GOSUB 6400
6250 RETURN
```

```
6300 REM --- move up on page
6305 CL = CC:CC = CC - 1:IF CC<1 THEN CC = 1
6310 RETURN
6400 REM --- move down on page
6405 CL = CC:CC = CC + 1:IF CC>CX THEN CC = CX
6410 RETURN
6500 REM --- place blank fields on the screen
6510 FOR I=1 TO CX:X=XY(I,1)+1:Y=XY(I,2):GOSUB 800
6515    PRINT STRING$(XY(I,0),".");
6520 NEXT I
6525 RETURN
```

Randemo5.MRG, this and the other three merge files will make RANDEMO5.BAS of your current RANDEMO4.BAS

```
10 REM --- RANDEMO5.BAS - Random Files with Screen Control
30 DIM FP$(20), SC$(24), XY(20,3)
51 CC=1:CX=1:CL=1
52 UP$=CHR$(5):DN$=CHR$(24):RT$=CHR$(4):LF$=CHR$(19)
53 CR$=CHR$(13):BS$=CHR$(8)
140 FM$=FF$+".MAP":FX$=FF$+".SCN"
150 GOSUB 5000: REM Read Map
160 GOSUB 6000: REM Read Screen
170 GOSUB 5300: REM Setup Fielding
1020 GOSUB 1600:REM Clear Buffer
1030 GOSUB 6100:REM Display Screen
1035 GOSUB 6500:REM Clear Data Fields
1040 GOSUB 1700:REM Enter Data
1050 REM
1060 X=24:Y=1:GOSUB 800
1600 REM --- Clear Data Buffer
1610 FOR I=1 TO CX:LSET FP$(I)="":NEXT I
1620 RETURN
1650 REM --- Display Data Fields
1660 FOR I=1 TO CX:X=XY(I,1)+1:Y=XY(I,2):GOSUB 800
1670 PRINT FP$(I);
1680 NEXT I
1690 RETURN
1700 REM --- Enter Data
1710 CC=1:CL=1
1720 GOSUB 6150
1730 GOSUB 1800:IF DONE THEN RETURN
1740 GOTO 1720
1800 REM --- Enter Line of Data
1810 IN$="":DONE=FALSE
1820 GOSUB 600
1830 IF C$=BS$ THEN GOSUB 1900
1831 IF C$=CHR$(3) THEN DONE=TRUE:GOTO 1880
1832 IF C$=UP$ OR C$=RT$ THEN GOSUB 6300:GOTO 1880
1833 IF C$=DN$ OR C$=LF$ THEN GOSUB 6400:GOTO 1880
1834 IF C$=CR$ THEN GOSUB 6400:GOTO 1880
1840 IF C$<" " THEN 1820
1850 IF IN$="" THEN GOSUB 1950
1860 IN$=IN$+C$:PRINT C$;:GOTO 1820
```

```
1880 IF IN$<>"" THEN LSET FP$(CL) = IN$
1890 RETURN
1900 REM --- Backspace over last character
1905 IF LEN(IN$)<1 THEN RETURN
1910 X=XY(CC,1)+1:Y=XY(CC,2)+LEN(IN$)-1
1915 GOSUB 800:PRINT ".";:GOSUB 800
1920 IN$=MID$(IN$,1,LEN(IN$)-1)
1930 RETURN
1950 REM --- Clear Data Field
1960 PRINT STRING$(LEN(FP$(CC)),".");
1970 X=XY(CC,1)+1:Y=XY(CC,2):GOSUB 800
1980 RETURN
2045 X=24:Y=1:gosub 800
2130    FOR I=1 TO CX:IF INSTR(FP$(I),SF$)<>0 THEN FOUND=TRUE:RETURN
2220 GOSUB 6100:REM Display Screen
2230 GOSUB 1650:REM Display Data Fields
2240 GOSUB 1700:REM Enter Data
2250 GOSUB 1200:REM Save Record
2260 RETURN
2270 REM
2280 REM
2290 REM
2300 REM
2310 REM
3045 X=24:Y=1:GOSUB 800
3220 GOSUB 6100:REM Display Screen
3230 GOSUB 1650:REM Display Data Fields
3240 REM
3250 X=24:Y=1:GOSUB 800
4040       FOR I=1 TO CX:PRINT "FIELD(";I;"): ";FP$(I)
```

Yes! to another year.

We have committed to another year of CodeWorks. You can do your part to help the efforts very simply:

1. Renew early. (Next issue ends this year.)

2. *Order our program diskettes. (Last year is* still available, *the current year diskette will be ready about the middle of September.)*

3. Tell your friends about CodeWorks.

We have a great year planned, stay with us!

Beginning Basic

Three easy and different pieces

Computing is rarely an end in itself, and it shouldn't be. It is a tool to be used as a means to an end. Problems in almost any discipline can be solved through programming, which makes it an interesting and challenging field to be in.

In this installment we are going to look at two or three "real life" problems that can be solved easily with a computer but would be tiresome and error prone with a calculator or by hand. We'll demonstrate some "throw-away" programs that can be written on the spur of the moment and then discarded. Of course, we could make a silk purse out of a sow's ear with any of them, by adding elaborate input checking, error checking, and the like, but that wouldn't make the final information any more valuable. In these cases, we just want a quick answer to a gnawing question and then go on about our business.

The first of these "down and dirty" programs came up when someone we know walked into our office with an unsigned lease contract in hand. The way it was written, it hinted that the monthly rental amount would increase by 5 percent per month for three years. He had been under the impression that the increase would be applied once per year for three years, and wanted to know what the difference would be. It didn't take more than a few minutes to whip up the following little program.

```
100 REM * Effect of 5% increase
110 REM * per mo. for 36 mo.
120 P=300
130 FOR I=2 TO 36
140     P=P+(P*.05)
150     PRINT I,P
160 NEXT I
```

When he found that the monthly lease payments would grow from $300 per month to over $1600 at the end of the lease period, he quickly insisted the lease be re-written in less ambiguous terms. End of problem.

The next program (see Pages.Bas listing) was brought to us by a local person who was preparing a newsletter for press. It was to be printed on 11 x 17 inch parent sheets of paper and folded into 8 x 11 form. Since the sheets he had to work with were 11 x 17, the question was: Which pages went together on one side of the 11 x 17 stock? In printing terms, the question would be which pages were "married."

To see this problem, look at this issue of CodeWorks (which you are obviously now reading) and note that if it had been printed that way, page 2 and page 39 would both be on one side of an 11 x 17 inch parent page (page 1 and page 40 would be the other side of that same piece of paper.) Note that if you were to run off a multi-page report like this on a copy machine, you would have the same problem, called "imposition" by the printers.

The question was if we could write a little program to show which pages went together for any size newsletter. We could, and we did. Since each parent sheet of 11 x 17 paper holds four pages, the newsletter must be in multiples of four. Line 110 asks how many pages there are and line 130 adjusts that number up to the next higher multiple of four. It does it by dividing the number of pages by four and checking the remainder. If the remainder is anything but zero, it keeps adding one to the number of pages and dividing again, until it finds zero as a remainder.

Then, in the loop between lines 140 and 180, we divide the total number of pages exactly in half and print them next to each other with a dash between them. Line 150 ends with a semicolon, keeping the cursor at that spot, and in line 160 if I is one, we print the fact that this is the cover and if I reaches one-half the maximum number of pages we print the fact that this is the centerfold of the newsletter. If it is neither of these we must print a null string to get the cursor down to the next position on the screen. In line 170 we check for each group of four pages that have been printed on the screen and insert another print statement to show

the separation between the parent pieces of stock paper.

That's really all it takes, but in the following lines we have added some "frills" to the program. The first of these is the note that odd numbered pages always go right. The next, in line 210, is a reminder of what the total number of parent sheets will be. In line 220, if the total "live" pages we had was a multiple of four, we simply end the program. Otherwise, we tell how many blank pages will be left over, keeping good grammar in mind and printing "page" for one and "pages" for more than one. You can see now why we had to equate A to P in line 120. Since A may get incremented in line 130, we needed to know what the original number of pages was to figure how many blanks there would be, if any.

If you should ever need to make short runs of multiple-page, folded reports on a copy machine, this program could be of use to you. For those of you whose BASIC includes the MOD function, change lines are given after line 260. It simplifies the code somewhat, and the lines (130 and 170) there now do exactly what MOD does.

Our next example is a little more esoteric, but still valuable to the person interested in it. It needs just a little background information to make sense (see the program Pipe.Bas.)

Pipe Organ building seems to be something from the past, but restoration of Pipe Organs is not. Many old Theater Organs from the 20's are being restored around the country by organ fans. In our part of the country some of them are finding their way into Pizza establishments (Pizza & Pipes), for example.

An organ keyboard has 61 keys, from two C's below middle C to three above. A normal "rank" of pipes then, contains 61 pipes, five full octaves plus one more note.

The *length* of a pipe will determine the frequency at which it will sound ("speak," in organ terms.) The longest pipe (two C's below middle C) will normally be about 8 feet long, while the smallest pipe, the third C above middle C, will be very small. (The term "pipsqueak" traces back to these pipes.) The curve created by a rank of pipes is not unlike that of a harp or the curve on a grand piano. The intensity of the sound, or volume, although largely determined by the amount of wind pressure applied, is also determined by the diameter of the pipe.

The problem, and the point of our program, is to find the scale factor for a rank of organ pipes. According to Audsley (see Reference 1), the overall balance of the rank can be moved by calculating which pipe number will have one-half the diameter of the largest pipe. Audsley suggests that the diameter should halve on the 16th to 19th pipe for a pleasing balance of volume. It turns out that the higher frequency sounds have more volume than the lower ones do, and by adjusting which pipe has the half diameter, you can balance the volume of the entire rank. One problem that occurs, however, with some scale factors is that the smallest pipe may be so small as to be impractical to construct.

Given the pipe with the half-diameter, we want our program to find the scale of the rank, and while we are at it, find the diameter and length of each pipe in the rank. To do this, we will give the program a trial scale factor and let it find the proper one empirically. It will calculate the size and diameter of all the pipes using our trial scale factor, then if it is going in the proper direction, it will continue in that direction. If not, it will switch and start looking in the other direction. When it has found the scale, and sizes of the pipes, so that the diameter of the specified pipe is on the numbered pipe we asked for, it will end the run.

It's not as much of a program as you might think. Basically, it contains two loops, the first of which goes through the pipes from number 2 to number 61, and the second to adjust the scale factor and do it all again until the half-diameter falls on the pipe we specified. The first loop is a For...Next from 210 to 270. The second loop, not as obvious, is between 170 and 380.

The first loop calculates the diameter and length of each pipe using the trial scale factor we provided in line 160. As long as the calculated diameter is larger than half, variable X is set to equal the loop counter (which represents the pipe number) in line 230. After the calculated diameter is equal to or less than the half-diameter, X will remain at that pipe number. The same holds true for the diameter in line 240. After the entire rank is calculated, control falls through lines 280 through 340. In lines 350, 360 and 370 a check is made to see if the pipe where the half-diameter occurred is equal to, larger than or less than the pipe number we specified. If it is less than or equal, we add 0.001 to the scale factor and run again. If it is larger, we subtract 0.001 from the scale factor and run again. If it is equal we are there, so we stop.

The scale factor tells us what percent any given pipe is compared to its next larger neighbor. It is important not only for length and diameter, but for the size of its "foot," the size of its mouth and all

the piece parts that make up the "speaking" part of the pipe. Although the program will print out values to several decimal places, in practice rounding to the nearest tenth of an inch would probably be acceptable. Those of you who find the subject interesting may want to check out the references given at the end of this article. In Reference 2, for example, some very ingenious methods are given to make pipes out of craft paper and enamel. Given today's technology, however, fiberglass may be a better substitute for the enamel. We once constructed such a pipe (G above middle C) and were absolutely amazed at how little air pressure it took to make it sound when its mouth was adjusted properly - also at how terrible it sounded and how much more pressure it took when it was not adjusted properly!

These programs have been presented to show how computing fits into just about any area of endeavor. In each of the three cases, the computer reduces tedious calculations to play. We never quit wondering what J. S. Bach would have done, had he had one. ■

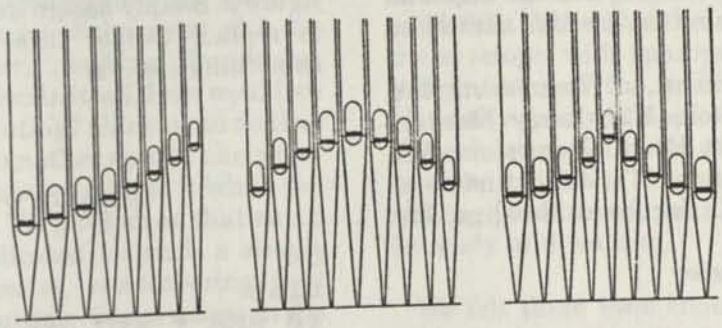
References:

1. Audsley, *The Art of Organ Building*, Vols 1 & 2, Dover Publications, Inc., New York. ISBN 0-486-21315-3 Hardcover.
2. Mark Wicks, *Organ Building for Amateurs*, Ward, Lock & Co., London. Softcover

Both these references and many other worthwhile books on this subject are available from The Organ Literature Foundation, Braintree, MA 02184. Ask for their catalog.

```
100 REM * Pipe.bas * figures scales for Organ Pipes
110 '
120 PRINT"The pipes are numbered from 1, the largest, to
130 PRINT"number 61, the smallest pipe in the rank.
140 PRINT
150 INPUT"You want 1/2 diameter on which pipe number ";Y
160 INPUT"What is your trial scale factor (.96 is a good one)";C
170 A=8 ' Diameter of largest pipe in inches.
180 B=A/2
190 L=96 ' Length of largest pipe in inches.
200 PRINT" 1",A,L
210 FOR I=2 TO 61
220   A=A*C
230   IF A>=B THEN X=I
240   IF A>B THEN D=A
250   L=L*C
260   PRINT I,A,L
270 NEXT I
280 PRINT"Pipe #","Diameter","Length"
290 PRINT
300 PRINT"The closest pipe with a diameter of 1/2 is #";X
310 PRINT"which has a diameter of ";D;" inches.
320 PRINT"The scale factor for the rank is: ";C
330 PRINT
340 FOR J=1 TO 1000:NEXT J ' short time delay
350 IF X<=Y THEN C=C+.001
360 IF X>Y THEN C=C-.001
370 IF X=Y THEN 390
380 GOTO 170
390 PRINT "End of run":END
```

```
100 REM * pages.bas
110 INPUT"How many pages of material do you have ";P
120 A=P
130 AA=INT(A-4*INT(A/4)):IF AA<>0 THEN A=A+1:GOTO 130
140 FOR I=1 TO A/2
150   PRINT I;"-";A+1-I;
160   IF I=1 THEN PRINT "  Cover" ELSE IF I=A/2 THEN PRINT"
    Centerfold" ELSE PRINT" "
170   II=INT(I-2*INT(I/2)):IF II=0 THEN PRINT
180 NEXT I
190 PRINT
200 PRINT"Odd numbered pages are always right-hand pages."
210 PRINT"The number of parent sheets required is: ";A/4
220 IF A-P=0 THEN END
230 IF A-P=1 THEN C$="page." ELSE C$="pages."
240 PRINT"You will have ";A-P;" blank ";C$
250 '
260 REM * lines to change if you have the MOD function.
270 'Line 130 IF A MOD 4<>0 THEN A=A+1:GOTO 130
280 'Line 170 IF I MOD 2=0 THEN PRINT
290 'Isn't MOD a neat function?
```



Puzzler

Still more on Pi and another new one

More on Pi

Tom Witt, of Rochester, NY sent along a couple of pages from *Science News Vol. 131* in reference to Srinivasa Ramanujan, born in 1887 in southern India, who developed an astonishing array of mathematical discoveries. Among these was one that allows one to compute pi to millions of decimal places. It's a very interesting article, too lengthy and complex to go into here, but check it out at your local library if you are interested.

Puzzler 10

We were more than surprised at the number of replies we received to Puzzler 10. Although there was considerable variation in the way you all did it, they all worked. We searched through them all and came up with a composite program (figure 1) which represents what we thought was the most direct and simple way to do it.

Here are those who submitted replies:
Tom Driver, New York, NY said "I don't have a direct use for most of the programs, but I always pick up pointers about programming and coding techniques."

Jerry Bails, of St. Clair Shores, MI said "The problem has two interesting points: 1. remembering what happened during the last loop; and 2. formatting the numerals with commas."

Bob Keegan from Fayetteville, AR said "This wasn't too tough - maybe other BASICs present more problems."

Bill Pottberg from Burlingame, CA said "These puzzles are fun. At first I had an answer that had a space and a comma after every number then I got rid of the space but there was still a comma after every number. Finally, I realized that the way to do it was to put the comma *before* the additional numbers, only."

Others were Larry Abbott, of Wyomissing, PA; E. L. Stanley, Clarkston, WA; Larry Meehan, Bremerton, WA; Robert Hood, Bremerton, WA; Fred Hallden, Huntington, NY; John Anderson, Arlington, MA and J. C. Jacobson, Rawlins, WY.

Puzzler 11

Our next puzzler looks like it could be a bit more difficult (see figure 2). As we mentioned in the last issue in Notes, MID$ can be either a statement or a function. The listing in figure 2 first assigns A$ and then prints it out. MID$ as a function is demonstrated next, in lines 50 and 60. Here, T$ contains that part of A$ starting at the second character and takes the second, third and fourth characters from A$. The original A$ at this point remains intact.

When MID$ can be used as a statement, however, we can change parts of A$. Line 80 replaces the second, third and fourth characters of A$ with "123" which makes A$ a different string than it originally was. You can see this from the sample run below the listing in figure 2. MID$ on the left of the equal sign is a very powerful BASIC statement, as you can see.

Now, the problem: Some BASICs do not allow MID$ on the left of the equal sign. What code could you substitute for line 80 to get the same results? You don't need to repeat the entire program in figure 2. Simply assign an A$ and show your code to replace two or three characters in it with something else. ■

Figure 1

```
10 DATA 220,340,230,340,170,350,
   390,450,410,450,430,450,480,
   450,510,450
20 FOR I=1 TO 8
30   READ A(I),B(I)
40 NEXT I
50 '
60 FOR I=1 TO 8
70   IF B(I)=B(I-1) THEN PRINT","
     ;STR$(A(I));:GOTO 90
80   PRINT:PRINT"Line";B(I);"is
     called from:"STR$(A(I));
90 NEXT I
100 END
```

Figure 2

```
Ok
list
10 REM * MID.bas for puzzler 11
20 A$="ABCDEF"
30 PRINT"The original A$ is ";A$
40 ' MID$ as a function
50 T$=MID$(A$,2,3)
60 PRINT "T$= ";T$
70 ' MID$ as a statement
80 MID$(A$,2,3)="123"
90 PRINT "A$= ";A$
Ok


run
The original A$ is ABCDEF
T$= BCD
A$= A123EF
Ok
```

Qtext1.Bas

An automatic addresser for Qtext

In Issue 9, Jan/Feb 1987, we published Qtext.Bas, a quick and simple text editor. Since we like to think that our programs are at least somewhat useful, we started using it on a daily basis. It turned out to be rather handy, especially when trying to get out a few quick letters.

After typing the letters, however, we still had to either make labels for the envelopes or type the addresses on the envelopes themselves. Heck, we thought, we just typed the name and address for the letter. Why not capture them and print them out automatically? But how could it be done without complicating the whole scheme of Qtext?

A couple of ways suggested themselves. One was to somehow mark the address portion of the letter and spin it off into a file of its own and then accumulate all the addresses for a given writing session. After finishing all the letters, this file could be called up and labels or envelopes printed in one swoop. The idea of creating another file to go with Qtext seemed to be out of character with the original program. Further, "marking" the address portion was a little problem in itself. How would we keep the marks from printing? One way to do that would be to reserve yet another special character and then search for it and eliminate it while the letter was being printed. We gave up on that as too cumbersome and complicated for such a simple program as Qtext. Besides, remembering that special character and putting it somewhere in the address lines seemed like asking for too much operator involvement. We wanted something that would happen "all by itself" if it was desired and not even be an option or get in the way of the original program if it was not. It seemed that no matter how we did it, there would have to be some way of knowing where that address was in the original letter. But we didn't really want to physically "mark" it. That led to the other solution.

This one gets rid of the requirement for having a separate file. It also gets rid of the "marking" problem. After all, the address usually occupies a given space in the letter. Why not use its *position* in the letter to find it and print it out? Now the only operator requirement is that he must always put the address at that location in the letter. Not too unreasonable, considering the alternatives.

In addition, we had to consider that some addresses may be three lines long and some four. We also wanted to be able to print either on label stock or on individually inserted envelopes, and we didn't want to have to move the paper guide on the printer between the two.

Qtext1.Bas is the result of our effort. It does what we want it to. First of all, if you do not want addresses at all, you say so and you are not bothered further. If you want labels for your addresses, you put the stock into your printer before the session, and as you type letters, the addresses (either three line or four line or mixed) will be printed as you type the address in the body of the letter. You won't be able to edit them like you would the rest of the letter, because they print immediately as you type. If you want addresses on envelopes, you say yes to addresses, no to labels and the program asks you to insert the first envelope in the printer. Then as you type the address in the body of the letter, it will also print on the envelope, with appropriate space from the left margin to put the address where it belongs. Naturally, you will need to insert a new envelope for each letter you write, something you don't need to do with labels. Now, when you have finished writing your letters, your labels or envelopes will be ready and waiting.

We felt there were enough changes in making this program to warrant changing Qtext to Qtext1.Bas. The program assumes that the address in the body of the letter will start on the third line and be either three or four lines long and be followed by a blank line. So that you know which line you are on, the new program prints the line number immediately prior to the new line prompt. Obviously, when you start to write, your printer must be on and either labels or an envelope inserted.

We detect the third to sixth lines by the line numbers that Qtext assigns. These appear in line 550 of the program, and if you want to pick out the fourth through seventh line, for example, change the 120 and 160 that are there now to 130 and 170, respectively. If we didn't select to print addresses at all, line 540 forces a jump right around line 550. Otherwise, line 550 will send us to the subroutine at 840.

The subroutine at line 840 decides where to print the address on the printer and also keeps track of line spacing so that three and four line labels can be intermixed. Keep in mind that we don't even get here if we haven't selected to print labels or envelopes and if the program that Qtext is creating is not between its lines 120 and 160. If we selected to print labels, then variable LB, which was initialized to one way up in line 170, will stay at one. This will cause the labels to be printed starting at the left margin of the printer. If we decided to print envelopes, then up in line 410 variable LB was set to 35, which makes the printer move over 35 spaces before starting to print the address on the envelope. You can adjust this number depending on whether or not you are printing on #10 envelopes or the smaller ones. Line 840 says that if the current line we are on in our letter is not blank, to LPRINT a string of blanks equal to LB and then print that current line. Then, still in line 840, we increment another variable, LC (for line count), by one. Variable LC is checked in the next two lines, 850 and 860, along with the contents of the current line we are typing, A$(I), and if the number of address lines we are typing peters out after three, we issue three line feeds to the printer to get to the next label. If we run out of address lines after four of them have been printed on the printer, we only issue two more line feeds to get to the next label. We are assuming that the printer is set to six vertical lines per inch, which is what most are these days. When printing addresses on envelopes all this spacing stuff is unnecessary, since you will be removing one and inserting another in any case. The printer will always start printing at the vertical linespace where you have positioned the labels or the envelope.

From a problem solving and computing standpoint, this program shows how we have traded what we considered an undesirable situation for one not quite so undesirable. We traded the requirement for yet another file and markers for remembering that the address in the body of the letter must go into a certain position. Compromises like this are not uncommon in computing, in fact, many times it's nice to even be able to find a solution that works so easily. We are now using this program to answer most of your letters. It has considerably more convenience than the original Qtext did. We hope you find it so too. ■

```
100 REM * QTEXT1.BAS * A QUICK AND SIMPLE TEXT EDITOR * WRITTEN FOR
110 REM * CODEWORKS MAGAZINE 3838 S. WARNER ST. TACOMA, WA 98409
120 REM * 206 475-2219 VOICE AND 206 475-2356 300/1200 MODEM
130 REM * (c)1986 80-NW Publishing Inc. Placed in Public Domain 1987
140 'CLEAR 5000: ' Use only if your machine needs to clear string
    space
150 DIM A$(1000),Z$(1000)
160 DEFINT I,L,N
170 L=100:IC=10:LA=0:LB=1:LC=0
180 CLS: ' Clear the screen and home the cursor
190 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
200 PRINT"         Q U I C K   T E X T   E D I T O R  Ver 1.1
210 PRINT"    a simple Text Editor using commands already in Basic
220 PRINT STRING$(60,45)
230 PRINT
240 PRINT"  This simple text editor allows you to enter text lines"
250 PRINT"into what becomes a program which can then be edited"
260 PRINT"using the BASIC program editor.
270 PRINT"  You must press RETURN at the end of each input line.
280 PRINT"  Quote marks in the text will be changed to apostrophes.
290 PRINT"  Use a period and the word end  (.END or .end)"
300 PRINT"at the start of a new line to terminate entry of text."
310 PRINT
320 PRINT"This program will append the extension .DOC to your
    filename."
330 INPUT"What will you name this text file";F$
340 INPUT"How many printed lines do you wish per page.";NL
350 INPUT"Do you want labels or addressed envelopes too (y/n) ";AN$
360 IF AN$<>"y" AND AN$<>"Y" THEN 420
```

```
370 PRINT"Your address in the letter must start on the 3rd line,
380 PRINT"and may be either a 3 or 4 line address.
390 LA=1:INPUT"Do you want address labels (y/n) ";AN$
400 IF AN$="y" OR AN$="Y" THEN PRINT"Put label stock in the printer.":
    GOTO 420
410 LB=35:PRINT"Insert an envelope in the printer."
420 F$=F$+".DOC"
430 Z$(0)="10 REM * THE NAME OF THIS FILE IS: "+F$
440 PRINT"The prompt for a new line is the ( symbol."
450 PRINT
460 A$=" LPRINT"+CHR$(34)
470 FOR I=1 TO 1000
480   PRINT I;"(";:LINE INPUT A$(I)
490   IF A$(I)=".END" OR A$(I)=".end" THEN 600
500   FOR Q=1 TO LEN(A$(I))
510     IF MID$(A$(I),Q,1)=CHR$(34) THEN MID$(A$(I),Q,1)=CHR$(39)
520   NEXT Q
530   Z$(I)=STR$(L)+A$+A$(I)
540   IF LA<>1 THEN 560
550   IF L=>120 AND L<160 THEN GOSUB 840
560   IF A$(I)="" THEN Z$(I)=Z$(I)+CHR$(32)+CHR$(34)
570   L=L+IC
580   IF INT(I-NL*INT(I/NL))=0 THEN Z$(I+1)=STR$(L)+
      " LPRINT CHR$(12)":L=L+IC:I=I+1
590 NEXT I
600 N=I-1
610 Z$(N+1)=STR$(L)+" LPRINT CHR$(12)":N=N+1
620 CLS
630 PRINT"You must save the file on disk before it can be"
640 PRINT"recalled and edited or printed."
650 PRINT"Do you want to (S)ave the file on disk or,"
660 PRINT"abort and (R)erun the program."
670 PRINT"   Enter S or R"
680 INPUT"Your choice ";X$
690 IF X$="R" OR X$="r" THEN RUN 100
700 IF X$="S" OR X$="s" THEN 720
710 GOTO 670
720 OPEN"O",1,F$
730 FOR I=0 TO N
740   PRINT #1,Z$(I)
750 NEXT I
760 CLOSE 1
770 PRINT
780 PRINT"You may now load the file like an ordinary BASIC"
790 PRINT"program and edit it using the BASIC editing commands."
800 PRINT"To print the text, simply ready your printer and
810 PRINT"RUN the program called ";F$
820 END
830 REM * Print labels or envelopes subroutine *
840 IF A$(I)<>" " THEN LPRINT STRING$(LB," ");A$(I):LC=LC+1
850 IF LC>3 AND A$(I)=" " THEN LPRINT" ":LPRINT" ":LPRINT" "
860 IF LC>3 AND A$(I)<>" " THEN LPRINT" ":LPRINT" "
870 RETURN
```

Card.Bas Update

Select and print multiple labels

The original version of Card.Bas (Issue 2) allows you to print a set of labels in whatever order you last sorted the file. If you do, you get one of each. But what if you want to just pick out one name and print a label for it, or more than one?

Humberto Nobile, of San Antonio, Texas, and several others have raised this question. It's no problem, really. The lines in figure 1 are all there is to it. These change lines will work with Card.Bas even if you have it configured into the Mini-Card system or if you have made the changes in Card1.Bas from issue 10.

To use it, you simply search for the record you want to print labels for. The new option 5 allows you to print labels for this record. The ON X GOTO line (1390) is changed to include line 2250 as the fifth item. Lines 2245 and on, are added to the end of the program. In these lines, you can configure your label any way you want. For example, between lines 2270 to 2290 you can print items from the record in any order and spacing. You can even print 2, 3, 4 or 5 line labels, but if you do you must remember to change the 3 in line 2300 accordingly. If you print a three-line label, it takes another three line spaces to get to the next label, so if you print a four-line label you only need to add two additional line spaces in line 2300. Line 2320 takes you back to the menu you just came from, so that you can search for another, edit, delete or quit and return to the main menu. You can, of course, print as many of one label as you specify in line 2250.

If you got your version of Card via our first year diskette, you may find lines of code already at 2250 to 2360. This was a mistake (see the Editor's Notes, this issue) and they should be deleted before you make these changes. ■

```
Second name
Second address
Seattle, WA          98188
```

```
Second name
Second address
Seattle, WA          98188
```

Left: Sample showing two of the same label printed.

Below: Lines to incorporate this change into the original Card.Bas program.

```
Changed->1370 PRINT" ENTER 5 - to print labels for this record"
Added--->1372 PRINT" ENTER 6 - to RETURN to main menu"
Changed->1390 ON X GOTO 1150,1090,1410,1550,2250,650
Added--->2245 REM * print more than one label routine *
Added--->2250 INPUT"How many of this label do you want ";HM
Added--->2260 FOR I1=1 TO HM
Added--->2270   LPRINT A$(I)
Added--->2280   LPRINT B$(I)
Added--->2290   LPRINT C$(I);TAB(20);D$(I)
Added--->2300   FOR J=1 TO 3:LPRINT" ":NEXT J
Added--->2310 NEXT I1
Added--->2320 GOTO 1320
```

Amort.Bas

Print amortization tables

Don Winchester, Puyallup, Washington. This program will print amortization tables on the screen or printer. It was developed on a Tandy IV, but changes for MS-DOS and Tandy I, III are included.

```
Car    AMORTIZATION SCHEDULE

Amount financed    $   10,000.00          Interest rate          10.5%
Monthly payment    $      256.03          Total payments           48
Balloon payment    $        0.00          Total loan years          4
                     First payment due  JANUARY     1987

                    ***** Year  1  ****  1987  *****
PAYMENT      MONTH      PRINCIPAL          INTEREST          BALANCE
    1          JAN          168.53             87.50         9,831.47
    2          FEB          170.01             86.03         9,661.46
    3          MAR          171.50             84.54         9,489.96
    4          APR          173.00             83.04         9,316.96
    5          MAY          174.51             81.52         9,142.45
    6          JUN          176.04             80.00         8,966.41
    7          JUL          177.58             78.46         8,788.83
    8          AUG          179.13             76.90         8,609.70
    9          SEP          180.70             75.33         8,429.00
   10          OCT          182.28             73.75         8,246.72
   11          NOV          183.88             72.16         8,062.84
   12          DEC          185.48             70.55         7,877.36
 1987  Total                2,122.64            949.78         7,877.36

                    ***** Year  2  ****  1988  *****
PAYMENT      MONTH      PRINCIPAL          INTEREST          BALANCE
   13          JAN          187.11             68.93         7,690.25
   14          FEB          188.75             67.29         7,501.50
   15          MAR          190.40             65.64         7,311.11
   16          APR          192.06             63.97         7,119.04
   17          MAY          193.74             62.29         6,925.30
   18          JUN          195.44             60.60         6,729.86
   19          JUL          197.15             58.89         6,532.71
   20          AUG          198.87             57.16         6,333.84
   21          SEP          200.61             55.42         6,133.23
   22          OCT          202.37             53.67         5,930.86
   23          NOV          204.14             51.89         5,726.72
   24          DEC          205.93             50.11         5,520.79
 1988  Total                2,356.56            715.85         5,520.79

                    ***** Year  3  ****  1989  *****
PAYMENT      MONTH      PRINCIPAL          INTEREST          BALANCE
   25
```

```
100 REM Amort/tr4  program by Don Winchester *****
110 REM AMOUNTS IN THIS PROGRAM MAY BE OFF A SMALL AMOUNT DUE TO
    ROUNDING TO WHOLE CENTS
120 CLS:CLEAR:REM    RESETS ALL VALUES TO 0 OR NULL  IF THE PROGRAM
    IS RERUN
130 Z$="#,###,###.##":X$="###":REM    PRINTS DOLLAR/CENT VALUES AND
    NUMBER POSSIONS IN PRINT OUTS
140 PRINT STRING$(80,"*");
150 PRINT STRING$(20," ");"A M O R T I Z A T I O N   P R O G R A M"
160 PRINT STRING$(80,"*");
170 PRINT "Enter Name or Item to be Amortized ";:INPUT NM$
180 PRINT "Enter Principal Amount to Amortize ";:INPUT P$
190 IF (VAL(P$)>0) AND (VAL(P$)<10000000#) THEN P=VAL(P$):PP=P:CP=P:
    GOTO 210
200 GOTO 180
210 PRINT "Enter Interest Rate %    ( 12.5 )    ";:INPUT I$
220 IF (VAL(I$)>0) AND (VAL(I$)<99) THEN I=VAL(I$)/100:CI=I:GOTO 250
230 GOTO 210
240 REM CODE TO COMPUTE THE INTEREST FOR PAYMENT AMOUNT
250 IF CI>1 THEN 270
260 CI=CI*100
270 CI=(CI/100)/12
280 PRINT "Enter Number of Years to Amortize  ";:INPUT N$
290 IF (VAL(N$)>0) AND (VAL(N$)<50) THEN N=VAL(N$):YN=N:N=N*12:CN=N:
    GOTO 320
300 GOTO 280
310 REM CODE TO COMPUTE THE PAYMENT AMOUNT
320 R=CI/((1+CI)^CN-1)+CI:R=R*CP
330 PRINT "Your monthly payment is ";USING Z$;R;
340 PRINT "          < ENTER > to keep < C > to change";
350 GOSUB 1180:REM                                    INKEY$ ROUTINE
360 IF A$="C" OR A$="c" THEN 400
370 IF A$="" THEN 340
380 GOTO 440
390 REM   USE ONLY TO CHANGE THE PAYMENT AMOUNT AS REQUIRED
400 PRINT "Enter new monthly payment amount    ";:INPUT R$
410 IF (VAL(R$)>=0) AND (VAL(R$)<1000000#) THEN R=VAL(R$):GOTO 440
420 GOTO 400
430 REM   USE IF THERE IS A BALLOON PAYMENT AT THE END OF THE LOAN
440 PRINT "Enter Balloon payment or < ENTER > ";:INPUT BL$
450 IF (VAL(BL$)>=0) AND (VAL(BL$)<10000000#) THEN B=VAL(BL$):GOTO
    470
460 GOTO 440
470 PRINT "Enter starting month, and year ( 3,1986 ) ";:INPUT M$,Y$
480 IF (VAL(M$)>0) AND (VAL(M$)<13) THEN M=VAL(M$):A=M:YR=VAL(Y$):
    GOSUB 1440:GOTO 500
490 GOTO 470
500 PRINT "To Print a copy of the Amortization Schedule < Y > or < N
    > "
510 GOSUB 1180:REM                                    TO INKEY$ ROUTINE
520 IF A$="Y" OR A$="y" THEN PR$="Y":GOTO 550
530 IF A$="N" OR A$="n" THEN PR$="N":GOTO 550
540 IF NOT (PR$="Y" OR PR$="N") THEN 500
550 PRINT CHR$(15):REM                                TURNS OFF CURSOR
560 GOSUB 1050:REM                                    PRINTS HEADERS ON SCREEN
570 IF PR$<>"Y" THEN 600
```

```
580 GOSUB 1200:REM                                  PRINTS HEADERS ON PRINTER
590 GOSUB 1400:REM                                  PRINTS YEAR HEADING ON
    PRINTER
600 FOR T=1 TO N:REM                                PRINT-OUT LOOP
610 WI=P*(I/12):TI=TI+WI:JI=JI+WI:REM               CODE FOR INTEREST AMOUNTS
620 WP=R-WI:TP=TP+WP:JP=JP+WP:P=P+WI-R:REM          CODE FOR PRINCIPAL AND
    BALANCE AMOUNTS
630 PRINT TAB(4) USING X$;T;:REM                    PRINTS PAYMENT NUMBER
640 REM  CODE FOR MONTH SELECTION
650 MA$="JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC"
660 MM$=MID$(MA$,(A-1)*3+1,3)
670 PRINT TAB(15) MM$;:REM                          PRINTS MONTHS
680 REM  TO PRINT PAYMENT NUMBER AND MONTH ON PRINTER
690 IF PR$="Y" THEN LPRINT TAB(6) USING X$;T;:LPRINT TAB(16);MM$;
700 A=A+1:REM                                       MONTH STEPPER
710 C=WP:PRINT TAB(25) USING Z$;C;:REM              PRINTS PRINCIPAL AMOUNT
720 IF PR$="Y" THEN LPRINT TAB(25) USING Z$;C;
730 D=WI:PRINT TAB(44) USING Z$;D;:REM              PRINTS INTEREST AMOUNT
740 IF PR$="Y" THEN LPRINT TAB(44) USING Z$;D;
750 E=P:PRINT TAB(69) USING Z$;E;:REM               PRINTS BALANCE AMOUNT
760 IF PR$="Y" THEN LPRINT TAB(69) USING Z$;E;
770 REM  CODE TO CONTROL PRINT OUT ON SCREEN TO 1 YEAR AT A TIME
780 L=L+1:IF T=1 THEN L=VAL(M$)
790 IF L<12 THEN 860
800 IF T=N THEN 860
810 REM  PRINTS YEARLY TOTALS, YEAR HEADINGS, RESETS MONTH STEPPER TO
    1 (JAN), AND CLEARS YEARLY TOTALS
820 GOSUB 1280:GOSUB 1360:A=1:JP=0:JI=0:L=0:IF PR$="Y" THEN 850
830 PRINT@1841, "Press any key to continue";:A$=INKEY$
840 IF A$="" THEN 830
850 GOSUB 1050:REM                                   REPRINTS HEADINGS FOR
    THE NEXT YEARLY PRINT-OUT
860 NEXT T:REM                                      PRINT-OUT LOOP
870 GOSUB 1280:REM                                  PRINTS THE YEARLY TOTALS
    FOR LAST YEAR
880 REM  PRINTS TOTALS FOR THE FULL LOAN PIEROD
890 PRINT TAB(1) " Loan  Total";
900 IF PR$="Y" THEN LPRINT TAB(4);" Loan  Total";
910 C=TP:PRINT TAB(25) USING Z$;C;
920 IF PR$="Y" THEN LPRINT TAB(25) USING Z$;C;
930 D=TI:PRINT TAB(44) USING Z$;D;
940 IF PR$="Y" THEN LPRINT TAB(44) USING Z$;D;
950 E=P:PRINT TAB(69) USING Z$;E;
960 IF PR$="Y" THEN LPRINT TAB(69) USING Z$;E;
970 IF PR$="Y" THEN LPRINT
980 REM  RESTART AND AUTO QUIT OPERATION
990 PRINT@1841, "Press < R > to restart < Q > to quit";
1000 GOSUB 1180:IF A$<>"R" AND A$<>"r" AND A$<>"Q" AND A$<>"q" THEN
     1000
1010 PRINT CHR$(14):REM                              TURNS ON CURSOR
1020 IF A$="R" OR A$="r" THEN 100:REM                RESTARTS THE PROGRAM
1030 IF A$="Q" OR A$="q" THEN 1160:REM               ENDS PROGRAM AND RETURNS
     TO SYSTEM
1040 REM  HEADER PRINTING SUBROUTINE
```

```
1050 CLS:PRINT NM$;"  AMORTIZATION SCHEDULE"
1060 PRINT:PRINT "Amount financed   $"USING Z$;PP;:PRINT STRING$(16,"
     ");
1070 PRINT "Interest rate            ";I$;"%"
1080 PRINT "Monthly payment   $"USING Z$;R;:PRINT STRING$(16," ");
1090 PRINT "Number of payments       ";USING X$;N
1100 PRINT "Balloon payment   $"USING Z$;B;:PRINT STRING$(16," ");
1110 Z=Z+1:REM                                  YEAR STEPPER FOR Payment
     year
1120 PRINT "Payment year             ";USING X$;Z
1130 PRINT "                        First payment due     ";MO$;"    ";
     Y$
1140 PRINT " PAYMENT     MONTH          PRINCIPAL              INTEREST
                BALANCE";
1150 RETURN
1160 CLS:END ' ENDS PROGRAM AND RETURNS TO READY PROMPT
1170 REM  INKEY$ SUBROUTINE
1180 A$="":A$=INKEY$:IF A$="" THEN 1180 ELSE PRINT:RETURN
1190 REM  PRINTS HEADERS ON PRINTER SUBROUTINE
1200 LPRINT TAB(5);NM$;"   AMORTIZATION SCHEDULE":LPRINT
1210 LPRINT TAB(5);"Amount financed
     $"USING Z$;P;:LPRINT TAB(52);"Interest rate        ";I$;"%"
1220 LPRINT TAB(5);"Monthly payment
     $"USING Z$;R;:LPRINT TAB(52);"Total payments       "USING X$;N
1230 LPRINT TAB(5);"Balloon payment
     $"USING Z$;B;:LPRINT TAB(52);"Total loan years     "USING X$;YN
1240 LPRINT TAB(25);"First payment due  ";MO$;"  ";Y$
1250 LPRINT
1260 RETURN
1270 REM  PRINTS YEARLY TOTALS ON PRINTER SUBROUTINE
1280 PRINT:PRINT TAB(1)YR;" Total";
1290 IF PR$="Y" THEN LPRINT  TAB(4) YR;" Total";
1300 C=JP:PRINT TAB(25) USING Z$;C;
1310 IF PR$="Y" THEN LPRINT TAB(25) USING Z$;C;
1320 D=JI:PRINT TAB(44) USING Z$;D;:E=P:PRINT TAB(69) USING Z$;E;
1330 IF PR$="Y" THEN LPRINT TAB(44) USING Z$;D;:LPRINT TAB(69) USING
     Z$;E;
1340 YR=YR+1:REM                                YEAR STEPPER
1350 RETURN
1360 IF PR$="Y" THEN  1380
1370 IF PR$<>"Y" THEN 1420
1380 LPRINT:LPRINT
1390 REM  PRINTS YEAR AND HEADER BETWEEN YEARLY PRINT-OUT ON PRINTER
     SUBROUTINE
1400 IF PR$="Y" THEN X=X+1:LPRINT TAB(26)"***** Year ";X;" ****
     ";YR;" *****";
1410 LPRINT TAB(4);"PAYMENT    MONTH        PRINCIPAL
     INTEREST                 BALANCE";
1420 RETURN
1430 REM  PICKS STARTING MONTH TO PRINT IN HEADER SUBROUTINE
1440 MR$="JANUARY  FEBRUARY MARCH    APRIL    MAY      JUNE     JULY
       AUGUST   SEPTEMBEROCTOBER  NOVEMBER DECEMBER "
1450 MO$=MID$(MR$,(M-1)*9+1,9)
1460 RETURN
```

Change lines for Tandy I and III machines.

```
Changed->100 REM Amort/tr3  program by Don Winchester *****
Added--->105 CLEAR 2000
Changed->140 PRINT STRING$(63,"*");
Changed->150 PRINT STRING$(5," ");"A M O R T I Z A T I O N   P R O G
R A M"
Changed->160 PRINT STRING$(63,"*")
Changed->320 R=CI/((1+CI)[CN-1)+CI:R=R*CP
Changed->330 PRINT "Your monthly payment is ";USING Z$;R
Changed->550 '
Changed->630 PRINT USING X$;T;:REM                  PRINTS PAYMENT NUMBER
Changed->670 PRINT TAB(9) MM$;:REM                       PRINTS MONTHS
Changed->710 C=WP:PRINT TAB(16) USING Z$;C;:REM       PRINTS PRINCIPAL
 AMOUNT
Changed->730 D=WI:PRINT TAB(30) USING Z$;D;:REM       PRINTS INTEREST
AMOUNT
Changed->750 E=P:PRINT TAB(43) USING Z$;E:REM         PRINTS BALANCE AM
OUNT
Changed->760 IF PR$="Y" THEN LPRINT TAB(69) USING Z$;E
Changed->830 PRINT:PRINT"Press any key to continue"
Changed->840 A$=INKEY$:IF A$="" THEN 840
Changed->890 PRINT:PRINT TAB(1) " Loan  Total";
Changed->900 IF PR$="Y" THEN LPRINT" ":LPRINT TAB(4);" Loan  Total";
Changed->910 C=TP:PRINT TAB(16) USING Z$;C;
Changed->930 D=TI:PRINT TAB(30) USING Z$;D;
Changed->950 E=P:PRINT TAB(43) USING Z$;E;
Changed->990 PRINT:PRINT"Press < R > to restart < Q > to quit";
Changed->1010 '
Changed->1060 PRINT:PRINT "Amount financed   $"USING Z$;PP;:PRINT STR
ING$(5," ");
Changed->1070 PRINT "Interest rate                ";I$;"%"
Changed->1080 PRINT "Monthly payment   $"USING Z$;R;:PRINT STRING$(5,
" ");
Changed->1100 PRINT "Balloon payment   $"USING Z$;B;:PRINT STRING$(5,
" ");
Changed->1130 PRINT "                      First payment due    ";MO$;"   ";
Y$
Changed->1140 PRINT"PMT #   MONTH          PRINCIPAL     INTEREST
BALANCE"
Changed->1300 C=JP:PRINT TAB(16) USING Z$;C;
Changed->1320 D=JI:PRINT TAB(30) USING Z$;D;:E=P:PRINT TAB(43) USING
Z$;E;
Changed->1400 IF PR$="Y" THEN X=X+1:LPRINT TAB(26)"***** Year ";X;" *
*** ";YR;" *****"
Changed->1410 LPRINT TAB(4);"PAYMENT    MONTH          PRINCIPAL
    INTEREST                  BALANCE"
```

Change lines for GW-BASIC (MS-DOS) machines.

```
Changed->100 REM Amort.Bas MS-DOS version by Don Winchester *****
Changed->830 LOCATE 24,1:PRINT"Press any key to continue";:A$=INKEY$
Changed->990 LOCATE 24,1:PRINT"Press < R > to restart < Q > to quit";
```

Fract.Bas

Make decimals from fractions

Lawrence Keyes, Burlington, Vermont. Here is a program that will take a fraction, a whole number or a combination of the two, or a decimal number and return the value in decimal notation.

While writing my own portfolio system I have reviewed many commercial and public domain portfolio programs. One aspect of these that often annoys me is having to enter stock and option prices in decimal form. Now to be sure, I have done it enough to memorize that seven-eighths is 0.875 but, after all, the computer is supposed to save you mental effort. Do *you* know what 15/16ths is?

Rather than put up with this, I wrote a set of BASIC routines which translate fractions into decimal form. The program FRACT.Bas is the result.

The program will take a fraction, a whole number, a combination of the two or a decimal number and return the value in decimal notation. The only restriction is that whole numbers and fractions must be separated by a space: 12 3/8, and so forth. Since the program accepts input as a string variable, it is possible to have incorrect output if you enter letters instead of numbers. The program does not check for such input errors.

FRACT.Bas was written on an IBM-PC, but should run on any computer. You will want to convert the mnemonic variable names to single letters if your BASIC doesn't allow multi-character variable names.

The most obvious use for these routines is for translating keyboard input. Another possibility is to modify the routines to translate quotations downloaded from an on-line database such as Dow-Jones News Retrieval.

Earlier BASICs can use long variable names, but there are some precautions to take. We found that all the variables in this program worked well on BASICs prior to version 5.1 with the exception of two: TOTAL and FRACTION. Total includes the keyword "TO" and fraction the keyword "ON" which cause syntax errors. Consequently, we changed these two to TTL and FRACT, respectively. Most of the earlier BASICs will look at the first two characters in a variable name as significant, however, the variable name can be longer if you like - so long as it contains no embedded keywords. The program as shown in the listing was checked out on both MS-DOS and on a Tandy Model III. We also had to change the code slightly at line 280, from a WHILE...WEND construct to the code shown there now. (Earlier BASICs don't have WHILE...WEND.)

We feel the program has further merit because it can be reduced to a minimum amount of code and incorporated into other programs as a subroutine where converting from fractions to decimals is desired, which we assume was the original intent of the author.

We found Mr. Keyes used an interesting technique in this program. Note lines 400, 490 and 540. The IF KIND$="DEC" THEN ELSE 540 (in line 490, for example) at first looked a little suspicious, but it works. When the IF clause is true, program flow simply falls through to the next line, otherwise, it takes the ELSE route. It's another way of saying: 490 IF KIND$<>"DEC" THEN 540. Further, it worked as written on both the machines we checked. - Eds. ●

```
100 REM * FRACT.BAS * Lawrence M Keyes * 5 May 86
110 REM * Accepts fractions or decimal numbers as strings
120 REM * in X$ and converts them to decimal output.
130 '
140 X$="":TTL=0:NUM=0:WHOLE=0:DENOM=0
150 '
160 CLS
170 PRINT"Enter a stock or option price using fractions.
180 PRINT"Be sure to separate the whole number from the fraction
190 PRINT"with a space.
200 INPUT X$
210 GOSUB 280 ' Strip leading blanks
220 GOSUB 320 ' Identify
230 GOSUB 380 ' Parse the input
240 PRINT
250 PRINT"The equivalent in decimal notation is: ";TTL
260 END
270 '
280 REM * Subroutine to strip leading blanks *
290 IF LEFT$(X$,1)=" " THEN X$=X$-LEFT$(X$,1):GOTO 290
300 RETURN
310 '
320 REM * Subroutine identify - scans input string to see if it is
330 REM * a whole number, fraction, or whole number and fraction.
340 IF INSTR(X$," ")=>1 THEN KIND$="BOTH":GOTO 360
350 IF INSTR(X$,"/")=>1 THEN KIND$="FRAC" ELSE KIND$="DEC"
360 RETURN
370 '
380 REM * Subroutine Parse Input - bites off a chunk of the input
390 REM * string and determines the values.
400 IF (KIND$="BOTH") THEN ELSE 490
410   WHOLE$=LEFT$(X$,INSTR(X$," ")-1)
420   WHOLE=VAL(WHOLE$)
430   FRACT$=RIGHT$(X$,(LEN(X$)-INSTR(X$," ")))
440   GOSUB 610
450   FRACT=VAL(NUM$)/VAL(DENOM$)
460   TTL=WHOLE+FRACT
470 RETURN
480 '
490 IF (KIND$="DEC") THEN ELSE 540
500   WHOLE=VAL(X$)
510   TTL=WHOLE
520 RETURN
530 '
540 IF (KIND$="FRAC") THEN ELSE 590
550   FRACT$=X$
560   GOSUB 610
570   FRACT=VAL(NUM$)/VAL(DENOM$)
580   TTL=FRACT
590 RETURN
600 '
610 REM * Subroutine xtract fraction - to extract a fraction
620 NUM$=LEFT$(FRACT$,(LEN(FRACT$)-INSTR(FRACT$,"/")))
630 DENOM$=RIGHT$(FRACT$,(LEN(FRACT$)-INSTR(FRACT$,"/")))
640 RETURN
```

Typer.Bas

Put your fancy printer through its paces

Staff Project. We have had a lot of fun with this one, and we think you will too. The program was written with user modification in mind. We had to, since there are so many different printers available these days.

In the last few years printers have become so sophisticated they almost dance a jig for you. And the price isn't bad either, compared to what you got earlier. They all have one thing in common, however, and that is codes, codes and more codes. The manual we got with our Micronics SG-10 is actually larger than that which came with our MS-DOS machine!

It is almost universal that printer control codes start with the escape sequence CHR$(27). What happens after that depends on the particular printer. Some may have as many as four or five character string sequences to make one change in mode of operation. Remembering and setting these control sequences can be a chore. Some applications programs have built-in shorthand codes that may be inserted. For example, WordStar has CTRL-PB for boldface, CTRL-PS for underscore, and many others. From BASIC, you can set the printer to a particular mode and print or LLIST an entire program in that mode. But what do you do when you want just one word in bold, or underscored?

Another problem, which undoubtedly varies from printer to printer, is that some modes are self-cancelling, while others are not. This means you must issue the codes to cancel the mode you are in before you can enter a new mode.

Typer.Bas is a program that will make a simulated electric typewriter out of your computer. That sounds like a step into the past, but it has a few features the old electric typewriter didn't have. First of all, it allows you to input an entire line before it prints it, which means you can take a good look at it before it is committed to paper. Secondly, it allows you to embed simple control codes (of your own choosing) to make mode changes. If you have a Star SG-10 or SG-15 printer you can use the program as written. Otherwise, you will need to customize it to your own printer. The program uses the dollar sign as a control character. It is always followed by something other than a space or a number, so there is no confusion when printing dollar amounts. This is also why we called it the "Dollar" electric typewriter.

Figure 1 shows the screen as it should look when you have the program running and have called for the help feature. Your typing will take place just below the ruler line. You may enter up to two screen lines of text there before pressing ENTER. The line may contain any control codes wherever you want them. They will take effect at the place in the line where you put them. If you will take a temporary look at line 1210 of the program, you will see that that entire line is called by inserting $F1 in your input line, eliminating the need for you to cancel all other modes you may have been in and setting the printer back to standard, dot matrix.

The current mode you are in will show on the screen, as well as the line of type you are on. You may force a page eject ($page) or use $end to quit. In addition, for those of you with 24-line screens, $help will toggle the help screen on and off.

When you have finished typing in your line of text, pressing ENTER will move the entire line to a position above the ruler line on the screen and you may go ahead with the next line below the ruler line while the printer prints the first one. The "Current mode" line near the top of the screen display will always show the last mode you were in. In some cases, where you can be in two modes at once - like letter quality and underline, the sub mode will show directly below the current mode. It will go away when you leave that sub mode. Current line tells you which line on the page you are currently at. If you have a machine that supports on-screen editing, you can use it to make any edits to the line you are on before you ENTER

it. If you do not have on-screen editing, you will simply have to backspace and retype the line.

Although our printer has many more codes, we picked the 15 we thought we would be using the most and incorporated them into the program. To customize the program to your printer, you will need to decide how many codes to include and what their code names will be. Put these in the data statement in line 1080. Put the number of codes first, then the codes themselves. In lines 1100 through 1130, put the plain English words describing the codes you just entered. Their placement in the data statements must correspond to the placement of the codes in line 1080. Then, keeping the same placement in mind, enter your printer commands starting at line 1210. Note that these lines all end with RETURN, because they are subroutines, called from line 750 and further defined by the variable M in the ON M GOTO in line 1200. If you use more than 15 codes, you will also need to change lines 910 and 920. For twenty items, for example, change the step in line 910 to 4, then add the fourth tab position and the F$(J+4) in line 920. The help feature should then print out four columns of five items each. You could get more codes if you use shorter descriptions and put five columns across.

You can make the codes anything that makes sense to you. We used the following in the program:

$F1 - standard dot matrix
$F2 - standard dot matrix italic
$F3 - dot matrix bold
$F4 - dot matrix bold italic
$SL - start letter quality
$PY - proportional spacing on
$EY - expanded mode on
$ST - superscript on
$SB - subscript on
$SU - start underline
$EU - end underline
$SO - cancel super/sub script
$EL - end letter quality
$PN - turn proportional spacing off
$EN - turn expanded mode off

Keep in mind that, depending on your printer, you will need to physically get out of some modes before you can get into another. If this is a problem, you can build the cancellation into the LPRINT lines starting at line 1210. In ours, line 1210 cancels several other modes before getting into standard font, for example.

The Program

The program is reasonably self-documented. The main modules of it are separated by empty remark statements. One of the first things to note is that if you try to run the program without having a printer on line it will hang up. This is because line 180 initializes the printer to its default "just turned on" mode. If the printer is not on, some computers will give a "device not available" error, while others will just sit and do nothing.

The code from line 200 to 260 is our general purpose LOCATE/PRINT@ routine. In this section, you remark all the lines that do not apply (you could even remove them entirely) and leave the one that applies to your machine unremarked. This subroutine was purposely put near the front of the program because BASIC starts looking for such subroutines from the lowest line number on, and since this subroutine is called so frequently, putting it there speeds the operation of the program. Because it is there, however, we need to put a GOTO around it.

The routine will handle the locate and print@ problem for most of the machines we cover in CodeWorks. The one that will probably need refinement is the one for CP/M machines; which use a variety of different methods to locate the cursor. Note also that some machines label the upper left corner of the screen as position 1,1 while others call it position 0,0. This accounts for the -1 in some of the code. Because this program is going to work with a stable (non-scrolling) screen, considerable housekeeping must be done to clear a line, for example, before printing new information there. The variables X and Y seemed to be the most logical variables to reserve for cursor (screen) locating purposes.

The next module, from 280 to 320, reads in the control codes. First, in line 290, we read variable K. This will read the number 15 from data line 1080. Note that this is also the number for the DIM statements in line 150, and tells how many codes there are. The codes themselves are now read in and put into array F$(J).

In the next section we read the description of the codes and put them into array F1$(J). In line 390 we clear the screen and home the cursor, the only time in the program that we do that.

The code from 400 to 500 initializes the screen with headings and a ruler line. Nothing in this

Samples from the article, using Typer.Bas. This is in expanded mode.

(This is in standard dot matrix mode F1) In the last few years printers have become so sophisticated they almost dance a jig for you.

(The same thing in italic F2) In the last few years printers have become so sophisticated they almost dance a jig for you.

(Now in bold, F3) In the last few years printers have become so sophisticated they almost dance a jig for you.

(And now in bold and italic, F4) In the last few years printers have become so sophisticated they almost dance.

This line is standard dot matrix without proportional.
This line is standard dot matrix with proportional.

Typer.Bas will allow you to underline and to print both superscript and subscript.

(An example of letter quality) In the last few years printers have become so sophisticated they almost dance a jig for you.

Typer.Bas is a program that will make a *simulated* electric typewriter *out of your computer.* That sounds like step into the **past,** but it has a few **features** the old electric typewriter *didn't* have.

First of all, it allows you to input an entire line before it prints it, which means you can take a good look at it

Secondly, it allows you to embed simple control codes (of your own choosing) to make mode changes.

Let's see if we can insert an expanded word in the **middle** of a line.
That worked ok.

Now let's see if we can do letter quality and proportional at the same time:
This should be letter quality with proportional spacing on and,
this should be letter quality with proportional spacing off.

Not very well, however, it apparently didn't do it after all.

As you can see, you can mix some modes and not others. Careful study of your printer manual should show which modes supercede others and which do not. You can then custom install your own codes into Typer.Bas.

```
        The CodeWorks - One Dollar Electric Typewriter
        ----------------------------------------------
($end) to quit, ($page) to page eject, ($help) help on/off
Current line: 1          Current mode: $F1-standard

----.----1----.----2----.----3----.----4----.----5----.----6----
```

Typer.Bas is a program that will $F2make$F1 a

A typical input line, shown where it would appear on the input screen.

```
$F1-standard          $F2-standard italic   $F3-bold
$F4-bold italic       $SL-letter quality    $PY-proportional on
$EY-expanded on       $ST-superscript on    $SB-subscript on
$SU-underline on      $EU-underline off     $SO-cancel sup/sub
$EL-end ltr quality   $PN-proportional off  $EN-expanded off
```

Screen display for Typer.Bas

Figure 1

section will change as long as the program runs. Not so for the following section of code. This is where the action takes place. It is a loop which starts at 530 and goes to 650. Line 530 goes to the upper part of the screen and tells which line of the page we are on. Line 540 then places the cursor just below the ruler line and waits for us to input something. Line 560 checks to see if we want to end the program, and if so, it does it - right there. Line 570 checks to see if we want to eject the page in the printer, and if so, sends a LPRINT CHR$(12) to the printer to do it. It then wipes A$ out (clears it to a null because we don't want to print the control), clears what we just typed and places the cursor at the starting position again. Line 580 checks to see if we want the help feature on or off the screen. If we do, it goes to the subroutine at 840, where if the help was on screen, it clears it and if it was not on screen, it puts it there. Variable HF is the help flag, and tells whether or not the help is on screen. Line 590 clears the area just above the ruler line (in case there may have been something there), and line 600 moves the cursor to that position so that the line we just ENTERED will print there.

We now must check the line for embedded codes and print it on the printer, so line 620 sends us to the subroutine at line 690. The first thing we want to do at the subroutine at 690 is to check and see if there are any dollar signs in the line. Line 690 checks for the dollar sign, and if there are none, line 700 says to LPRINT the line and return via the GOTO 800 in line 700. If there is a dollar sign in the line, control falls through to line 710. Tricky stuff happens here.

The line we have just input is called A$. In line 710 we are going to cruise down that line, one character at a time, and if the character is not a dollar sign, we LPRINT it and leave the print head where it is. This happens via the ELSE route in line 730. If C$ happens to be a dollar sign, then lines 740, 750 and 760 come into play. The loop counter here is variable M, and K represents the maximum number of codes we have. This loop will now compare what it just found (a dollar sign and the two characters after it) to all our codes to see if there is a match. If there is no match, it must mean that the dollar sign is real, and probably stands for what dollar signs usually stand for ($1,000.00, or something like that.) However, if a match is found in line 750, we go off to yet another subroutine at 1200, where the loop count in M will tell us which code it was, go to the appropriate line to set up the

printer, and then return. As soon as we return to line 750 from the subroutine at 1200, we go to still another subroutine at 1000 to update the current mode on the screen. Lines 1000 and 1010 handle the current mode update, while lines 1030 and 1040 handle the submode update. Note that we have neatly avoided, so far, printing the actual control code on the printer. When we return to line 750 from the subroutine at 1000, we immediately increment the J count by 2, which jumps us right over the two characters following the dollar sign and keeps them from printing. We then jump to line 780, which is the NEXT J line. We then keep on going down the line (A$) because it's quite possible there may be more than one control code in a given line. When we have finished with printing A$ on the printer (one character at a time) we encounter line 790, which forces a linefeed to the printer to get us to the next line. We then return to where subroutine 690 was called from in the first place, line 630. Line 630 clears our input area, line 640 increments I by one (to keep an accurate line count) and line 650 sends us right back to 530 to input another line of text.

We think you are going to have some fun playing with this one. Let us know how it turned out. ■

Typer.Bas listing, for all machines

```
100 REM * TYPER.BAS * ELECTRIC TYPEWRITER *
110 REM * Written for CodeWorks magazine 3838 S. Warner St. Tacoma WA
120 REM * 98409 (206)475-2219 voice and (206) 475-2356 300/1200 modem
130 REM * This program won't start unless you have your printer on.
140 'CLEAR 2000 ' Use if your machine needs to clear string space.
150 DIM F$(15),F1$(15) ' DIM for the max number of codes you want.
160 '
170 '----- initialize normal printing as a default
180 LPRINT CHR$(27) "B" CHR$(1);:M=1
190 '
200 ' --General purpose locate/print@ routine. Unremark as needed.
210 GOTO 290
220 LOCATE X,Y:RETURN ' GW-BASIC machines
230 'PRINT@((X-1)*64)+(Y-1),;:RETURN ' 64-col print@ machines
240 'PRINT@((X-1),(Y-1)),;:RETURN ' 80-col 24-line, 0-0 print@
    machines
250 'PRINT@(X,Y),;:RETURN ' 80-col 24-line, 1-1 print@ machines
260 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN 'CP/M, adjust
    to suit
270 '
280 '----- read in the control codes
290 READ K
300 FOR J=1 TO K
310   READ K$:F$(J)=K$
320 NEXT J
330 '
340 '----- read in the control names
350 FOR J=1 TO K
360   READ K$:F1$(J)=K$
370 NEXT J
380 '
390 CLS ' clear screen and home cursor, adjust to suit your machine
400 I=1
```

```
410 PRINT TAB(8)" The CodeWorks - One Dollar Electric Typewriter"
420 PRINT TAB(8)"-----------------------------------------------"
430 PRINT"($end) to quit, ($page) to page eject, ($help) help on/off
440 '
450 ' ----- make a ruler reference line
460 X=8:Y=1:GOSUB 220
470 PRINT"----.----1----.----2----.----3----.----4----.----5----.----
    6----"
480 '
490 X=4:Y=1:GOSUB 220:PRINT"Current line: "
500 X=4:Y=25:GOSUB 220:PRINT"Current mode:
    ";:IF I=1 THEN PRINT F$(M)"-"F1$(M)
510 '
520 '----- start of main control loop
530 X=4:Y=14:GOSUB 220:PRINT I ' tells which line you are on.
540   X=10:Y=1:GOSUB 220
550   LINE INPUT A$
560   IF LEFT$(A$,4)="$end" OR LEFT$(A$,4)="$END" THEN CLS:END
570   IF LEFT$(A$,5)="$page" OR LEFT$(A$,5)="$PAGE" THEN LPRINT
      CHR$(12):A$="":X=10:Y=1:GOSUB 220:PRINT STRING$(32,32):X=10:Y=1:
      GOSUB 220:GOTO 550
580   IF LEFT$(A$,3)="$he" OR LEFT$(A$,3)="$HE" THEN GOSUB 840:A$="":
      GOTO 550
590   X=6:Y=1:GOSUB 220:PRINT STRING$(128,32)
600   X=6:Y=1:GOSUB 220
610   PRINT A$
620   GOSUB 690
630   X=10:Y=1:GOSUB 220:PRINT STRING$(128,32)
640   I=I+1
650 GOTO 530
660 END ' ----- of main control loop
670 '
680 '----- subroutine to find and interpret codes
690 T=INSTR(A$,"$")
700 IF T=0 THEN LPRINT A$:GOTO 800 ' there weren't any.
710 FOR J=1 TO LEN(A$)
720   C$=MID$(A$,J,1)
730   IF C$="$" THEN 740 ELSE 770
740   FOR M=1 TO K
750     IF MID$(A$,J,3)=F$(M) THEN GOSUB 1200:GOSUB 1000:J=J+2:GOTO
        780
760   NEXT M
770   LPRINT C$;
780 NEXT J
790 LPRINT" "
800 RETURN
810 END
820 '
830 '----- display help - delete this section and line 580 for 16
    line screens
840 IF HF=0 THEN 900
850 X=18:Y=1:GOSUB 220:PRINT STRING$(160,32)
860 X=20:Y=1:GOSUB 220:PRINT STRING$(160,32)
870 X=22:Y=1:GOSUB 220:PRINT STRING$(80,32)
```

```
880 X=10:Y=1:GOSUB 220:PRINT STRING$(80,32)
890 X=10:Y=1:GOSUB 220:HF=0:RETURN
900 X=18:Y=1:GOSUB 220
910 FOR J=1 TO K STEP 3
920   PRINT F$(J)"-"F1$(J);TAB(22);F$(J+1)"-
      "F1$(J+1);TAB(44);F$(J+2)"-"F1$(J+2)
930 NEXT J
940 X=10:Y=1:GOSUB 220:PRINT STRING$(80,32)
950 X=10:Y=1:GOSUB 220
960 HF=1
970 RETURN
980 '
990 '----- subroutine to update the current mode on screen
1000 IF M=<9 THEN X=4:Y=39:GOSUB 220:PRINT STRING$(20,32)
1010 IF M=<9 THEN X=4:Y=39:GOSUB 220:PRINT F$(M)"-"F1$(M)
1020 X=5:Y=39:GOSUB 220:PRINT STRING$(20,32)
1030 IF M >9 THEN X=5:Y=39:GOSUB 220:PRINT STRING$(20,32)
1040 IF M >9 THEN X=5:Y=39:GOSUB 220:PRINT F1$(M)
1050 RETURN
1060 '
1070 '----- The first data element must indicate how many items
     follow.
1080 DATA 15,$F1,$F2,$F3,$F4,$SL,$PY,$EY,$ST,$SB,$SU,$EU,$SO,$EL,$PN,
     $EN
1090 '----- there must be as many items here as in the data line
     above.
1100 DATA standard,standard italic,bold,bold italic,letter quality
1110 DATA proportional on,expanded on,superscript on,subscript on
1120 DATA underline on,underline off,cancel sup/sub,end ltr quality
1130 DATA proportional off,expanded off
1140 '
1150 '- subroutine to convert codes to printer commands.
1160 '- Here and in the data statements above, you can set up
1170 '- your own codes and adjust for your particular printer.
1180 '- These are for the Star Micronics SG-10 or SG-15 and do
1190 '- do not represent all possible codes for those machines.
1200 ON M GOTO 1210,1220,1230,1240,1250,1260,1270,1280,1290,1300,1310,
     1320,1330,1340,1350
1210 LPRINT CHR$(27) "5";CHR$(27) "F";CHR$(27) "B" CHR$(1);:RETURN
     '$F1
1220 LPRINT CHR$(27) CHR$(52);CHR$(27) CHR$(70);:RETURN '$F2
1230 LPRINT CHR$(27) CHR$(69);CHR$(27) CHR$(53);:RETURN '$F3
1240 LPRINT CHR$(27) CHR$(69);CHR$(27) CHR$(52);:RETURN '$F4
1250 LPRINT CHR$(27) "B" CHR$(4);:RETURN '$SL
1260 LPRINT CHR$(27) "p" CHR$(1);:RETURN '$PY
1270 LPRINT CHR$(27) "W" CHR$(1);:RETURN '$EY
1280 LPRINT CHR$(27) "S" CHR$(0);:RETURN '$ST
1290 LPRINT CHR$(27) "S" CHR$(1);:RETURN '$SB
1300 LPRINT CHR$(27) "-" CHR$(49);:RETURN '$SU
1310 LPRINT CHR$(27) "-" CHR$(48);:RETURN '$EU
1320 LPRINT CHR$(27) "T";:RETURN '$SO
1330 LPRINT CHR$(27) "B" CHR$(5);:RETURN '$EL
1340 LPRINT CHR$(27) "p" CHR$(0);:RETURN '$PN
1350 LPRINT CHR$(27) "W" CHR$(0);:RETURN '$EN
```

Speed.Bas

You see more than you think you do

Staff Project. Here is something different. At first glance, it doesn't seem to do much, but you will be surprised when you try it.

Just how much of what your senses perceive is consciously retained? If you see something for a fraction of a second, say, can you describe it? Under hypnosis, some witnesses of crimes have been able to read off license plate numbers which they couldn't even remember seeing in their normal waking state. It isn't always successful, but numerous cases attest to the fact that it can be done.

Most advertisements involve direct suggestion. Subliminal stimulation, the presentation of stimuli too faint or too brief to be perceived, has been offered as an example of indirect suggestion; for example, although the claim has been disputed, popcorn sales were reported to increase by subliminal suggestion via the motion picture screen. The movie was *Picnic*, with William Holden and Kim Novak, in a theater in New Jersey in 1957.

Most speed reading programs start, or at least include, a session with what is called a *Tachistoscope*. This is a machine, not unlike a slide projector, which flashes a group of numbers on a screen in a darkened room. The machine can control the length of time the numbers are exposed. Exposures of 1/10 second are possible. Those learning to speed read are encouraged to write down the numbers they see. When you first go through this exercise the first thought that comes to you is that you cannot possibly see or retain those numbers. Then, as you relax and throw all caution to the wind and write down whatever comes to mind, you find to your amazement that you probably have most of the numbers right. You find that you do not need to consciously read off each digit in a five-digit group; simply absorbing the entire group is sufficient. In short, as the speed of the machine is increased, you learn to trust your visual sense more and more. Basically, that is what speed reading training is all about.

It is very likely you have already done this type of thing. Have you ever scanned a newspaper and saw your name flash somewhere on the page? It takes you a little while to find it exactly, but your subconscious "eye" saw it immediately.

Based on this idea, we have developed the program, Speed.Bas. It is meant to simulate a Tachistoscope, and allows you to play with this idea at your leisure. The program was developed with GW BASIC under MS-DOS, but we are including change lines for the Tandy I, III and IV machines.

The program picks a five-digit number at random and displays that group for a length of time which you can choose. There are six different speeds you can select, from 2 seconds down to .06 seconds. The number appears near the middle of your blanked screen, and you simply write down on paper what you think you saw. The number is internally stored so that you can check your answers later. The numbers appear at about six second intervals.

One of the problems was trying to get the timing down for a variety of machines. To help make this easier, we set variable TM in line 130 to a number that you can adjust so that all ten groups of five-digit numbers appear in one minute. Then all other times are derived from that number. Don't count on the timing being exact, however, it will be only a rough approximation. This due to the differences in the persistence of the phosphor on various video

screens. Obviously, the larger the number in TM, the slower (and longer) the numbers will appear.

When you are comfortable and can read all the numbers when they appear for two seconds, you can increase the speed and learn to read them faster. Eventually, you should be able to read them correctly at .06 seconds. Actually, it sounds like a hokey deal - doesn't it? You simply must try it to find out.

The Program Description

The program starts out by setting the main timer, TM, in line 130. We mentioned how to set this earlier in the article. Line 170 makes a randomizer for those machines that need it, and gets its random seed from the system clock. The general purpose locate/print@ subroutine follows. Unremark the appropriate line for your machine and leave all others remarked (or don't even type them in.) The program heading comes up next and says essentially what we have already discussed in this article. At the end of the program heading, you are asked to press ENTER and to pick a display time. Depending on the time you pick between lines 440 and 490, the lines from 530 through 580 will divide the main timer and set variable TD (time of display) appropriately.

Now that we have the display time defined in TD, we show you where the numbers will appear on your screen. Next, starting at line 670, we pick five random digits that are between zero and nine. Line 720 concatenates them into A$. In line 740 we accumulate the groups of numbers into the A$(J) array, so that we can display them all later for checking purposes.

The time interval between groups of numbers is set in line 750. In line 760 we print the five-digit group of numbers at the center of the screen, and after the interval set by variable TD (display time), we wipe it off again. We then clear A$ and go to make the next group of digits.

When all ten groups of digits have been displayed you can press ENTER to see what the numbers really were. Try it, you may just be a little surprised. ■

Speed.Bas listing, for GW-BASIC. See page 38 for changes for other machines.

```
100 REM * SPEED.BAS * Written for CodeWorks magazine
110 REM * 3838 S. Warner St. Tacoma WA 98409 (206) 475-2219
120 'CLEAR 2000 'Use only if you need to clear string space.
130 TM=800 'Adjust value here so that all 10 groups of
140 '       numbers appear in one minute.
150 '
160 REM * Make randomizer if needed
170 RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
180 '
190 ' - General purpose locate routines. Unremark one for your
    machine.
200 GOTO 270
210 LOCATE X,Y:RETURN                    ' GW-BASIC
220 'PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy I/III
230 'PRINT@((X-1),(Y-1)),;:RETURN    ' Tandy IV
240 'PRINT@(X,Y),;:RETURN                ' CP/M MBASIC using Print@
250 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN 'CP/M, adjust
    to suit
260 '
270 CLS
280 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
290 PRINT"          T A C H I S T O S C O P E  P R O G R A M
300 PRINT"                learn to trust your visual senses
310 PRINT STRING$(60,45)
```

```
320 PRINT
330 PRINT"You will be shown 10 different number groups of five
    digits.
340 PRINT"Approximately once every six seconds a group will appear at
350 PRINT"the center of your screen. It will stay there for a very
    short
360 PRINT"time, depending on your answer to the next question. Write
    the
370 PRINT"number down on a piece of paper. DO NOT analyze, just write
380 PRINT"down what you think you saw. You'll do better if you relax.
390 PRINT
400 INPUT"Press ENTER to select the display time ";XX
410 '
420 CLS
430 PRINT"Pick the time duration each number group will be on screen."
440 PRINT TAB(10);"1 = .06 second
450 PRINT TAB(10);"2 = .12 second
460 PRINT TAB(10);"3 = .25 second
470 PRINT TAB(10);"4 = .5  second
480 PRINT TAB(10);"5 =  1  second
490 PRINT TAB(10);"6 =  2  seconds
500 INPUT "The number of your choice ";XX
510 IF XX<1 THEN 420
520 ON XX GOTO 530,540,550,560,570,580,420
530 TD=INT(TM)/512:GOTO 610
540 TD=INT(TM)/128:GOTO 610
550 TD=INT(TM)/32:GOTO 610
560 TD=INT(TM)/8:GOTO 610
570 TD=INT(TM)/2:GOTO 610
580 TD=INT(TM)
590
600 REM * Show where numbers will appear on screen.
610 X=12:Y=34:GOSUB 210
620 PRINT "0 0 0 0 0"
630 X=13:Y=25:GOSUB 210
640 PRINT"Your number will appear here"
650 FOR I=1 TO 2000:NEXT I
660 '
670 REM * Pick the numbers, display them and then blank them.
680 CLS
690 FOR J=1 TO 10
700   FOR I=1 TO 5
710     A=INT(RND(1)*10)
720     A$=A$+STR$(A)
730   NEXT I
740   A$(J)=A$
750   FOR T=1 TO TM*5:NEXT T 'about a 6 second interval
760   X=12:Y=34:GOSUB 210:PRINT A$
770   FOR T=1 TO TD:NEXT T 'the display time set by TM
780   X=12:Y=34:GOSUB 210:PRINT STRING$(20,32)
790   A$=""
800 NEXT J
```

```
810 '
820 REM * Find out what the numbers really were.
830 FOR I=1 TO 1000:NEXT I
840 INPUT"Press ENTER to check your numbers";XX
850 PRINT
860 FOR J=1 TO 10
870    PRINT A$(J)
880 NEXT J
890 END 'of program
```

Change lines for Tandy I and III machines.

```
Changed->100 REM * SPEED/TR3 * Tandy III version Written for CodeWork
s
Changed->120 CLEAR 2000 'Use only if you need to clear string space.
Changed->130 TM=250 'Adjust value here so that all 10 groups of
Changed->170 'RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->210 'LOCATE X,Y:RETURN                      ' GW-BASIC
Changed->220 PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy I/III
Changed->710      A=RND(10)-1
```

Change lines for Tandy IV.

```
Changed->100 REM * SPEED/TR4 * Tandy 4 version Written for CodeWorks
Changed->130 TM=600 'Adjust value here so that all 10 groups of
Changed->170 'RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->210 'LOCATE X,Y:RETURN                      ' GW-BASIC
Changed->230 PRINT@((X-1),(Y-1)),;:RETURN     ' Tandy IV
Changed->710      A=RND(10)-1
```

Programming Notes

One of our readers called to say that Keycode.Bas in the last issue would not quit running when he hit the BREAK key, and he had to re-boot his computer to get it to stop. The question was why? I couldn't think of any logical reason at the time, but driving home from work that evening it finally dawned on me. He was using MS-DOS, in which you can turn BREAK on and off. Apparently some application program he had been using prior to running Keycode.Bas had turned it off. From MS-DOS Ready, you can type BREAK, and it will tell you that it is on or off. Then you can change it to whatever you like by typing BREAK ON or BREAK OFF.

While we were checking on this one in the manuals, we also came across VERIFY. Again, you can type VERIFY and it will tell you that it is on or off. And you can change it just like you can BREAK. With verify on all disk writes will be verified (read back and checked) but it takes a little more time to write to disk with verify on.

In notes for last issue we mentioned that the plus sign and the asterisk can be used for OR and AND, respectively. It turns out that the minus sign can also be used for the exclusive OR function (one or the other, but not both.) IF (A=1)-(B=2) THEN... would mean that if A were equal to 1 or B were equal to 2 (but NOT A=1 AND B=2) then the condition would be met.

CodeWorks 1st Year Programs on Diskette

All the programs published during the 1st year of CodeWorks magazine are available on the following $5^1/_4$ soft-sector formats:

- ☐ PC/MS-DOS GW-BASIC 40 track DSDD
- ☐ CP/M MBASIC specify format and computer type:
- ☐ TRSDOS Model I 35 track SDSS
- ☐ TRSDOS Model III 1.3 40 track SSDD
- ☐ TRSDOS IV 6.2 40 track SSDD

If your format is not covered by the above, please inquire and we will try to accommodate you.

Only 50¢ per program!

Check "Disk Order" in the order form below and return this page or a photocopy to us. Diskettes will be sent 1st Class postpaid.

Special Subscriber price $20.00

Note: We cannot provide hard-sector formats. Non-subscriber price $30.00

Subscription ORDER FORM

787

NOTE: The entire set of (7) first year issues is still available at $24.95. Please specify "First Year" if you are ordering this set.

Computer type:

Comments:

Please enter my one year subscription to *CodeWorks* at $24.95. I understand that this price includes access to download programs at NO EXTRA charge.

- ☐ New subscription
- ☐ Renewal subscription
- ☐ Check or MO enclosed.
- ☐ Bill me later.
- ☐ Charge to my VISA/MasterCard # _____ Exp. date _____
- ☐ **Diskette Order**

Referred by _____ # _____

Please Print Clearly:

Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to: ***Codeworks***
3838 South Warner St.
Tacoma, WA 98409

Charge card orders may be called in (206) 475-2219 between 9 AM and 4 PM weekdays, Pacific time.

Index Update

Additions to CWINDEX.DAT

Wood.bas, reference, issue 11, page 2
Qkey.bas, correction, delete last record, issue 11, page 3
Card.bas, reference, issue 11, page 3
Form.bas, reference, issue 11, page 3
Rcard.bas, reference, issue 11, page 3
Mcard.bas, reference, issue 11, page 3
Poker.bas, reference, issue 11, page 3
Download, problems with direct statement in file, issue 11, page 4
Misc, route screen to printer, issue 11, page 4
Card.bas, reference, make numbers sort right, issue 11, page 4
Puzzler, generating Pi, discussion, issue 11, page 6
Puzzler, print formatting from arrays, issue 11, page 6
Graph2.bas, main program, issue 11, page 7, plots two curves on same graph
Random files, article w/sample input screen, issue 11, page 11
Misc, program, Keycode.bas, find ASCII values of keys, issue 11, page 15
Ira.bas, main program, issue 11, page 16, compute IRA amounts
Retire.bas, main program, issue 11, page 16, compute withdrawal amounts
Misc, technique, lining up headings with print using, issue 11, page 16
Misc, technique, converting input whole numbers to decimals, issue 11, page 16
Notes, global definition of strings, issue 11, page 20
Notes, syntax for PEEK and POKE, issue 11 page 20
Misc, article on what IRA is, issue 11, page 21
Addressr.bas, main program, issue 11, page 22, make addresses easy
Diff.bas, main program, issue 11, page 27, a file comparison utility
Misc, program, Roll.bas, find length of stuff on a roll, issue 11, page 30
Poker.bas, corrections to prevent cheating, issue 11, page 33
Poker.bas, conversion for Tandy IV, 7 players, issue 11, page 35
Notes, AND and OR with + and * instead, issue 11, page 38
Notes, long lines in GW-BASIC and + instead of ;, issue 11, page 38
Notes, MID$ as a function and as a statement, issue 11, page 38
Notes, &H and &O for hex and octal numbers, issue 11, page 38
Notes, integer division compared to MOD, issue 11, page 38
Misc, update to this index, issue 11, page 40

If you are using Qkey.Bas to keep a running index of CodeWorks articles and notes, these are the changes to bring Cwindex.Dat up to date through the last issue.

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
US Postage
PAID
Permit No. 774
Tacoma, WA

CodeWorks

Issue 13 Sep/Oct 1987

CONTENTS

It's RENEWAL time!

see page 39

CodeWorks

Issue 13 **Sep/Oct 1987**

Editor/Publisher
Irv Schmidt
Associate Editor
Terry R. Dettmann
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Advisor
Al Mashburn

Editor's Notes

Programming a theoretical problem is easy. You just do it and no one seems to care. In the real world, however, it doesn't go like that at all. It seems that the last person to catch an error in a program is the programmer himself.

As a case in point, I was showing Bob Perez the changes to Stat87.Bas that make the divisional standings, and after showing him how it worked I went back to the main menu. He said he wanted to see it again. So I selected 4 from the menu and sure enough, it came up again - but with all the number of wins doubled from the previous time. Oops! Time to go back in an clear the array before displaying this information so that it will not accumulate. Something like that is easy to overlook, and usually takes someone not so close to the program to point out.

Since we pride ourselves on publishing useful programs, it was very disconcerting the other day, when we found something in Wood.Bas (from Issue 3) that may have been better. Leave it to a carpenter to point it out. The problem was to construct four wall-hung shelves from one sheet of wood. It turned out we could, and the program showed how to cut it using mostly cuts down the length of the wood. These, however, were interspersed with several cross cuts.

His question was that since he was using a radial-arm saw and had to turn the saw head ninety degrees for these cuts, why not print out all the lengthwise cuts first so that the saw would not need to be moved. This would be especially helpful where several pieces all had the same width and could all be cut at one setting of the saw, insuring identical dimensions.

Simple, we said. Just sort the pieces by cut code after a solution has been found. Then all the long cuts would be listed first, and cross cuts second. As you can well imagine, it was not that simple. There are pieces that needed to be cross-cut *first* and then cut along the length. So we came up with a new problem for Wood.Bas. How to order the pieces to minimize resetting the saw? Our usual pragmatic answer was to use a table saw instead of the radial arm saw. It's not a very good answer though.

Incidentally, the project had twenty pieces to it and the program dropped out the solution on the first pass. Earlier, we had tried it with slightly larger dimensions for each shelf, and it finally came up with a solution on the 39th of 41 passes! His customer was not pleased with the larger size though, so we went back to the smaller one. It's things like this that put a program through its paces and wring out the last desirable details...

...Which is why larger companies use the Beta test sites for their major productions. They give a program to a whole bunch of people in the industry and let them find the bugs and hangups. Then they fix them and release an error-free (hopefully) program to the public.

This is the annual renewal issue. We hope you will. Although this is being written in the heat of summer, we hope to have the Second Year CodeWorks diskette ready by the time you receive this issue. Diskette sales help us pay the bills, and you get a working copy of every program published at a very nominal price. This year we will also include many of the little "shorties" from Beginning BASIC and elsewhere in the issues. And why not? If there is extra space on the disk they can just as well be included. See page 39 for the details. And, for those of you who didn't opt for the First Year Diskette, it is still available and there is a spot for ordering it as well.

We hope you all enjoyed the summer and are looking forward to a pleasant autumn. We are.

Irv

Forum

An Open Forum for Questions & Comments

...I am considering moving up to MS DOS and GW BASIC from the Color Computer. The computer I am seriously considering is the Leading Edge Model D. There is a deal available at work where we can get a system for less than $900. This is the basic unit with 2 disk drives and monochrome monitor. I would appreciate any comments you may have received from any of your subscribers on this unit...Keep up the excellent work.

Ermon L Higdon
Grain Valley, MO

Comments we have received regarding the Model D have all been very favorable.

I have just received my July/August issue. I am very pleased that you are committed to another year. Although I have a Tandy Model 100 I can run many of your programs. DIFF.Bas was a program I've always been looking for... You people have described how to change the MOD and the LOCATE instruction. Now I would like to ask if you could give me a way to use the function when my BASIC doesn't have it. Is there a way to use the conversion function without writing it every time you have to use it? This would open up many more of your programs for me. I always can get at least one or two programs from one of your issues so with that I am getting my money's worth and then some. Thanks for a wonderful magazine.

Robert Barden
Kissimmee, FL

You can put the code to simulate the MOD function in a defined function. Or, put it into a subroutine and each time you need it, equate your variable to X, go to the subroutine and after returning, equate the X back to whatever it came from. It's still a lumpy way to do it though.

The July/August issue came about a week ago, and finding the program Typer.Bas of interest, I spent several hours entering it into my IBM Compatible - then spent a number of hours correcting my typing goofs. I have run into a problem with it: When I enter RUN, it stops at line 360 with the message "no more information." Why? Keep up the good work.

Roger C Higgins
Fort Madison, IA

From the error message you get in line 360 it appears that you are not using a standard GW BASIC. We can only assume that our equivalent would be "out of data." This error occurs when you try to read more data elements than exist. Data elements must be separated by commas, except that the last one in any line of code should not have a comma after it. Make sure you have as many data elements as you are trying to read. Hope this helps.

I have received issue 12 on the continuation of the Randemo series and found problems with it. It seems that the program has been devoted to the MS DOS type computers, which is a first with your magazine. This is something I am unhappy to see since all other magazines as well have deserted the 8-bit machine in favor of the 16-bit machines...

Raymond J Jasek
Spooner, WI

The Randemo series is designed to work with 8-bit machines as well as 16-bit. The number of bits makes no difference to us at all, but the version of BASIC does. As stated in our reply to your letter, if you will send an LLIST of your program we will check it out and return it to you with comments.

Many thanks for your help on my toggling Kena Characters. Thanks to your great magazine with instructional help, good programs and clear explanations...

My Model III does not recognize the term MOD 6. I need some help. I think it is the MOD function Someplace I remember you wrote in an early CodeWorks issue how to deal with this problem but I can't find it for the life of me...

W P Frakes
Cuyahoga Falls, OH

We found the MOD thing in issue 7 on page 14. We really should have made a Programming Note about it. You can convert the MOD function like this: If it says: IF X MOD Y=0, *you change it to:* X1=INT(X-Y*INT(X/Y)):IF X1=0, *etc. Also see the program Pages.Bas in the July/August 1987 issue on page 15. It shows the conversion for MOD.*

Reference Programming Notes, page 38, issue 12: The minus sign should only be used in simple conditions such as the one given in the example: IF

(A=1)-(B=2) THEN...

In the event B equals 2, the entire condition is numerically evaluated as +1, and any non-zero value is read as true; however, this peculiar feature of Microsoft BASIC would not give a proper evaluation of a more complex condition, such as: IF (C=3)- ((A=1)-(B=2)) THEN..

In this instance, if both B=2 and C=3 then the total expression would be evaluated as -2 (true), contrary to the meaning of the exclusive OR.

The correct way to write a longer set of mutually exclusive conditions is to use the plus sign, as follows: IF (A=1)+(B=2)+(C=3)+(D=4)=-1 THEN...

The left member of the equation evaluates to -1 only in the instance that one and only one of the 4 atomic conditions is true. It is important that the right member of the equation (-1) be stated in this instance. By changing the minus one to minus two, one can stipulate the action to be performed when exactly two (any two) of the conditions apply. I would think this would be useful in game applications.

Jerry Bails
St. Clair Shores, MI

You are right, the minus sign can only be used in simple expressions.

...I would like to see CodeWorks start discussing the newer BASICs. I think that these newer levels of BASIC will be very useful to us down the road and I would like to start using one of them but would like some guidance in selecting an appropriate one.

Thank you for your attention. Am looking forward to a continuation of many interesting issues of CodeWorks.

Robert B Munroe
Glen Ellyn, IL

By "newer BASICs" we assume you are referring to TrueBASIC, TurboBASIC and QuickBASIC. Our own preference is Microsoft's QuickBASIC, followed by TurboBASIC from Borland. We haven't yet had the opportunity to evaluate TrueBASIC. Using QuickBASIC reminded me more of the C language than BASIC. Perhaps if we did an evaluation of all three and printed the results we would all have a better starting point. Good idea.

As usual, your July/August edition of CodeWorks contains some interesting and practical programs. It's always fun to read the articles and (they are) informative too. Speed.Bas is a good example.

There is an error in line 620 of Mr. Keyes' Fract.Bas program on page 27. The line should read:

620 NUM$=LEFT$(FRACT$,(INSTR(FRACT$, "/")-1))

The correction is particularly important if the numerator contains more than one digit, e.g., 100/3.

Robert L Anderson
St. Albans, WV

...My son sold me a Corona computer, monitor, dual disk drive and an Itoh printer. The printer quality is fantastic, but my BASIC programs will print approximately one-half page and quit with the message "device timeout in line XXX." I have typed in MODE LPT1:,,P while in DOS and the results are still the same. Please try to help me or refer me to someone that can.

Rose Mary Gall
Fayetteville, OH

What you did is supposed to work. The "P" parameter is supposed to prevent the timeout condition. Both the IBM and Tandy implementations of MS DOS have this feature. There is no guarantee that all manufacturer's use all of the features in the MODE command of MS DOS. Our only advice would be to check with the Corona dealer.

I recently received a program disk that includes in its instructions "To save space, these files are compressed into ARCHIVE, having an extension of .ARC on its filename." The only reference I can find in any of my publications is a three line paragraph in my copy of Norton Utilities.

There is nothing in my MS DOS version 3.2 manual nor in any manual I have. I thought you might like to cover ARCHIVE in a future issue. Don't know whether or not I can wait that long, but sure would like some help.

David H Smith
Edinburg, TX

You need a program called UNARC to uncompress those files. It should be included on the program disk you received. Both ARCHIVE and UNARC are available on CompuServe and other sources. The reason you cannot find references to it is that ARCHIVE is not a standard feature of MS DOS, but something that was added by others.

Your comment on my letter (in issue 12) regarding the POKE to activate the Special Characters (on the Tandy IV) reminded me that I didn't give you all the information you might

require. Address 2964 decimal is the location for the Special Characters toggle when using either TRSDOS 6.2 or LDOS 6.3 operating systems. If you are using the 6.1 version of TRSDOS the correct address to poke is 3013. The comments regarding the scroll protect apply to both addresses. Hope this helps...

Larry Meehan
Bremerton, WA

...In reference to your Poker7.Bas program for the Tandy IV. It is my experience that to avoid any problems which may arise from the use of the toggle switches CHR$(21), CHR$(22), CHR$(0) and CHR$(16) to access any of the special characters, it is best to include these switches within the lines or statements themselves. Therefore, line 3110 of Poker7.Bas would read: 3110 B$(J)=D$+CHR$(21)+CHR$(SU)+CHR$(21).

Using this technique of getting into and out of special character modes not only keeps things organized, it also saves memory and program space, as well as keeping you from getting into trouble. Unless you properly exit the special character modes, you may either get trapped inside the mode or find out that your normal codes now represent something entirely different.

Michael A Talley
Elkins Park, PA

Thank you. Tandy III and IV owners please note. This looks like the best solution yet.

Case closed? Well, almost but not quite. I seem to have uncovered a peccadillo in Poker's personality. The scenario goes this way...

Player 1 dealt the hand. Player 2 opened and I (player 3) raised with my Aces high two pair. Player 4 raised me. All called, and I raised back. This was all before the draw, and the wagering became quite spirited. Finally, everyone had folded except for me and player 4. I drew my one card, and player 4 stood pat (drat!!). I could only hope he was playing the high pair bluff, as I had failed to make a full house.

After the draw, when it was time to bet, the folded opener (player 2) put $15 into the pot. He didn't seem to know he had folded earlier. It didn't really interfere with the hand except it was an illegal wager. Oh yes, player 4 had a straight, and I went down in flames (double drat!!).

Except for this picayune peccadillo, Poker is alive and well and is behaving itself admirably at last.

Arthur Melanson
Audubon, NJ

Darned if I know what caused that, but glad to hear you are having fun with it.

...In regard to the Mini-Card Database System, I like it but not as much as I did Card.Bas all by itself. The report generator, Rcard.Bas, seems to be too slow. The printer seems to be waiting on the computer about one-half the time as the program builds strings to print. Is there any way to make this function go faster? The main use I have exercised this program on is my son's paper route list - a four column list of names, address, etc., for substitute delivery boys. After it prints eight or 10 lines there is a long pause as the printer sits idle.

Keep up the good work. I hope to see a third year of publication! Your programs are way above those found in (others) for understandability and usability...

Samuel Laswell
South Haven, MI

Your comments regarding the speed of Rcard.Bas are well taken. For openers, add line 715 RETURN. That alone should speed things up considerably. I have talked with Terry about this and we have decided that Rcard.Bas would be an excellent vehicle with which to demonstrate speed-up tactics. We will do an article on it in a near future issue. As you may have guessed, the problem is that with all the string reassignments going on, the garbage man has to come in and clean things up. We neatly avoided the garbage man in Card.Bas by using pointers instead of switching strings. That won't work here because we are dealing with the strings themselves. It will be interesting to see what techniques Terry will use in this case.

...There used to be a Budget Management program on cassette tape. It was a household program to keep track of all the bills and checking account. Would you have anything like this? Or could you make one? I know you could make an even better one.

You sure give a lot of information for so little a price. Keep up the good work.

Walter F Texter
Allentown, PA

There are so many ways to go with a budget program it's hard to get started. Generally, these kinds of program require a lot of user input, and after a while you get tired of it all and quit using it. We are aware of the one you are talking about, and have used it in the past and gave it up for the reasons just stated. A checkbook is something you carry around and keep up as you go along - no need to repeat that process with a computer unless you

need meticulous and uniform records. What we'd like to see is a program that would project cash flow requirements for a few months in advance and print addresses on envelopes for those creditors who don't provide a return envelope. Also it should give you a way to prioritize your bills, to see who is going to get shorted this month (does anyone ever have enough to pay all the bills all the time?). We will entertain suggestions from all readers interested in such a program and perhaps put something together along the suggested lines.

I would like to locate the following programs for a Tandy Model IV with 2 drives and TRSDOS 6.01.02: A General Ledger program, an Inventory program and a Mail List program. Hopefully, they would be available in the public domain or shareware environment but, if not, I would consider buying professionally available programs if I could find them. Can you help?

Jere S Keefer
315 S Park Ave
Mercersburg, PA 17236

The Galactic Mail File system for the Model IV was very good. The other two we are not sure about. You might contact Powersoft, at 17060 Dallas Parkway, Suite 114, Dallas, Texas 75248.

I was most surprised at seeing the organ pipes program in issue 12. Back about 12 years ago, a local church lost its head and replaced a circa 1920 pipe organ - 6 ranks - with an electric organ.

I bought the pipe organ from them and put it away, awaiting my retirement. This is now happening, and I am pulling the organ together to get it going and add additional home-made ranks. (Would like to correspond with anyone else interested.) Thanks, and CodeWorks is really great.

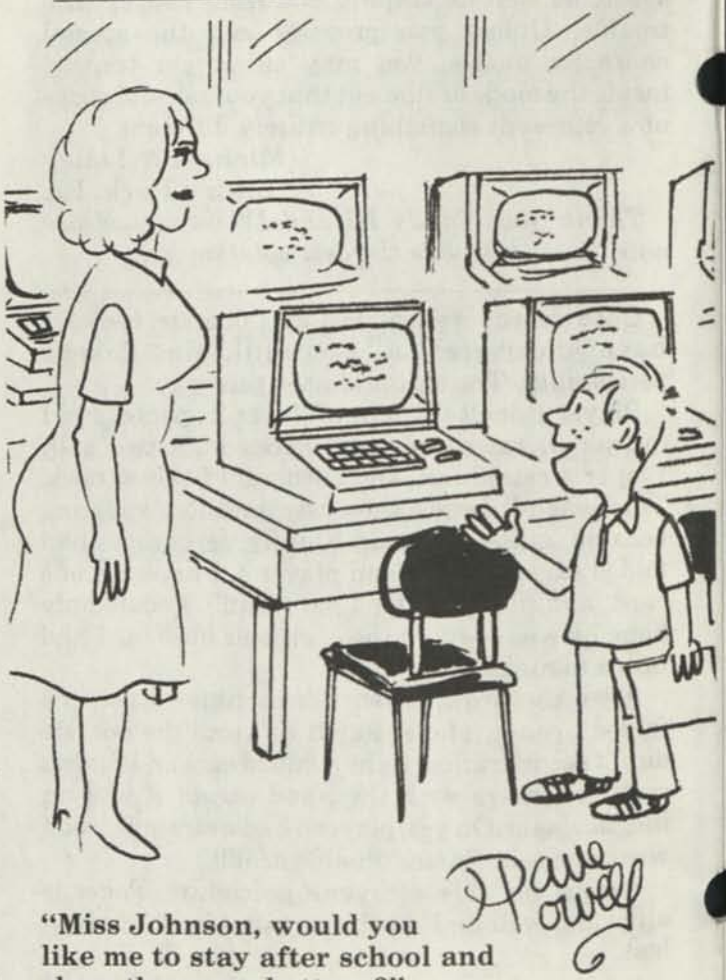
Joe Steakley
PO Box MAPS
Greensboro, AL 36744

I refer you again to the Wicks book mentioned in issue 12. It seems to be a goldmine of good ideas and information. Also, for anyone else interested in musical instruments, string instruments in particular, there is an interesting publication right here in our town called "American Lutherie" which is published quarterly. Their address is 8222 South Park Avenue, Tacoma Washington 98408 (206) 472-7853. Tim Olsen is the Editor and we meet from time to time and talk publishing, computers and music.

Thanks folks. Just a week before the end of July I was fretting because I had so few letters for Forum. It's now the 3rd of August and all of a sudden dozens of letters appeared as if by magic - maybe it's real magic. Now, since you apparently can read my mind, what will be the game in the Nov/Dec 87 issue? We have one scheduled, and also a Mutual Fund tracker program that we think is rather neat and was written by a reader. Also coming then is a personal appointment calendar (I believe some of you asked for that in Forum earlier this year). Then, of course, Terry's Randemo series will be continuing through much of the next year. I, for one, can't wait until he wraps it up because we have uses for it.

The whole reason for telling you all this is that I am unabashedly soliciting your renewals for CodeWorks' third year. And by the time you get this issue the second year diskettes will be ready too. Also by that time, NFL will be in full swing (or just about). We will be putting the weekly stats on the download each Tuesday as we did last year. Isn't Autumn an exciting time? - Irv

---



"Miss Johnson, would you like me to stay after school and clean the erase buttons?"

Ticket.Bas

Create short-run tickets easily

Spencer Hall, Lacey, Washington

Whether it's a concert at the high school, a play by the local little theater or just an epic drama put on by children in the back yard on a lazy August afternoon, Ticketmaster will provide tickets to the show in any quantity you may require. You can even raffle off door prizes using numbers automatically printed on the stub and body of each ticket if you request them. Tickets are in the standard advance sale format, long and narrow, and they are printed ten to the page in two rows of five-up each. You can save the copy to disk, recall it later and change it. This is convenient if sales are better than you expected, if you decide to add more performances on other dates, or if you stage a similar event in the future.

Unlike most programs used to type original text, Ticketmaster doesn't allow much freedom to choose format details such as length of line, number of lines or even which lines may be used. Its job is to guarantee that what you produce looks convincingly like a ticket and that ten of them will fit on a standard 8 1/2 by 11 inch sheet. To this end, it divides the ticket into five panels of four lines each, suggests what ought to be printed in each panel, places a mandatory line of either hyphens or blanks between each panel, and centers the text on each line. As if this were not enough it limits the length of each line to 15 characters.

How to use TICKETMASTER

When you run Ticket.Bas under BASIC you are first presented with the following menu:

Choose:
(1) Enter or edit the current ticket
(2) Review entire ticket
(3) Print tickets
(4) Save the current ticket
(5) Load a ticket
(6) Erase current ticket
(7) Consult HELP panel
(8) Exit from TICKETMASTER

Items 4 and 5 allow saving the text of your ticket to disk and retrieving it later. Suppose your Glee Club has a fall, winter and spring concert. Just bring up the ticket and change the necessary details.

Normally, however, you'll choose option (1) and begin to design a new ticket. As we said above, the ticket is divided into five panels of four lines each. Each panel has a name, which appears on the screen to remind you of what is needed. You may ignore these names and write anything you like in a panel - the program won't know. As you write keep these important facts in mind:

Lines are 15 characters long, maximum. The program won't accept more.

Every line will be centered on the ticket as soon as you ENTER it.

Press the ESC key at any time to return to the menu.

As soon as you have typed the last line of the last panel, the first panel will reappear with the cursor in place of the first character. Press ENTER (or RETURN) to move the cursor to each successive line without change.

If you type something different, however, the new line will be centered in place of the old one as soon as you ENTER it. To restore your previous text don't press ENTER, press ESC.

When only one character is typed on a line and entered it will be reproduced fifteen times, across the full width of the line.

This last feature is great for dressing up your tickets using *,=,$, etc. These lines look especially good on the stub.

It also activates one of Ticketmaster's most powerful features. When you enter a line of number signs, #, the print routine will replace it with a serial number on each ticket beginning with 1000. Do this on the stub and again somewhere on the body of the ticket and you're all set to raffle off door prizes.

The HELP panel, option (7) will remind you of everything we have just presented here.

Option (2) allows you to scroll your complete ticket onto the screen. This feature is always available as you are writing or editing because you can use the ESC key as described above to get to the menu and option (2).

Option (3), the command to print tickets, asks if you wish to suppress the divider lines which appear at the bottom of each panel during option (1), edit and between each panel during option (2), review. Leave them in and it will be easier to see

where to cut the tickets apart. You may also feel that they give the ticket a certain amount of class. You are next asked how many panels of ten tickets you wish. You may decide to choose the default of one panel to use on a photocopier or even an offset press. *(But you lose your consecutive numbering that way. -Ed.)* Ticketmaster tickets look great in offset printing on colored card stock.

Before answering this question, which will start printing, be sure your printer is on line and set for 10 characters per inch and 80 characters per line. These are the default settings for most letter size machines, so that just turning the printer off and on will set it to use Ticketmaster.

System Changes

Ticketmaster places information at specific places on the screen using the subroutine at line 160, which is actually just a pointer to the line containing instructions written for your machine. We've published line 160 as: GOTO 170, but, unless you are using a GW-BASIC computer, you should change this GOTO to one of the other lines which appear in the listing. Also, if your BASIC does not support the FILES command, you will have to delete lines 1170 and 1180 and do without the luxury of seeing the files stored on your disk. If your BASIC has another statement which can list a directory, use it in line 1180.

Program Notes

There are certain things one can do while using a BASIC program that produces errors which the BASIC interpreter will refuse to process. Instead, the interpreter will announce the error and dump the user unceremoniously back into command mode. Variables created during the session are lost. It's the mark of a rank beginner when a program does this. Major applications devote many bytes to detecting exactly what the user has done wrong and politely asking for something else to proceed on. We could, for instance, have examined every character the user entered as a file name in options (4) and (5) and issued warnings when invalid ones were detected. This would have made the program longer. Worse yet, the code which places lines on the screen for editing (820-880 and 500-630) uses string space in huge gulps. It can produce an "out of string space" error in GW-BASIC if you press the ENTER key so fast that there is no time to send out the notorious "garbage collector" to recycle abandoned string memory. We've chosen to solve both problems by creating an "error trap" at the program start which simply says that an error condition exists and returns the user to the main menu. If, while using Ticketmaster, you are suddenly confronted with the main menu you may be certain that this trap has sprung. Not to worry. You haven't lost your text. Just request the desired option and proceed as if nothing happened.

Leave the remark in line 10 there if you are typing this program in from the magazine. You will need the error messages from BASIC to help catch any typos you make. After everything seems to be going well, be sure to remove the remark.

If there is one subroutine that belongs in every BASIC program, it's the one in line 280. It captures a keystroke in Z$. Anything which can be generated by a keystroke can be placed in Z$ and a book could be written about the use a program can make of the value thus captured. The simplest use which can be made of an INKEY$ routine is to pause the screen while the user examines it. In such cases it is convenient to have a line such as 270. By calling this line with GOSUB as is done in lines 870 and 1560, we can give the user advice on how to proceed. In Ticketmaster we get the ASCII number corresponding to the key pressed using the ASC() function to place it in ZA.

Line 790 converts this ASCII code to the actual number. The ASCII code for Arabic numeral 1 is 49. Subtract 48 from 49 and you have a number variable which points to a routine in line 830. The advantage of doing this is that the option numbers are "hot;" no need to press ENTER. You get action as soon as the number is pressed.

Line 280 also contains a test for the ESC key and sends the user back to the main menu if it is detected. With no further programming effort, this option is available to the user whenever subroutine 280 is called.

When we call this subroutine and expect either a (Y)es or (N)o answer we can detect either upper or lower case responses by performing a test on the ASCII value, ZA. Line 910 contains an example of this technique.

To understand why we AND ZA with 95 we must first remind ourselves that the upper case Y has an ASCII code of 89 and the lower case y has an ASCII code of 121. Now remember that the computer stores ZA, not as an Arabic 89 or 121 but as binary 01011001 for 89 or 01111001 for 121. These two binary numbers are identical except for the sixth digit from the right. This is true of the ASCII values for all upper and lower cases letters in the alphabet. The sixth digit from the right is called bit five because computer numbering always start with 0. When this bit is "set," which means that it is a 1, or that electric current is passing thru that address in the chip which records bit 5, it adds the value 32, which is the digit

2 raised to the fifth power. This changes 89, to 121, whose ASCII equivalent, as we just said, is lower case y. If we can "turn off" bit five or, in computer language, reset it (which is the same as changing the 1 to a 0), we have succeeded in changing a lower case letter to its upper equivalent.

The AND function in BASIC lets us do just that. AND compares two binary numbers, bit by bit and leaves a bit "on" or equal to 1 only if it is "on" or equal to 1 in both of the submitted numbers. The binary value of 95 is 01011111. In it, bit 5, the bit we want turned off, is indeed off. AND any number with 95 and all its bits, thru bit 6, will remain unchanged except bit 5, which can never be on because it is being ANDed with a 0.

If you are mentally on your toes you may be asking about that zero in bit 7 at the left end of all bytes. Bit 7, when set to equal 1, adds 128 to the byte value. There is no ASCII character with a value that high which can be entered from the keyboard, so bit 7 doesn't matter. In fact, the ASCII standard stops at byte value 127. Any characters assigned to the numbers from 128 to 255 are assigned at the discretion of the hardware manufacturer. But that is another story. ■

Editor's Notes: If your printer is not set to 80-column width normally, set it there before printing tickets. This program makes use of the automatic line return provided by the line printer.

Also, the ESC key on some machines consists of using the shift and up arrow keys together.

Main listing for Ticket.Bas (GW-BASIC version)

```
5 REM * Ticket.Bas * Written for Codeworks by Spencer Hall
10 'ON ERROR GOTO 1620 'unremark after debugging the program.
20 DATA STUB,EVENT,COMMENT,LOCATION,DATE/TIME/PRICE
30 DEFINT A-Z
40 '  ** Strings for later use **
50 I$=STRING$(15,"-")
60 RN$="      ####      "
70 FOR B=0 TO 4:T$(B,5)=I$:NEXT
80 FOR B=0 TO 4:READ P$(B):NEXT
90 BL$=STRING$(16,32)
100 ZB=15
110 TF$="\              \" ' 14 spaces between \'s
120 '  ** Jump to program start **
130 GOTO 670
140 '                          SUBROUTINES
150 '  ** Screen location **
160 GOTO 170  'NOTE: GOTO the correct line for your computer
170 LOCATE X,Y:RETURN                       'GW-BASIC
180 'PRINT@((X-1)*64)+(Y-1),;:RETURN        '64-col PRINT@ machines
190 'PRINT@((X-1),(Y-1)),;:RETURN           '80-col, 24-line,0-0 PRINT@
    machines
200 'PRINT@(X,Y),;:RETURN                   '80-col, 24-line,1-1 PRINT@
    machines
210 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN '
    CP/Mmachines
220 '   ** Cursor controls **
230 X=L:ZT=24
240 Y=ZT:GOTO 160
250 Y=24:GOTO 160
260 '  ** INKEY$ accept and test keystroke **
270 PRINT TAB(19) "Press any key to continue";
280 Z$=INKEY$:IF Z$="" THEN 280 ELSE ZA=ASC(Z$):IF ZA=27 THEN 670
    ELSE RETURN
300 '  ** Start of line to edit **
310 X=L:Y=24:GOTO 160
320 '  ** Advance line to edit **
330 L=L+1
340 IF L>4 THEN L=1:B=B+1
350 IF B>4 THEN B=0:L=1
360 RETURN
```

1000 Fall Concert	1001 Fall Concert	1002 Fall Concert	1003 Fall Concert	Cc
Annual Fall Concert 1000	Annual Fall Concert 1001	Annual Fall Concert 1002	Annual Fall Concert 1003	
Support your School Band	Support your School Band	Support your School Band	Support your School Band	
Central High School Auditorium	Central High School Auditorium	Central High School Auditorium	Central High School Auditorium	
30 Sept. 1987 7:00 P.M. $5.00	30 Sept. 1987 7:00 P.M. $5.00	30 Sept. 1987 7:00 P.M. $5.00	30 Sept. 1987 7:00 P.M. $5.00	

1005 Fall Concert	1006 Fall Concert	1007 Fall Concert	10 Fa Cor
Annual Fall Concert 1005	Annual Fall Concert 1006	Annual Fall Concert 1007	
Support your School Band	Support your School Band	Support your School Band	
Central High School Auditorium	Central High School Auditorium	Central High Schoc Auditc	
30 Sept. 1987 7:00 P.M. $5.00	30 Sept. 1987 7:00 P.M. $5.00	30 Se 7:	

Sample page of tickets.

We have added the blue dashed lines to show where cuts are to be made. Note also the space to staple near the bottom of each ticket.

```
370 '   ** Display text of ticket **
380 CLS:PRINT P$(B)
390 FOR L=1 TO 5
400 GOSUB 310:PRINT T$(B,L)
410 NEXT L:RETURN
420 L=4:GOSUB 310:PRINT T$(B,L);:RETURN
430 '   ** Get user's filename **
440 ZB=8:ZC$="":PRINT"Enter filename without extension: ";
450 LINE INPUT T$
460 T$=T$+".TKT":ZB=15:RETURN
470 '
480 '   ** INKEY$ keyboard input **
490 '
500 GOSUB 230:ZZ$="":PRINT ZC$;:GOSUB 240
510 PRINT" ";:GOSUB 240
520 GOSUB 280:IF ZA=13 AND ZZ$="" AND ZC$="" THEN ZZ$=" ":GOTO 620
525 IF ZA=13 THEN 590
530 IF ZA=8 AND LEN(ZZ$)=0 THEN 520
540 IF ZA=8 AND LEN(ZZ$)=ZB THEN GOSUB 240:PRINT" ";:ZZ$=LEFT$(ZZ$,
    LEN(ZZ$)-1):GOSUB 240:GOTO 520
550 IF ZA=8 THEN GOSUB 240:PRINT" ";:ZT=ZT-1:ZZ$=LEFT$(ZZ$,LEN(ZZ$)-
    1):GOSUB 240:GOTO 510
560 IF LEN(ZZ$)=ZB THEN 520
570 GOSUB 240:PRINT CHR$(ZA);:ZZ$=ZZ$+CHR$(ZA)
580 IF LEN(ZZ$)=ZB THEN 520 ELSE ZT=ZT+1:GOTO 510
590 IF ZZ$="" THEN 630
600 IF LEN(ZZ$)=1 THEN ZZ$=STRING$(15,ZZ$):GOTO 620
610 ZZ$=STRING$((15-LEN(ZZ$))/2,32)+ZZ$
620 GOSUB 250:PRINT BL$;:GOSUB 250:PRINT ZZ$;:RETURN
630 GOSUB 250:PRINT BL$;:GOSUB 250:PRINT ZC$;:ZZ$=ZC$:RETURN
640 '
650 '              PROGRAM STARTS HERE
660 '
670 CLS:PRINT STRING$(23,45);" The CodeWorks ";STRING$(22,45)
680 PRINT TAB(12)"TICKETMASTER: Ticket Editor and Printer"
690 PRINT STRING$(60,45)
700 PRINT:PRINT"Choose:
710 PRINT TAB(15)"(1) Enter or edit the current ticket"
720 PRINT TAB(15)"(2) Review entire ticket"
730 PRINT TAB(15)"(3) Print tickets"
740 PRINT TAB(15)"(4) Save the current ticket"
750 PRINT TAB(15)"(5) Load a ticket"
760 PRINT TAB(15)"(6) Erase current ticket"
770 PRINT TAB(15)"(7) Consult HELP panel"
780 PRINT TAB(15)"(8) Exit from TICKETMASTER"
790 GOSUB 280:N=ZA-48
800 ON N GOTO 820,1280,900,1060,1170,1360,1400,1580:GOTO 790
810 '   ** Enter or edit **
820 FOR B=0 TO 4
830 GOSUB 380
840 FOR L=1 TO 4
850 GOSUB 310
860 ZC$=T$(B,L):GOSUB 500:T$(B,L)=ZZ$:MV=0
870 NEXT L:GOSUB 270:NEXT B
880 CLS:GOTO 820
```

```
890 '  ** Print tickets **
900 CLS:RN=999:PRINT"Printer must be on line now....":LPRINT " "
910 PRINT:PRINT"Suppress divider lines? (Y)":GOSUB 280:ZA=ZA AND 95
920 GOSUB 230:PRINT:LINE INPUT "Print how many pages of ten tickets
    each: ";NP$
930 NP=VAL(NP$):IF NP=0 THEN NP=1
940 FOR PG=1 TO NP:FOR PN=1 TO 2
950 FOR B=0 TO 4:FOR L=1 TO 4:FOR T=1 TO 5
960 IF ASC(T$(B,L))=35 THEN LPRINT USING RN$;RN+T;:GOTO 980
970 LPRINT USING TF$;T$(B,L);
980 NEXT T:NEXT L
990 IF ZA=89 THEN DL$=" " ELSE DL$=STRING$(15,"-")
1000 FOR T=1 TO 5:LPRINT USING TF$;DL$;:NEXT T
1010 NEXT B
1020 FOR I=1 TO 3:LPRINT " ":NEXT I:RN=RN+5:NEXT PN
1030 LPRINT CHR$(12):NEXT PG
1040 GOTO 820
1050 '  ** Save ticket **
1060 CLS:GOSUB 440
1070 J=1
1080 OPEN "R",1,T$,15
1090 FIELD 1,15 AS L$
1100 FOR B=0 TO 4:FOR L=1 TO 4
1110 LSET L$=T$(B,L):PUT 1,J:J=J+1:NEXT:NEXT
1120 CLOSE:MV=1
1130 GOTO 820
1140 '  ** Load ticket **
1150 'NOTE: If your system does not support the FILES statement or
     uses
1160 '      a different word to list a directory, change or delete
     1180-1190.
1170 CLS:PRINT"Your disk contains the following ticket files.....":
     PRINT
1180 FILES"*.TKT"
1190 GOSUB 440
1200 J=1
1210 OPEN "R",1,T$,15
1220 FIELD 1,15 AS L$
1230 FOR B=0 TO 4:FOR L=1 TO 4
1240 GET 1,J:T$(B,L)=L$:J=J+1:NEXT:NEXT
1250 CLOSE:MV=1
1260 GOTO 820
1270 '  ** Review ticket **
1280 CLS:PRINT"Beginning now, press any key to halt/restart scrolling"
     :GOSUB 280
1290 CLS:FOR B=0 TO 4:FOR L=1 TO 5
1300 PRINT TAB(24) T$(B,L)
1310 Z$=INKEY$:IF Z$="" THEN 1330
1320 Z$=INKEY$:IF Z$="" THEN 1320
1330 NEXT L:NEXT B
1340 GOSUB 270:GOTO 670
1350 '  ** Erase current ticket **
1360 CLS:IF MV=0 THEN PRINT"This ticket has not been saved...O.K. to
     erase it? (Y)" ELSE 1380
1370 GOSUB 280:ZA=ZA AND 95:IF ZA<>89 THEN 670
1380 FOR B=0 TO 4:FOR L=1 TO 4:T$(B,L)="":NEXT L:NEXT B:MV=0:GOTO 820
```

```
1390 '  ** HELP panel **
1400 CLS:PRINT"Press any key to read each of these instructions"
1410 PRINT"or press ESC to return to the menu.":PRINT
1420 PRINT TAB(10)"At any time during ENTER/EDIT you may"
1430 PRINT TAB(10)"return to the menu by pressing ESC.":PRINT:GOSUB
     280
1440 PRINT TAB(10)"Every line you type and <ENTER> will be"
1450 PRINT TAB(10)"centered on the ticket.":PRINT:GOSUB 280
1460 PRINT TAB(10)"If you type and <ENTER> a single character"
1470 PRINT TAB(10)"it will be repeated across the ticket.":PRINT:
     GOSUB 280
1480 PRINT TAB(10)"If the single character repeated across"
1490 PRINT TAB(10)"the screen is the number sign (#), the"
1500 PRINT TAB(10)"printer will center a serial number"
1510 PRINT TAB(10)"on this line on every ticket.":PRINT:GOSUB 280
1520 PRINT TAB(10)"The cursor will return to the first line"
1530 PRINT TAB(10)"after you have typed the last.":PRINT:GOSUB 280
1540 PRINT TAB(10)"Anything you type and <ENTER> over"
1550 PRINT TAB(10)"existing text will replace it.":PRINT:GOSUB 280
1560 GOSUB 270:GOTO 670
1570 '  ** Exit the program  **
1580 CLS:IF MV=0 THEN PRINT "Current ticket has not been saved. O.K.
     to exit? (Y)":GOTO 1600
1590 PRINT "Ready to exit Ticketmaster? (Y)"
1600 GOSUB 280: ZA=ZA AND 95:IF ZA<>89 THEN 670
1610 CLS:ON ERROR GOTO 0:END
1620 RESUME 1630
1630 CLS: PRINT"Temporary error condition....returning to menu"
1640 FOR Z=1 TO 1000:NEXT
1650 GOTO 670
```

Change lines for Ticket.Bas for Tandy III

```
Added--->15 CLEAR 10000
Changed->110 TF$="%                    %" ' 14 spaces between %'s
Changed->160 GOTO 180  'NOTE: GOTO the correct line for your computer
                                                    'GW-BASIC
Changed->170 'LOCATE X,Y:RETURN                     '64-col PRINT@ machines
Changed->180 PRINT@((X-1)*64)+(Y-1),;:RETURN
Changed->270 PRINT:PRINT TAB(19) "Press any key to continue";
Changed->460 T$=T$+"/TKT":ZB=15:RETURN
Changed->1080 OPEN "R",1,T$
Changed->1170 INPUT"ENTER FILE NAME WITH EXTENSION";T$
Deleted-> 1180
Deleted-> 1190
Changed->1210 OPEN "R",1,T$
```

Change lines for Ticket.Bas for Tandy IV

```
Changed->160 GOTO 190  'NOTE: GOTO the correct line for your computer
                                                    'GW-BASIC
Changed->170 'LOCATE X,Y:RETURN                     '80-col, 24-line,0-0 PR
Changed->190 PRINT@((X-1),(Y-1)),;:RETURN
INT@ machines
```

Puzzler

Answers to string insertion and a new one

Puzzler 11 asked how to simulate MID$ on the left side of the equal sign. From the many replies received, we are selecting two that gave rather lucid detail and covered the problem well. Thanks to all others who sent their answers.

Jerry Bails of St. Clair Shores, Michigan wrote:

"No one should be without the use of the MID$ function or its equivalent. Here is a user defined function that will do the same job. As always, a user defined function is preferable to writing the expanded code for each particular string insertion.

```
DEF FNMD$=
LEFT$(A$,X-1)+LEFT$(B$,Y)+RIGHT$(A$,
LEN(A)-(Y+X-1))
```

The following statements are equivalent:

A$=DEFFNMD$(A$,X,Y,B$) as defined above, and:

MID$(A$,X,Y)=B$ as in Microsoft BASIC, where the four parameters in the user defined function are:

A$ is the original string,

X is the position where substring is to be inserted,

Y is the maximum length of substring to be replaced,

B$ is the replacement string.

NOTE: As with the MID$ function, if the replacement string B$ is shorter than Y, then its length determines the number of characters that are replaced. If B$ is longer than Y, it is truncated to length Y before insertion."

Mark Gardner, Glendale, California had this:

"My solution to Puzzler 11 - either one line of code or four lines of code, depending on how closely you wish to simulate the original MID$ statement behavior. In the following discussion, A$ is the source string, B$ is the replacement string, S is the start position and C is the replacement string count.

In the original MID$ statement MID$(A$,S,C)= B$

If S is less than one or greater than length of A$, then the statement is in error.

C is optional. If C is absent, all of B$ is used until each character of A$ from S is replaced. The length of A$ does not change. (C must be provided to new code; if C is set to -1, new code will treat it as missing.)

If C is zero or B$ is empty, then A$ is not changed.

If C is less than length of B$, then only the first C characters of B$ are used.

The length of A$ is never changed, even if specification of C and length of B$ would provide characters for addition.

The complex solution must include these checks on arguments before using the simple solution:

LINE 30: If count is absent (set to -1), use length of replacement string.

LINE 40: If start position plus count would exceed length of source, change count.

LINE 50: If count is greater than number of replacement characters, set equal.

LINE 60: Simple solution - can be used alone if user satisfies requirements.

```
30 IF C<0 THEN C=LEN(B$)
40 IF S+C-1>LEN(A$) THEN C=LEN(A$)-S+1
50 IF C>LEN(B$) THEN C=LEN(B$)
60 A$=LEFT$(A$,S-1)+LEFT$(B$,C)+RIGHT$
(A$,LEN(A$)-C- S+1) "
```

Jerry Bail's use of the user defined function suggested the next puzzler to us. Here it is:

```
10 DEF FNN$(X)=RIGHT$(STR$(X),LEN(STR$
(X))-1)
20 A=1234
30 PRINT FNN$(A)
```

This user defined function will do three things at the same time. What are they?

Some of you purists may argue that two of the three things are inherent in each other and that there are only two things it does. We will accept it either way.

Some of you have said that the puzzlers were too easy. We are trying to encourage more of you to try them. They are not necessarily puzzles, per se, but point out some facet of coding that should help you in your programming efforts. The more such "tools" you acquire in your arsenal, the easier it becomes to solve programming challenges. Besides that - they are sorta fun. ■

NFL87.Bas

The Oracle tries again

Staff Project. We are going to give our Oracle another shot at the NFL season. There are minor changes to NFL87.Bas and some new features in Stat87.Bas. Weekly statistics will be on our download system by Tuesday afternoons during the regular season.

It's here again! The NFL season is underway and our Oracle is back to see if he can do better than he did last year. No cosmic changes have been made to it; we tried, but couldn't find any major way to improve over last year's version. After running through the stats many times and checking against actual scores, however, we decided to be a little more pessimistic about point spreads. (You'll have to admit, it was just a little bit generous last year.)

If you still have NFL86 from last year there are few changes to make to update it to NFL87.Bas. The following lines changed: 100, 130, 1270, 1280 and the data (schedule) lines from 1430 through 1690. Lines 100 and 130 are simply identification changes. Lines 1270 and 1280 are changed to cut the point spread approximately in half over last year. NFL87.Bas will, of course, be available on the download system.

If you want to use NFLSTAT.Bas from last year, it will still work. We have, however, updated it and added a new feature you may find interesting. It now prints the divisional standings through whatever week you have the statistics for. Several additional features have been added to make it easier to use. It is also now called STAT87.Bas to differentiate it from NFLSTAT. It too, will be available on the download.

Stat87.Bas

Although much similarity exists between Stat87.Bas and last year's stat program, enough changes have been made to make it qualify as a new program. If you do not have the MOD function in your BASIC, change lines 570 and 1150 as follows:

```
570 X1=INT(L1-28*INT(L1/28)):IF X1<>0 THEN etc., etc...

1150 X1=INT(I-14*INT(I/14)):IF X1<>0 THEN etc., etc...
```

In addition, if your BASIC is other than GW-BASIC, remark line 230 and un-remark the appropriate line for your particular PRINT@ or CP/M cursor locating convention.

The added lines (from 1360 on) support the new option 4 in the main menu. Figure 1 is a sample output produced by these lines (based on week 15 of last year's data.) Notice that the program awards a half-point for tie games. This section of code prints the division headings on the screen and then goes back and fills in the teams in the proper order. This took a bit of tricky maneuvering since two of the divisions only have four teams while the others each have five. Let's run through this section of code and see how it happens.

When you select option 4 from the main menu program flow goes to line 1370. Since it will take a little bit of time (and it's good practice to never let a user wonder what's happening), this is a good place to print something to let the him know that something is, indeed, going on. So in line 1240 we simply print "Calculating..." and let it go at that.

Up in line 480 we cleared our temporary "games won" array (TS). We need to do this because you may want to select option 4 more than once in a run, and without clearing, the number of wins would double each time you did. This is something that needs to be watched for any time you have accumulating counters.

Now we go through the entire array of Stat.Dat and check to see if a team's score is greater than their opponent's score. If it is, we add one to the "won" array. If it is equal, we add .5 to the "won" array. This all happens in lines 1400 to 1420. Note that in line 1400, L1 tells us how many sets of weekly stats are in the Stat.Dat file. One set being: Team number, week number, 1st downs, points scored and points allowed.

Next, we clear the screen and print the divisional heading at the right spots. Lines 1450 through 1470 do that. In lines 1480 to 1530 we set variables P and Q to the start and end of each set of team numbers for each division. As we said earlier, some divisions only have four teams and the rest have

five, so a simple STEP in the loop will not work. While we are here, we also set the horizontal position to print on the screen (Y) for our general purpose Locate/Print@ subroutine in line 230. Note that the X position of the cursor will need to be incremented *inside* the loop that prints the teams and their wins on the screen. From this section of code, all lines take us to the subroutine at line 1590.

This subroutine first sorts the teams in each division in descending order (lines 1590 to 1670), then prints the list of teams at the appropriate place on the screen (lines 1700 to 1740.) If you do not have the SWAP command in your BASIC, optional lines to do the switching are provided to the right of lines 1630 and 1640. Since the largest number of items to sort will never exceed five, a simple Bubble sort works well here. Notice that the sort and print subroutine from lines 1590 on, work on only one division at a time. The RETURN in line 1750 then takes us back up to 1480-1530 to get the next division to work with.

Notice also, in line 1700 and 1720, how we need to manipulate X (the row to print on the screen.) If the team number is less than 15 (the three NFC divisions), we let X equal 1 for starts, and then increment it down the screen for each team printed. If we are printing the AFC divisions, we start X at row 8 and increment it down from there for each team. The actual incrementing code (X=X+1 in line 1720) *must* be inside the loop where we GOSUB to the Locate/Print@ subroutine or it won't work. Note that the column position (Y) can be outside of this loop because it does not change during the printing of any one division.

In line 1730 we format the number of wins as two places plus one decimal place to take care of the half-games because of ties. If you are wondering where the team name came from all of a sudden, it's been there all the time in T$(I), and the number of wins is contained in the corresponding variable TS(I).

After the last division is printed on the screen, line 1540 prompts us to press Enter for the main menu, and we are done. From a programming standpoint this code tends to illustrate what is necessary to print a "nice" display on the screen. It takes quite a bit of code, as you can see, but it's always fun to watch once you get it working right.

So there we are for NFL87. Even though it's still July as this is being written, we can almost sense the approaching smells of autumn in the air, and are looking forward to another exciting season of the great American pastime. ■

NLF87.Bas main listing (for GW-BASIC)

```
100 REM ** NFL87.BAS * NFL PROJECTION PROGRAM * CODEWORKS MAGAZINE *
110 REM ** 3838 S. Warner St. Tacoma,WA 98409 (206)475-2219 VOICE
120 REM ** (206)475-2356 300/1200 MODEM * Requires a data file made
130 REM ** with the accompanying program STAT87.BAS
140 '
150 'CLEAR 10000:'Use only if your Basic requires cleared string
    space.
160 '
170 DIM A(420,5),B(28,6),T$(28),F(28,5),P(364)
180 '
190 DATA REDSKINS,COWBOYS,EAGLES,GIANTS,CARDS,BEARS,VIKINGS
200 DATA PACKERS,LIONS,BUCS,NINERS,RAMS,SAINTS,FALCONS
210 DATA DOLPHINS,PATRIOTS,JETS,BILLS,COLTS,STEELERS,BROWNS
220 DATA BENGALS,OILERS,SEAHAWKS,RAIDERS,BRONCOS,CHARGERS,CHIEFS
230 '
240 REM * READ IN THE TEAM NAMES *
250 FOR I=1 TO 28
260   READ T$(I)
270 NEXT I
280 '
290 REM * NOW READ IN THE SEASON SCHEDULE **
300 FOR I=1 TO 364
310   READ S:P(I)=S
320 NEXT I
330 '
```

```
340 CLS:'This is a clear screen command, change to suit your Basic.
350 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
360 PRINT"              N F L   F O O T B A L L   O R A C L E"
370 PRINT"                Projects Winner and point-spread"
380 PRINT STRING$(60,"-")
390 PRINT
400 PT=0
410 INPUT"Projection for which week number";W
420 IF W>16 THEN PRINT"Oracle can only project weeks 4 through 16.":
    GOTO 410
430 IF W<4 THEN PRINT"Insufficient Data, wait until week 4 to start":
    GOTO 410
440 INPUT"Enter 1 for printer output, else just Enter";PT
450 W1=W-1:W2=W-2:W3=W-3:W4=W-4
460 PRINT TAB(10)"The Oracle is busy ..."
470 WN=W1*28
480 '
490 REM ** READ STATISTICS FROM STAT.DAT FILE **
500 PRINT"Reading the statistics file ..."
510 PRINT"Throwing chicken bones over his shoulder...
520 OPEN "I",1,"STAT.DAT"
530 FOR I=1 TO WN
540   IF EOF(1) THEN 590
550   FOR J=1 TO 5
560     INPUT #1,A(I,J)
570   NEXT J
580 NEXT I
590 IF I<WN THEN PRINT"Statistics for weeks 1 through";W1;"not
    complete.":END
600 CLOSE 1
610 '
620 REM * FIND AVERAGE FOR SEASON **
630 PRINT"Finding the season average for each team..."
640 FOR X=1 TO 28
650   FOR I=1 TO WN
660     IF A(I,1)<>X THEN 710
670     IF A(I,2)>=W THEN 710
680     FOR J=3 TO 5
690       N(J)=N(J)+A(I,J)
700     NEXT J
710   NEXT I
720   F(X,1)=X
730   FOR J=3 TO 5
740     F(X,J)=N(J)/W1
750   NEXT J
760   FOR J=1 TO 5:N(J)=0:NEXT J
770 NEXT X
780 '
790 REM ** FIND EACH TEAM AVERAGE FOR LAST THREE WEEKS
800 PRINT"Finding the last three week average for each team..."
810 FOR X=1 TO 28
820   FOR I=1 TO WN
830     IF A(I,1)<>X THEN 890
840     IF A(I,2)<W AND A(I,4)>A(I,5) THEN B(X,6)=B(X,6)+1
850     IF A(I,2)<>W1 AND A(I,2)<>W2 AND A(I,2)<>W3 THEN 890 .
```

```
860      FOR J=3 TO 5
870        C(J)=C(J)+A(I,J)
880      NEXT J
890    NEXT I
900    B(X,1)=X
910    FOR J=3 TO 5
920      B(X,J)=C(J)/3
930    NEXT J
940    FOR J=1 TO 5:C(J)=0:NEXT J
950 NEXT X
960 CLS
970 '
980 PRINT"PROJECTION FOR WEEK ";W
990 PRINT"Week";W;TAB(16)"Oracle's";TAB(30)"--------- 3 week Averages
    ---------"
1000 PRINT TAB(16)"Rating";TAB(25)"Won";TAB(30)"1st
     downs";TAB(43)"Score";TAB(54)"Pts Allowed"
1010 IF PT<>1 THEN 1190
1020 LPRINT"The CodeWorks NFL ORACLE PROJECTION FOR WEEK ";W
1030 LPRINT" "
1040 LPRINT"Key to column headings"
1050 LPRINT TAB(10)" 1- Teams plus Oracle's Winner projection"
1060 LPRINT TAB(10)" 2- Oracle's overall rating number (not a score)"
1070 LPRINT TAB(10)" 3- Number of games won this far in the season"
1080 LPRINT TAB(10)" 4- Last 3 weeks average 1st downs"
1090 LPRINT TAB(10)" 5- Last 3 weeks average points scored"
1100 LPRINT TAB(10)" 6- Last 3 weeks average points allowed"
1110 LPRINT TAB(10)" 7- Season average 1st downs"
1120 LPRINT TAB(10)" 8- Season average points scored"
1130 LPRINT TAB(10)" 9- Season average points allowed"
1140 LPRINT TAB(10)"10- Actual score (you fill in after the games)
1150 LPRINT TAB(10)"11- Actual point spread (fill in this too.)
1160 LPRINT" "
1170 LPRINT"1";TAB(16)"2";TAB(21)"3";TAB(26)"4";TAB(30)"5";TAB(34)"6";
     TAB(41)"7";TAB(45)"8";TAB(49)"9";TAB(56)"10";TAB(66)"11"
1180 LPRINT" "
1190 SI=(((W-1)*28)+2)-84
1200 FOR S=SI TO SI+26 STEP 2
1210   X=P(S-1):X1=P(S)
1220   X$=T$(X):X1$=T$(X1)
1230   S0=F(X,3)+B(X,3)+(2*F(X,4))+(4*B(X,4))+(40-F(X,5))+3*(40-B(X,
       5))
1240   T0=F(X1,3)+B(X1,3)+(2*F(X1,4))+(4*B(X1,4))+(40-F(X1,5))+3*(40-
       B(X1,5))+20
1250   S5=INT(S0+.5):T5=INT(T0+.5)
1260   IF S5=T5 THEN X1$=X1$+" by 1"
1270   IF S5>T5 THEN X$=X$+" by"+STR$(INT(((S5-T5)+.5)/10)+1)
1280   IF S5<T5 THEN X1$=X1$+" by"+STR$(INT(((T5-S5)+.5)/10)+1)
1290   PRINT X$;TAB(16);S5;TAB(25);B(X,6);TAB(31);INT(B(X,3));TAB(43);
       INT(B(X,4));TAB(55);INT(B(X,5))
1300   PRINT X1$;TAB(16);T5;TAB(25);B(X1,6);TAB(31);INT(B(X1,3));
       TAB(43);INT(B(X1,4));TAB(55);INT(B(X1,5))
1310   PRINT
1320   IF PT<>1 THEN 1360
```

```
1330    LPRINT X$;TAB(15);S5;TAB(20);B(X,6);TAB(25);INT(B(X,3));
        TAB(29);INT(B(X,4));TAB(33);INT(B(X,5));TAB(40);INT(F(X,3));
        TAB(44);INT(F(X,4));TAB(48);INT(F(X,5));TAB(55)"_____"
1340    LPRINT X1$;TAB(15);T5;TAB(20);B(X1,6);TAB(25);INT(B(X1,3));
        TAB(29);INT(B(X1,4));TAB(33);INT(B(X1,5));TAB(40);INT(F(X1,3));
        TAB(44);INT(F(X1,4));TAB(48);INT(F(X1,5));TAB(55)"_____";
        TAB(65)"_____"
1350    LPRINT" ":GOTO 1400
1360    TC=TC+1
1370    IF TC=>4 THEN PRINT"Press Enter for more";:INPUT XX:CLS:TC=0
        ELSE 1400
1380    PRINT"Week";W;TAB(16)"Oracle's";TAB(30)"--------- 3 week
        Averages ---------"
1390 PRINT TAB(16)"Rating";TAB(25)"Won";TAB(30)"1st
    downs";TAB(43)"Score";TAB(54)"Pts Allowed"
1400 NEXT S
1410 IF PT=1 THEN LPRINT CHR$(12):' Printer top of form command
1420 END
1430 REM * The 87-88 NFL schedule for weeks 4 thru 16 *
1440 DATA 20,14,27,22,21,16,19,18,8,7,10,9,6,3
1450 DATA 5,1,12,13,23,26,28,25,2,17,15,24,11,4
1460 DATA 13,5,3,2,7,6,27,10,9,8,18,16,23,21
1470 DATA 17,19,28,15,1,4,22,24,14,11,20,12,25,26
1480 DATA 19,20,24,9,15,17,16,23,21,22,3,8,13,6
1490 DATA 10,7,4,18,27,25,26,28,5,11,1,2,12,14
1500 DATA 14,23,18,15,6,10,22,20,2,3,26,7,8,9
1510 DATA 16,19,17,1,11,13,5,4,24,25,28,27,12,21
1520 DATA 23,22,19,17,28,6,25,16,13,14,3,5,20,15
1530 DATA 10,8,1,18,21,27,9,26,7,24,11,12,4,2
1540 DATA 14,21,6,8,2,9,26,18,25,7,20,28,27,19
1550 DATA 10,5,1,3,23,11,13,12,15,22,16,4,24,17
1560 DATA 18,21,2,16,9,1,23,20,12,5,19,15,7,10
1570 DATA 17,28,22,14,8,24,13,11,4,3,25,27,6,26
1580 DATA 14,7,18,17,21,23,9,6,8,28,19,16,20,22
1590 DATA 5,3,11,10,4,13,27,24,26,25,15,2,12,1
1600 DATA 28,9,7,2,22,17,8,6,23,19,15,18,13,20
1610 DATA 3,16,5,14,10,12,4,1,26,27,21,11,25,24
1620 DATA 14,2,19,21,28,22,12,9,3,4,27,23,11,8
1630 DATA 24,20,1,5,18,25,16,26,10,13,6,7,17,15
1640 DATA 18,19,22,21,2,1,23,13,15,3,7,8,17,16
1650 DATA 25,28,4,5,20,27,9,10,14,12,26,24,6,11
1660 DATA 8,4,28,26,7,9,16,18,13,22,3,17,20,23
1670 DATA 11,14,24,6,5,10,19,27,21,25,1,15,2,12
1680 DATA 21,20,1,7,18,3,22,23,9,14,8,13,17,4
1690 DATA 5,2,24,28,10,19,27,26,6,25,12,11,16,15
1700 ' End of 1987-88 schedule data
```

The program makes use of teams numbered as follows:

1 - Redskins
2 - Cowboys
3 - Eagles
4 - Giants
5 - Cardinals
6 - Bears
7 - Vikings
8 - Packers
9 - Lions
10 - Buccaneers
11 - 49'ers
12 - Rams
13 - Saints
14 - Falcons
15 - Dolphins
16 - Patriots
17 - Jets
18 - Bills
19 - Colts
20 - Steelers
21 - Browns
22 - Bengals
23 - Oilers
24 - Seahawks
25 - Raiders
26 - Broncos
27 - Chargers
28 - Chiefs

Changes to NFL87.Bas for Tandy III

```
Changed->100 REM ** NFL87/BAS * NFL PROJECTION PROGRAM * CODEWORKS MAG
AZINE *
Changed->130 REM ** with the accompanying program STAT87/BAS
Changed->150 CLEAR 10000:'Use only if your Basic requires cleared stri
ng space.
Changed->490 REM ** READ STATISTICS FROM STAT/DAT FILE **
Changed->520 OPEN "I",1,"STAT/DAT"
Changed->990 PRINT"Week";W;TAB(16)"Oracle's";TAB(30)"------- 3 week Av
erages ---------"
Changed->1000 PRINT TAB(16)"Rating";TAB(25)"Won";TAB(30)"1st downs";TA
B(43)"Score";TAB(52)"Pts Allowed"
Changed->1380   PRINT"Week";W;TAB(16)"Oracle's";TAB(30)"------- 3 week
 Averages ---------"
Changed->1390 PRINT TAB(16)"Rating";TAB(25)"Won";TAB(30)"1st downs";TA
B(43)"Score";TAB(52)"Pts Allowed"
```

Changes to NFL87.Bas for Tandy IV

```
Changed->100 REM ** NFL87/BAS * NFL PROJECTION PROGRAM * CODEWORKS MAG
AZINE *
Changed->490 REM ** READ STATISTICS FROM STAT/DAT FILE **
Changed->520 OPEN "I",1,"STAT/DAT"
```

NFL Schedule Checker

After obtaining the new NFL schedule we used Maker.Bas to create the new data statements. Then we started wondering if there wouldn't be an easy way to check and see if we had entered all the data correctly. So we whipped up a little program (see listing) to make sure that the numbers one through 28 appeared once, and only once, for each week from week four through week 16. It cannot tell if the numbers are placed correctly within each week; we had to run the season on dummy data and check the teams against the published schedule for that. But it does uncover any entry errors.

We extracted the data lines from NFL87.Bas and then merged them with our checking program. It works by taking the team numbers for each week of play and sorting them into ascending order. After being sorted, they should read from one to 28. Then we simply compare the sorted list against a loop count to see if they match. The loop count tells us what number *should* be there, and the sorted list at that same point will tell what actually is there.

The program is essentially a large loop with three other loops operating inside it. The outer loop (from lines 160 through 370) looks at one week at a time from four through 16. The next loop (from line 190 to 210) reads 28 items of data from the data lines and stores the data in A(K). Lines 240 through 310 sort the data in A(K) into ascending order. If you do not have the SWAP command, see line 1530 in Stat87.Bas for the appropriate changes to make. The third loop (lines 340 to 360) does the actual comparison. If the loop count, J, is equal to what is in array element A(J), nothing happens. Otherwise, we print the week number (X tells us what the week number is), what the number should be (the J count), and what the array A(J) holds at that point.

And sure enough, after running the program we found we had punched in the wrong number in two different places. We fixed them and then ran the entire season using last year's stats just to make sure that the right teams were playing each other. They were. ●

```
100 REM Dcheck.bas * check out NFL schedule numbers
110 CLS
120 PRINT "week","should be","was"
130 DIM A(30)
140 '
150 ' for weeks 4 through 16
160 FOR X=4 TO 16
170 '
180 ' read in the 28 team numbers
190   FOR K=1 TO 28
200     READ A(K)
210   NEXT K
220 '
230 ' now sort them into ascending order
240   F=0
250   FOR I=1 TO 27
260     L=I+1
270     IF A(I)=<A(L) THEN 300
280     SWAP A(I),A(L)
290     F=1
300   NEXT I
310   IF F=1 THEN 240
320 '
330 ' compare loop count 1-28 to sorted list
340   FOR J=1 TO 28
350     IF A(J)<>J THEN PRINT X,J,A(J)
360   NEXT J
370 NEXT X
380 'errors will show on the screen.
390 'no errors indicate no duplicate or missing numbers.
400 END
1430 REM * The 87-88 NFL schedule for weeks 4 thru 16 *
1440 DATA 20,14,27,22,21,16,19,18,8,7,10,9,6,3
1450 DATA 5,1,12,13,23,26,28,25,2,17,15,24,11,4
1460 DATA 13,5,3,2,7,6,27,10,9,8,18,16,23,21
         etc.,  etc.,  etc.
```

The program Dcheck.Bas which we used to check the NFL schedule data.

Figure 1

Sample output of Stat87.Bas for divisional standings

```
-NFC East-
13.0 Giants
11.0 Redskins
 7.0 Cowboys
 5.5 Eagles
 3.5 Cards

-NFC Central-
13.0 Bears
 8.0 Vikings
 5.0 Lions
 4.0 Packers
 2.0 Bucs

-NFC West-
10.0 Rams
 9.5 Niners
 7.0 Saints
 6.5 Falcons

-AFC East-
10.0 Patriots
10.0 Jets
 8.0 Dolphins
 4.0 Bills
 2.0 Colts

-AFC Central-
11.0 Browns
 9.0 Bengals
 6.0 Steelers
 4.0 Oilers

-AFC West-
11.0 Broncos
 9.0 Seahawks
 9.0 Chiefs
 8.0 Raiders
 4.0 Chargers

Press enter for menu?
```

Stat87.Bas main listing (for GW-BASIC)

```
100 REM * STAT87.BAS * CODEWORKS MAGAZINE * 3838 S. WARNER ST. TACOMA
    WA.
110 REM * 98409 (206) 475-2219 VOICE (206) 475-2356 300/1200 MODEM
120 REM * Maintains the stats for NFL87.
130 REM * If no file exists then in command mode,type OPEN"O",1,"STAT.
    DAT"
140 REM * and press ENTER, then type CLOSE and press ENTER. This
    creates an
150 REM * empty file called STAT.DAT. You can then run this program.
160 '
170 PRINT"Loading STAT.DAT data file .."
180 ' CLEAR 10000: ' Use only if your machine needs to clear string
    space.
190 DIM A(420,5),T$(28),TS(28),TM(28),TM$(28)
200 '
210 REM * General purpose locate/print@ subroutine
220 GOTO 290
230 LOCATE X,Y:RETURN                        'GW-BASIC
240 'PRINT@((X-1)*64)+(Y-1),;:RETURN         'Tandy I/III
250 'PRINT@((X-1),(Y-1)),;:RETURN            'Tandy IV
260 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN ' CP/M
270 '
280 REM * Set up the team names in data lines
290 DATA Redskins,Cowboys,Eagles,Giants,Cards,Bears,Vikings
300 DATA Packers,Lions,Bucs,Niners,Rams,Saints,Falcons
310 DATA Dolphins,Patriots,Jets,Bills,Colts,Steelers,Browns
320 DATA Bengals,Oilers,Seahawks,Raiders,Broncos,Chargers,Chiefs
330 '
340 REM ** READ IN THE EXISTING STAT FILE **
350 WN=420
360 OPEN "I",1,"STAT.DAT"
370 FOR I=1 TO WN
380   IF EOF(1) THEN 430
390   FOR J=1 TO 5
400     INPUT #1,A(I,J)
410   NEXT J
420 NEXT I
430 CLOSE 1
440 L1=I-1
450 '
460 REM * READ IN THE TEAM NAMES AND CLEAR TEMP (TS) ARRAY.
470 FOR I=1 TO 28
480   READ T$(I):TS(I)=0
490 NEXT I
500 '
510 CLS: ' Clear the screen and home the cursor.
520 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
530 PRINT"          N F L   W E E K L Y   S T A T I S T I C S"
540 PRINT"        Maintains statistics for 1987-88 NFL Football"
550 PRINT STRING$(60,"-")
```

```
560 PRINT
570 IF L1 MOD 28 <>0 THEN PRINT"There is extra (or missing) data in
    the file" ELSE PRINT"The file is currently updated through week";
    L1/28
580 PRINT
590 PRINT TAB(10)"1 - Update the file"
600 PRINT TAB(10)"2 - Edit an item in the file"
610 PRINT TAB(10)"3 - View the entire file"
620 PRINT TAB(10)"4 - Show Divisional standings"
630 PRINT TAB(10)"5 - Save the updated file and END"
640 PRINT
650 INPUT" Your choice";X
660 IF X<1 OR X>5 THEN 650
670 ON X GOTO 710,880,1080,1370,1250
680 END
690 '
700 REM * UPDATE THE FILE ROUTINE **
710 CLS
720 INPUT"UPDATE STATISTICS FOR WHICH WEEK NUMBER";W
730 IF W=<L1/28 THEN PRINT"The file appears to be updated through
    that week.":GOTO 720
740 J=L1+1
750 FOR X=1 TO 28
760   PRINT"For the ";T$(X);" for week ";W
770   INPUT"How many first downs ----------";A(J,3)
780   INPUT"How many points did they score --";A(J,4)
790   INPUT"and they allowed how many points-";A(J,5)
800   A(J,1)=X:A(J,2)=W
810   PRINT
820   J=J+1:L1=L1+1
830 NEXT X
840 PRINT"Press Enter for menu";:INPUT X:GOTO 510
850 END
860 '
870 REM ** EDIT AN ITEM IN THE FILE ROUTINE **
880 CLS
890 PRINT "EDIT DATA - You supply the team number and week number."
900 PRINT
910 INPUT"What team number are you looking for ";X
920 INPUT"What week number are you looking for ";W
930 FOR I=1 TO L1
940   IF A(I,1)=X AND A(I,2)=W THEN 970
950 NEXT I
960 PRINT"That item is not in the file":GOTO 1040
970 PRINT T$(A(I,1));A(I,1):PRINT"Week->";A(I,2):PRINT"1st Dns->";A(I,
    3):PRINT"Score->";A(I,4):PRINT"Pts Allowed->";A(I,5)
980 PRINT
990 INPUT"Enter correct team number ---";A(I,1)
1000 INPUT"Enter correct week number ---";A(I,2)
1010 INPUT"Enter correct 1st downs -----";A(I,3)
1020 INPUT"Enter correct score ---------";A(I,4)
1030 INPUT"Enter correct points allowed ";A(I,5)
1040 INPUT"Press Enter for menu";X:GOTO 510
1050 END
1060 '
```

```
1070 REM * VIEW THE FILE ROUTINE **
1080 CLS
1090 PRINT"TEAM #    ";"WEEK    ";"1ST DOWNS    ";"SCORE    ";"PTS
     ALLOWED"
1100 FOR I=1 TO L1
1110   FOR J=1 TO 5
1120     PRINT USING "###";A(I,J);:PRINT"      "; 'six spaces here
1130   NEXT J
1140 PRINT
1150 IF I MOD 14<>0 THEN 1200 ELSE PRINT"Press Enter for more, or Q
     to quit.";
1160 XX$=INKEY$:IF XX$="" THEN 1160
1170 IF XX$="Q" OR XX$="q" THEN 510
1180 CLS
1190 PRINT"TEAM #    ";"WEEK    ";"1ST DOWNS    ";"SCORE    ";"PTS
     ALLOWED"
1200 NEXT I
1210 GOTO 510
1220 END
1230 '
1240 REM * SAVE THE FILE AND END ROUTINE **
1250 OPEN "O",1,"STAT.DAT"
1260 FOR I=1 TO L1
1270   FOR J=1 TO 5
1280     PRINT #1,A(I,J);
1290   NEXT J
1300 NEXT I
1310 CLOSE 1
1320 PRINT"THE FILE STAT.DAT IS NOW SAVED"
1330 PRINT"END OF PROGRAM."
1340 END
1350 '
1360 REM * find divisional standings *
1370 PRINT "Calculating ...";
1380 '
1390 REM * find games won and fill the TS( ) array
1400 FOR I=1 TO L1
1410   IF A(I,4)>A(I,5) THEN TS(A(I,1))=TS(A(I,1))+1 ELSE IF A(I,
       4)=A(I,5) THEN TS(A(I,1))=TS(A(I,1))+.5
1420 NEXT I
1430 '
1440 REM * clear screen and print the headings *
1450 CLS
1460 X=1:Y=1:GOSUB 230:PRINT"-NFC East-";TAB(25);"-NFC Central-";
     TAB(50);"-NFC West-"
1470 X=8:Y=1:GOSUB 230:PRINT "-AFC East-";TAB(25);"-AFC Central-";
     TAB(50);"-AFC West-"
1480 P=1:Q=5:Y=1:GOSUB 1590
1490 P=6:Q=10:Y=25:GOSUB 1590
1500 P=11:Q=14:Y=50:GOSUB 1590
1510 P=15:Q=19:Y=1:GOSUB 1590
1520 P=20:Q=23:Y=25:GOSUB 1590
1530 P=24:Q=28:Y=50:GOSUB 1590
1540 PRINT:INPUT"Press enter for menu";XX
1550 RESTORE:GOTO 470
```

```
1560 END
1570 '
1580 REM * sort standings into descending order
1590 F=0
1600 FOR I=P TO Q-1
1610   L=I+1
1620   IF TS(I)=>TS(L) THEN 1660
1630   SWAP TS(I),TS(L) ' or TM(I)=TS(I):TS(I)=TS(L):TS(L)=TM(I)
1640   SWAP T$(I),T$(L) ' or TM$(I)=T$(I):T$(I)=T$(L):T$(L)=TM$(I)
1650   F=1
1660 NEXT I
1670 IF F=1 THEN 1590
1680 '
1690 REM * print each team's standing
1700 IF P<15 THEN X=1 ELSE X=8
1710 FOR I=P TO Q
1720   X=X+1
1730   GOSUB 230:PRINT USING "##.#";TS(I);:PRINT" "+T$(I)
1740 NEXT I
1750 RETURN
```

Changes for Stat87.Bas for Tandy III

```
Changed->100 REM * STAT87/BAS * CODEWORKS MAGAZINE * 3838 S. WARNER ST
. TACOMA WA.
Changed->170 PRINT"Loading STAT/DAT data file .."
Changed->180 CLEAR 10000: ' Use only if your machine needs to clear st
ring space.
Changed->230 'LOCATE X,Y:RETURN                    'GW-BASIC
Changed->240 PRINT@((X-1)*64)+(Y-1),;:RETURN    'Tandy I/III
Changed->360 OPEN "I",1,"STAT/DAT"
Changed->570 X1=INT(L1-28*INT(L1/28)):IF X1<>0 THEN PRINT"There is ext
ra (or missing) data in the file" ELSE PRINT"The file is currently upd
ated through week";L1/28
Changed->1150 X1=INT(I-14*INT(I/14)):IF X1<>0 THEN 1200 ELSE PRINT"Pre
ss Enter for more, or Q to quit.";
Changed->1250 OPEN "O",1,"STAT/DAT"
Changed->1630   TM(I)=TS(I):TS(I)=TS(L):TS(L)=TM(I)
Changed->1640   TM$(I)=T$(I):T$(I)=T$(L):T$(L)=TM$(I)
Changed->1730   GOSUB 230:PRINT USING "##.#";TS(I);:PRINT" "+T$(I);
```

Changes for Stat87.Bas for Tandy IV

```
Changed->100 REM * STAT87/BAS * CODEWORKS MAGAZINE * 3838 S. WARNER ST
. TACOMA WA.
Changed->170 PRINT"Loading STAT/DAT data file .."
Changed->230 'LOCATE X,Y:RETURN                    'GW-BASIC
Changed->250 PRINT@((X-1),(Y-1)),;:RETURN       'Tandy IV
Changed->360 OPEN "I",1,"STAT/DAT"
Changed->1250 OPEN "O",1,"STAT/DAT"
Changed->1320 PRINT"THE FILE STAT/DAT IS NOW SAVED"
Changed->1730   GOSUB 230:PRINT USING "##.#";TS(I);:PRINT" "+T$(I);
```

Beginning Basic

Time and time again

In the last issue we looked at three diverse programs. Here, we will look at three more that are somewhat related to each other in that they deal with leap years, days between dates and the determination of day of the week.

The first program (see Leap.Bas) is a very simple and short program to determine whether or not a year is a leap year. What we want to point out here is the reduction of a spoken logic statement to one of code. Stated in crude terms, it says "If a year is divisible by four it is a leap year - except if the year is also divisible by 100 then it is not - but if it is also divisible by 400 it is." How do you reduce that lumpy sentence into clear, concise code?

Since we are talking about something being divisible in all three cases we can use the modulus operator to see if there is a remainder of zero. The code in lines 140, 150 and 160 simply divides the given year by four, 100 and 400, respectively, to see if there is a remainder. Only an even zero remainder will indicate exact divisibility. Those of you who have the MOD function in your BASIC can replace the first part of lines 140, 150 and 160 with the remarked portion at the end of each line. We now have the remainder of the division by four, 100 and 400 in variables Y1, Y2 and Y3. In line 170 we make our logic statement in code. It says "If the year is divisible by four (Y1=0) and *not* divisible by 100 (NOT Y2=0) *or* the year is divisible by 400 (Y3=0) *then* the year is a leap year *otherwise* (else) it is not a leap year. Note that since the two conditions are mutually exclusive (the year must be a leap year or not, it cannot be both) we can use the *else* clause in the same statement to indicate when the year is not a leap year.

The point of all this is that logical expressions can become quite convoluted quickly. Even seasoned programmers will find difficulty with them from time to time. Many times the problem is that not all possible cases are covered in a logical construction. The most straightforward approach to these problems is to define the logic thoroughly before attempting to code it. Look for all cases and include them in your statement. Then try to reduce the statement to its simplest form - and only then, put it into code.

Our second program (see Day.Bas) finds what day of the week any given date falls on (within limits.) The algorithm for this program is contained in lines 310 through 400. For any given date, you may want to run through this section of code using a hand calculator to see what happens. Variable Z in line 400 will end up being a number from 1 to 7, which (in line 420) will point to the correct day of the week in variable D$(Z). The algorithm has been "floating" around in computing for years - we don't know what its origins are. The reason the year 1752 is a limit is that during that year there was a calendar reform and consequently, calculations through that period would be off by several days. Although the upper limit to years is not specified in this program it seems that it may well be 31 December, 3999. The reason for this is that in the year 4000 another correction to the calendar will be necessary to keep "Earth time" tied correctly to "Sun time." Let's not worry about it.

It is interesting to note that line 140 and 200 through 220 exist only so that the month name can be printed in the output line (420). This also required that M$ be dimensioned to 12 in line 120. All that for cosmetic output - nice, but not necessary. The two loops could have been written in one line of code, but isn't it easier to read and follow this way? The error check in line 250 checks for more than 31 days, but what about months that have only 30 or 29? All the lines in the algorithm from 310 to 400 could have been written on one line. Again, it's easier to follow this way. Line 310 is interesting because it lets K equal 1 only for the months of January and February, otherwise it is zero. It could have just as well said: "If M<3 then K=1 ELSE K=0." It works just as well as is, however. Although we don't know who the authors of these programs are, it is interesting to see the different ways that people do things.

The third program (see Daydate.Bas) does what the second program did (in a different way) and additionally, gives the ability to calculate inclusive days between given dates.

Let's pick on it for a while, shall we? There is a nice touch in lines 210 and 220. Where you would usually find a statement like: "IF X<1 OR X>3 THEN 200" to trap for out of range menu selection numbers, this author simply put a GOTO 200 after the ON X GOTO line. If X is not 1, 2 or 3, flow

automatically falls through to line 220 and the question in line 200 is repeated.

This author must have decided to use some sort of reference date - see line 350, where January 1, 1801 is coded directly into the program. (It also turns out that that date was a Wednesday, which explains the sequence starting with Wednesday at line 560.) Using this reference date also neatly avoids the 1752 problem encountered in the Day.Bas program. Both options 1 and 2 of the menu utilize the total days calculator routine and the unreal date trap subroutines. Why is the code from line 550 through 630 a subroutine? It is used once per run, and could have been inserted after line 380.

It looks like this program figures the total elapsed days between zero and the first date, then between zero and the second date and subtracts the two to find the days between dates. It also appears that the 1 January, 1801 date is in there only as a reference for the day of the week. The total days calculator routine (at line 430 and on) is similar, but not the same, as the one used in the program Day.Bas. It also makes an exception for January and February (see line 450) but does it in a different way. Line 380 is especially interesting. Let T=2000 and follow it through with a calculator. You will find that in the end, Q will be equal to 10, and the subroutine at 550 will pick "Monday" out of the list, which happens to be correct. Now do it again with T=2001, and sure enough, Q will equal 12 and will pick out "Tuesday." The whole scheme here works only on the decimal portion of the total days divided by seven. It would be interesting to find out where some of these algorithms originated.

It's fun picking over old programs. You can learn a lot of what to do and also what not to do. The three presented here all work, yet do things slightly differently. Even though their style is not the current CodeWorks style, many interesting ideas can be gleaned from them. ■

Listing 1

```
100 REM * Leap.bas * determines if a year is a leap year
110 'This could easily become a subroutine in another program.
120 '
130 INPUT"Enter the year to be tested ";Y
140 Y1=INT(Y-4*INT(Y/4))        'Y1=Y MOD 4
150 Y2=INT(Y-100*INT(Y/100))    'Y2=Y MOD 100
160 Y3=INT(Y-400*INT(Y/400))    'Y3=Y MOD 400
170 IF (Y1=0 AND NOT Y2=0) OR Y3=0 THEN PRINT "It's a leap year" ELSE
    PRINT "No, it is not a leap year."
180 END
```

For comparison purposes only, here is the above program written in the C language.

```
/* Leap - determines if a year is a leap year. */
main ( )
  {
        int year, yr4, yr100, yr400;
        printf ("Year to be checked.\n");
        scanf ("%d", &year);

        yr4 = year % 4;
        yr100 = year % 100;
        yr400 = year % 400;

        if ((yr4 == 0 && yr100 != 0) || yr400 == 0)
          printf ("Is a leap year.\n");
        else
          printf ("Is not a leap year \n");
  }
```

```
100 REM * Day.bas * finds day of week from year 1753 and after
110 '
120 DIM M$(12)
130 DATA Sunday,Monday,Tuesday,Wednesday,Thrusday,Friday,Saturday
140 DATA Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
150 '
160 FOR I=1 TO 7
170   READ D$(I)
180 NEXT I
190 '
200 FOR I=1 TO 12
210   READ M$(I)
220 NEXT I
230 '
240 INPUT"What is the day (DD)    ";D
250 IF D>31 THEN 240
260 INPUT"What is the month (MM) ";M
270 IF M>12 THEN 260
280 INPUT"What is the year (YYYY)";Y
290 IF Y>1752 THEN 310
300 PRINT"Year must not be prior to 1753":GOTO 240
310 K=INT(.6+(1/M))
320 L=Y-K
330 Q=M+12*K
340 P=L/100
350 Z1=INT(P/4)
360 Z2=INT(P)
370 Z3=INT((5*L)/4)
380 Z4=INT(13*(Q+1)/5)
390 Z=Z4+Z3-Z2+Z1+D-1
400 Z=Z-(7*INT(Z/7))+1
410 PRINT
420 PRINT M$(M);D;Y;" falls on a ";D$(Z)
```

Listing 2

```
100 REM * Daydate.bas * day of week and days between dates
110 '
120 'CLEAR 1000 ' only if you need to.
130 CLS
140 PRINT"Days between dates and day of week
150 PRINT"Valid between 1 Jan 1801 through 31 Dec 2099
160 PRINT
170 PRINT"Enter 1 - for days between dates
180 PRINT"Enter 2 - for day of week
190 PRINT"Enter 3 - to quit
200 PRINT:INPUT"Your choice ";X
210 ON X GOTO 250,350,760
220 GOTO 200
230 '
240 REM * days between dates
250 INPUT"Enter first date (DD,MM,YYYY)";D,M,Y
260 GOSUB 650:GOSUB 430:K=D1
270 INPUT"Enter second date (DD,MM,YYYY)";D,M,Y
280 GOSUB 650:GOSUB 430
```

Listing 3

```
290 T=D1-K+1:T=ABS(T):IF T<>1 THEN 300 ELSE 310
300 PRINT"There are ";T;" days between these two dates (inclusive).":
    GOTO 320
310 PRINT"There is one day between these two dates (inclusive).
320 PRINT:PRINT"Press enter to continue":INPUT X:GOTO 130
330 '
340 REM * day of week routine
350 M=1:D=1:Y=1801:GOSUB 430:K=D1
360 INPUT"Day of week for which date (DD,MM,YYYY)";D,M,Y
370 GOSUB 650:M2=M:D2=D:Y2=Y:GOSUB 430:T=D1-K
380 Q=T/7:W=INT(Q):Q=Q-W:Q=Q*100:Q=INT(Q):Q=Q/7:Q=INT(Q):GOSUB 550
390 PRINT"For the date given the day of the week is ";B$
400 INPUT"Press enter to return to the menu";X:GOTO 130
410 END
420 '
430 REM * total days calculator routine
440 D1=(Y*365)+D:L=M
450 IF M<=2 THEN 490
460 L=(L*.4)+2.3
470 L=INT(L)
480 D1=D1-L:Y=Y+1
490 M=((M*31)+(Y-1)/4)
500 M=INT(M):D1=D1+M
510 IF Y=1900 THEN D1=D1+1
520 RETURN
530 END
540 '
550 REM * days of the week
560 IF Q=0 THEN B$="Wednesday"
570 IF Q=2 THEN B$="Thursday"
580 IF Q=4 THEN B$="Friday"
590 IF Q=6 THEN B$="Saturday"
600 IF Q=8 THEN B$="Sunday"
610 IF Q=10 THEN B$="Monday"
620 IF Q=12 THEN B$="Tuesday"
630 RETURN
640 '
650 REM * unreal date trap
660 IF M<=0 OR M>=13 THEN CLS:PRINT"Unreal month ";D;M;Y;:GOTO 750
670 IF D=0 OR D>31 THEN CLS:PRINT"Bad day ";D;M;Y;:GOTO 750
680 IF D=>31 AND (M=4 OR M=6 OR M=9 OR M=11) THEN CLS:PRINT"There are
    not that many days in that month":GOTO 750
690 IF Y<1801 OR Y>=2100 THEN CLS:PRINT"Year is out of range";D;M;Y;:
    GOTO 750
700 IF M=2 AND D=>30 THEN CLS:PRINT"February does not have that many
    days":GOTO 750
710 IF M=2 AND D=29 AND Y=1900 THEN CLS:PRINT"There was no Feb 29th
    in 1900":GOTO 750
720 IF M=2 AND D=29 THEN Z=Y/4
730 IF Z-INT(Z)<>0 THEN CLS:PRINT"There is no Feb 29th in that year":
    GOTO 750
740 RETURN
750 FOR X=1 TO 2000:NEXT X:GOTO 130
760 CLS:PRINT"Done":END
```

Menumake.Bas

Letting your computer make its own menus

Alex Roosakos, Millbrae, California. Here's a takeoff on our program Maker.Bas (from Issue 1) which automatically creates menus you can merge right into your working programs. It's another case of a program creating a program.

Over half of most BASIC programs consist of housekeeping while the remaining portion of the program does the actual work. This housekeeping consists of main menus, neat displays and instructions and is necessary, but usually takes up much programming time. To eliminate some of the time and effort I created Menumake.Bas to create a menu based on your specifications. I tackled the menu problem since it usually requires the most effort, but you can use the ideas here to help eliminate some of the other areas of housekeeping.

Program Notes

Menumake.Bas actually creates an ASCII BASIC program which you can merge with whatever program you are writing. You can also use the program created by Menumake.Bas as the start of your program and work around it.

The program consists of four blocks of code, two of which are passes through the A$(X) array. In line 140 change the CLS command to suit your computer's clear screen command and add CLEAR 5000 if your BASIC is prior to Version 5.0.

The first block of code (lines 190-420) is an input routine which gets all the information needed to create a menu from the user. You are asked for the title lines, a line preceding the options, the options, a query line, the starting line number the created program will have and the increment between lines of the program (see the sample run.) On a 16-line screen there is enough space for ten lines to be printed including the title of the menu and the options. In other words, if you enter three for the number of lines for the title, then you can enter seven more option lines. As the program stands, you may not use more than three lines of text for the title of your menu; the remaining space is used for options. You may modify this limit by changing the "3" in lines 250 and 260 to the maximum number of lines the title of your menu can have. Also, if your computer has more than 16 lines, you can change the "10-L" in line 340 to utilize the added space by changing the "10" to a greater number.

The last few lines of this block ask you for the starting line and increment values for the program to be created. Use these to fit (squeeze) a menu into an already created program. When you input a visual line (title, option, etc.) you may use the RETURN key to signify a blank line.

Programming the Program

In the next two blocks the program makes two passes through the A$(X) array. The first pass puts the custom code into the A$(X) array. The first few A$(X)'s are stuffed with the title and border of the menu. The title of the menu is centered on the screen in lines 480 and 490. Following the title, the line before the options is coded into the array, and the loop puts the options into the next series of the array A$(X).

The query is put into the array next, and finally an INKEY$ routine is incorporated into the program. In line 650, a REM is put into the array.

After the entire array is filled with code, a second pass is made through it. This second pass attached leading line numbers to each A$(X) in order for BASIC to accept it as a program, and not as a sequential or direct file. This loop uses the starting line number (S) and the increment value (N) to create the proper sequence of line numbers.

The next and last block of Menumake.Bas writes the finished menu to disk under a filename you tell it, using a sequential file loop. You can now load and use your newly created menu.

Program starts on page 32

Using the Menu

The INKEY$ routine in this menu saves the user response in I$. You must change the line containing the REM to a line containing a statement redirecting program control, such as an ON GOTO statement. If you use an ON GOTO, use VAL(I$), as the numeric variable (i.e., ON VAL(I$) GOTO ...) The newly created program is saved as an ASCII file.

(Ed. note: Those of you with BASIC prior to version 5.0 should also insert CLEAR 1000: right after the CLS: in line 460, otherwise the STRING$ used will run you out of default string space.) ■

Sample run

Below is the program Menumake created when we tried it. Below that is a screen dump of the results of that program.

```
100 CLS:PRINT STRING$(64,"-");
110 PRINT TAB(18);"T E S T  for  M E N U M A K E"
120 PRINT TAB(23);"to see if it works"
130 PRINT STRING$(64,"-");
140 PRINT
150 PRINT TAB(20);"Here are your options"
160 PRINT TAB(20);" 1. Input data"
170 PRINT TAB(20);" 2. Output data"
180 PRINT TAB(20);" 3. Quit"
190 PRINT TAB(20);"Your choice ";
200 I$=INKEY$:IF I$="" THEN 200
210 PRINT I$;:FOR X=1 TO 200:NEXT X
220 IF VAL(I$)>0 AND VAL(I$)< 4 THEN 250
230 PRINT:PRINT TAB(20);"BAD ENTRY: PLEASE RE-ENTER"
240 FOR X=1 TO 1000:NEXT X:GOTO 100
250 REM * PLACE AN ON VAL(I$) HERE
```

```
----------------------------------------------------------------
                T E S T  for  M E N U M A K E
                    to see if it works
----------------------------------------------------------------
                  Here are your options
                   1. Input data
                   2. Output data
                   3. Quit
                  Your choice
```

```
100 '-----------------------------------------------------
110 '                   Menu Program Maker
120 '                 By Alex Roosakos, 1987
130 '-----------------------------------------------------
140 CLS:DIM A$(50),P$(20)
150 PRINT STRING$(64,"-")
160 PRINT TAB(22);"M E N U  M A K E R
170 PRINT TAB(21);"By Alex Roosakos 1987
180 PRINT STRING$(64,"-")
190 '-----------------------------------------------------
200 '                   Get user input
210 '-----------------------------------------------------
220 PRINT:PRINT"To leave a line blank (such as a title line "
230 PRINT"or an option, etc.), answer prompt with <ENTER> "
240 PRINT"instead of a string.":PRINT
250 PRINT"How many lines is your title (no more than 3)"
260 INPUT L:IF L=0 OR L>3 THEN 250
270 FOR I=1 TO L
280   PRINT"Enter line #";I;" of the title: ":LINE INPUT T$(I)
290 NEXT I
300 PRINT"Enter the line to appear before the options:"
310 LINE INPUT B$
320 PRINT"How many options will be in your menu"
330 PRINT"  (no more than ";10-L;")"
340 INPUT P:IF P=0 OR P>10-L THEN 320
350 FOR I=1 TO P
360   PRINT"Enter option #";I;".";:LINE INPUT P$(I)
370 NEXT I
380 PRINT"Enter the query line: "
390 LINE INPUT Q$
400 INPUT"What starting line number would you like ";S
410 INPUT"What increment between lines would you like ";N
420 PRINT:PRINT"Working . . . "
430 '-----------------------------------------------------
440 '      1st pass: put info into strings
450 '-----------------------------------------------------
460 A$(1)="CLS:PRINT STRING$(64,"+CHR$(34)+"-"+CHR$(34)+");"
470 FOR I=2 TO L+1
480   Z$=STR$(32-INT(LEN(T$(I-1))*.5))
490   A$(I)="PRINT TAB("+RIGHT$(Z$,LEN(Z$)-1)+");"+CHR$(34)+T$(I-1)+
      CHR$(34)
500 NEXT I
510 A$(L+2)="PRINT STRING$(64,"+CHR$(34)+"-"+CHR$(34)+");"
520 A$(L+3)="PRINT"
530 A$(L+4)="PRINT TAB(20);"+CHR$(34)+B$+CHR$(34)
540 FOR I=L+5 TO P+L+4
550   X=I-L-4
560   A$(I)="PRINT TAB(20);"+CHR$(34)+STR$(X)+". "+P$(X)+CHR$(34)
570 NEXT I
580 Y=P+L
590 A$(Y+5)="PRINT TAB(20);"+CHR$(34)+Q$+" "+CHR$(34)+";"
600 A$(Y+6)="I$=INKEY$:IF I$="+CHR$(34)+CHR$(34)+" THEN"+STR$(S+N*(Y+
    5))
```

```
610 A$(Y+7)="PRINT I$;:FOR X=1 TO 200:NEXT X"
620 A$(Y+8)="IF VAL(I$)>0 AND VAL(I$)<"+STR$(P+1)+" THEN"+STR$(S+N*(Y+
    10))
630 A$(Y+9)="PRINT:PRINT TAB(20);"+CHR$(34)+"BAD ENTRY: PLEASE RE-
    ENTER"+CHR$(34)
640 A$(Y+10)="FOR X=1 TO 1000:NEXT X:GOTO"+STR$(S)
650 A$(Y+11)="REM * PLACE AN ON VAL(I$) HERE"
660 '-----------------------------------------------------
670 '         2ND  Pass: attach leading line #s
680 '-----------------------------------------------------
690 FOR I=S TO S+N*(Y+11) STEP N
700   M=M+1
710   A$(M)=STR$(I)+" "+A$(M)
720 NEXT I
730 '-----------------------------------------------------
740 '       Get file info and write program to disk
750 '-----------------------------------------------------
760 PRINT"Program creation done, please enter filename and extension "

770 LINE INPUT F$
780 PRINT "Writing program to disk...
790 OPEN "O",1,F$
800 FOR I=1 TO Y+11
810   PRINT #1, A$(I)
820   PRINT " .";
830 NEXT I
840 CLOSE
850 PRINT:PRINT "Done.
860 PRINT"Now, just load the program ";F$
870 PRINT"to use your menu.
880 END
```

Programming Notes

LINE INPUT #1 from a disk file will read in lines from an ASCII program or file *up to the carriage return* in most of the BASICs we cover. All BASIC program lines end with a carriage return (you actually put it there when you enter the line.) It is interesting to note that (on MS-DOS BASIC at least) you can only enter 251 characters into a line directly from the keyboard. You can, however, edit the line (or concatenate another string to it) to make it 255 characters long. When you try to put one more character into the line after the 255th, BASIC will tell you that the string is too long.

Tandy Models I, III and IV have an annoying habit of erasing the line just below where you PRINT@. To keep this from happening, just put a semicolon after the PRINT@.

MS-DOS machines have a "print screen" key. Holding down a shift key and pressing "Print" will print the current screen to the printer. Holding down the CTRL key and pressing "Print" will start sending all output to both the screen and the printer. Pressing them again will stop the output to the printer. Does anyone know of a way to include such routing from BASIC? In some of the older machines, you could include a BASIC statement that said: SYSTEM "DUAL ON" (or OFF) to send selected portions of output to the printer as well as the screen. We have been looking for the CHR$ combination that would represent the CTRL-Print keys, but can't seem to find it.

Random Files

An Indexing program to work with Randemo5

Terry R. Dettmann, Associate Editor. In this installment of the Randemo series, Terry introduces indexing as a way to sort large files. The index program is made by using much of what is already in Randemo5. He also provides a sample indexing program so we can all see how it works.

What is Indexing?

A very simple problem to deal with in the original CARD.BAS program was sorting the database. In order to sort, you just ran the data through a sort routine in the program. Since everything was in memory, there was no real problem. With RANDEMO.BAS, it's no longer so easy.

How do you sort a file which is enormously larger than memory? You could sort it using the same techniques used for an in memory sort, just view the file the same way you do an array in memory. Simple isn't it? Problem is, you may have to wait until the next coming before any realistic sort is done. Disk access speed (especially with floppy disks) will kill you.

Beyond the problem of pure sorting, is the problem of having to move around the data within the file. Moving all of this data to get it in order every time you want to sort is a major time consumer. So how do we get around it? How can we live with the disk access times?

There are two things we have to get rid of, 1) the need for a lot of access to the data base to get data and 2) the need to move records around from where they exist. The answer to this is indexing.

An index is a list of the records we want in the order in which we want to access them. We can build an index of a data base by scanning through the data base and recording the record number and the entry we want to sort on. With a smaller file, we can sort easily. This is still a major job if we want to do it for a very large file. Eventually, we get to a point where we must build the index as we enter data into the data base.

However, we'll start with the simpler approach. To start with, we'll build our index after the data is entered into the data base. This is sometimes useful since we have a large overhead if we have to enter a record in the data base and the index at the same time. This can slow down data entry. By turning off automatic indexing, we can speed up data entry, but then we're back to needing an indexing program.

To illustrate the principal, I've included a small sample program that builds a tiny (10 item) data base in memory and then indexes it. This program isn't realistic, but you will notice that CARD.BAS uses the same technique for sorting.

The INDEX.BAS Program

INDEX.BAS is a simple demonstration program which illustrates the concept of an index. Lines 10-40 set up the program. I've allocated 10 data lines (LN$(10)) and so 10 index entries (IX(10)). IDX is a flag to keep track of whether the current data in the program is indexed or not.

Lines 200-300 are a simple menu control for the program allowing you to end the program (option 0), enter 10 items of data (option 1), index the 10 items (option 2), display how they are stored in LN$ (option 3), and display the indexed data (option 4).

Subroutine 1000, Enter Sample Data, simply enters ten lines of any text you want to enter into LN$. Nothing fancy, just enter it. Once entered, if you select option 3 (subroutine 3000), the program will show you what you typed in. If you also try to select option 4 (subroutine 4000), you'll find that the program tells you that the "Data isn't indexed yet".

Select option 2 (subroutine 2000). It will index the data (subroutine 2100) by entering the record number of each record in the index array IX. Then the data is sorted (subroutine 2200) using a SHELL sort to arrange the index so that the numbers at each location point to the data in sorted order.

Now if you try to display the data by selecting option 4, you'll see the data in sorted order. Selecting option 3, however, shows that it is *still in the original order in LN$!*

This is the point of indexing, we've arranged the data without moving it. We'll do the same thing with our random files.

Let's Start with Randemo5

In order to build an indexing program

(RANINDEX.BAS), we'll build on what we've already done by borrowing code from the RANDEMO5.BAS program.

RANDEMO5.BAS has all of the routines already in it to allow us to access the data base we built. We could rebuild them all, but you'll find there's an advantage in working with a copy of the original program AND maintaining the same line numbering. *Be sure you keep an unaltered copy of RANDEMO5 - it's still the main program - we are simply borrowing some of it to make the index program.* By starting with a copy of RANDEMO5.BAS renamed to RANINDEX.BAS, we can preserve everything we've done and use the same subroutines we're already familiar with.

Further, as we make changes in the future making RANDEMO more powerful, we can make EXACTLY that SAME CHANGE in RANINDEX to keep the two programs consistent. (We'll do the same thing with our RANPRINT program when we build our printing utility.)

Let's strip out what we don't need from our copy of RANDEMO5 and see what it leaves us. We won't be needing the definitions in lines 51-53, so delete them. Similarly, we don't need to load a screen display, so delete line 160 and the subroutine it refers to (lines 6000-6525 can go).

The whole main line of the program will be replaced, so delete lines 220-350. We'll only be scanning records, so we need to retain a subroutine which gets them for us (1400) and we'll do some work with them so keep everything between 500 and 970 (a couple of these won't get used, but we'll consider these to be our 'library').

The subroutine at 5000 is needed to load the mapping routines. Everything else can go, so delete lines 1000-1310 and lines 1500-4080. This leaves us with a pretty short program, but a lot of useful utility routines. We'll make a few changes in the lines that still exist (including adding something to our map reading routines). Let's go through the listing to see what we've added.

Adding Indexing

Starting at the beginning of the program, the most important change of all occurs in line 10, just to let you know what program this is now. Next we added line 31 to define the arrays DA$ which will hold the contents of the fields we want to sort on, IX which will hold the index to array DA$ and array IR, and the array IR which will hold the record number where we'll find the data in the corresponding DA$ array entry. I've defined each to be 400 in size so this is the present limit of the program. You can make these numbers as large as your computer will allow, this will set the limit on the size of random data base you can handle. You should be able to get this to be pretty big.

In order to expand beyond whatever limit you can set internally, we'll have to change our techniques a little. Later, we'll change the indexing program by adding a sort merge for the data that will make the size of your data base almost unlimited (at least, normal hard disks today won't hold that much data).

Moving on, we add a new variable (NX) which will be the number of records indexed. We change the screen header (lines 110 and 210) to reflect the new program, but otherwise file initialization remains the same as it was.

Once we get to line 215, the program becomes indexing instead of random file editing. First, we'll allow index files to be named anything. Each such file will have an extension '.IDX' to indicate its use as an index. By allowing any name for an index, we can create many index files for a single data base. Each index can serve a special purpose and coexist at the same time on our disk.

Next, we get the number of the field to sort on (from the '.map' file). If we don't get a legal field number, line 230 will return us to the input again.

Lines 240 through 270 are a scan through the data base. Each record is retrieved (subroutine 1400) and checked to see if it is deleted (line 250). If the record has not been deleted, then we add it to the index (subroutine 1000). This subroutine starts by advancing the counter of records selected (line 1010) and then grabs a copy of the data field we want to sort on (line 1020) and setting the index to the current location.

Once the data is all in memory, we sort it the same way we did in the INDEX.BAS program (SHELL SORT). Subroutine 2000 takes care of this in exactly the same way that we used in INDEX.BAS. Subroutine 3000 saves the index to the index file.

With the data saved, we have the option (line 300-310) to check the index data on the screen. Lines 400-450 prints each record to the screen by retrieving them in the order specified by the sorted index.

All of these changes are pretty clear, but there is a subtle change which is essential to the operation of the program. We've added line 5001 to set the counter CX to zero, and line 5225 to increment CX every time we define a map field. These two lines have to be added because our screen mapping routine was used to define them in RANDEMO5. Here we don't have a screen and we want to access any mapped field, therefore, we'll just count the number of fields defined in the mapping.

Now you've got it, you can generate an index to any data file you've built with RANDEMO5. Such an index will be sorted on the data in the file. We could sort by name, city, or whatever fields you've defined for the data base. There are some limitations at this point however:

1) The sort does a string sort of the data in the fields. If we've stored numerical data there, the sort will sort it into alphanumeric order instead of numeric order. We can change the sort to allow numeric sorting later.

2) The program selects all non-deleted records for sorting. We could improve the program by allowing you to select data for the index and choose to sort it or not. For example, you might want to pick out all records for the state of Washington without sorting (this will speed things up).

3) The program is limited to the size of the index arrays defined at the beginning of the program. By modifying the index program to do a sort/merge style operation, we can build unlimited size data bases which are fully sorted.

What we'll do with Indexing next issue

Next issue, we'll make use of all of this indexing to control access to the data base by building a report program generator that will use an index we build using this program. This will give us a very flexible way of building a data base and then using what we have to build interesting reports. We'll add selection to the indexing program (so you can tell it which records to report on) and then build the compatible report program generator which will write a report on the records indexed in the order they're indexed in.

With a report generator in hand, RANDEMO becomes more than a play program, it becomes a serious tool for data base operations. If you've a mind to, try building a report generator yourself. I've shown you how to do it (well sort of anyway) in the RANINDEX program in lines 400-450. Could you go from there to write a useful report program? Look back to the report program we did to go with CARD.BAS, this should give you some interesting ideas. If you do try, write and let me know how you've done. I'd be interested in seeing what you're coming up with. ■

```
10 REM --- RANINDEX.BAS - Random File Indexing
20 REM --- Terry R. Dettmann for Codeworks Magazine
30 DIM FP$(20), SC$(24), XY(20,3)
31 DIM DA$(400), IX(400), IR(400)
40 DEF FNCTR$(X$)=STRING$((WD-LEN(X$))/2," ")+X$
41 DEF FNLF(X) = LOF(X)/128
50 WD=80:LN=24
51 NX=0
60 FALSE=0:TRUE = NOT FALSE
100 REM --- file setup
110 CLS:PRINT FNCTR$("RANDOM FILE INDEXING"):PRINT:PRINT
120 LINE INPUT"FILENAME: ";FF$
125 FD$=FF$+".dat":FS$=FF$+".stk"
130 OPEN "R",1,FD$:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
135 IF LOF(2)=0 THEN LSET SK$=MKI$(1):PUT 2,1
140 FM$=FF$+".MAP":FX$=FF$+".SCN"
150 GOSUB 5000: REM Read Map
170 GOSUB 5300: REM Setup Fielding
200 REM --- main menu
210 CLS:PRINT FNCTR$("RANDOM FILE INDEXING"):PRINT:PRINT
215 LINE INPUT"Name of the index: ";FI$:FI$=FI$+".idx"
220 INPUT"Sort on what field number";FX
230 IF FX<1 OR FX>CX THEN PRINT"OOPS - no such field":GOTO 220
240 FOR RN=1 TO FNLF(1):GOSUB 1400
250     IF FP$(1)="DELETED" THEN 270
260     GOSUB 1000
270 NEXT RN
280 GOSUB 2000
290 GOSUB 3000
```

```
300 LINE INPUT"Want to check the index (y/n)?";YN$
310 IF LEFT$(YN$,1)="N" OR LEFT$(YN$,1)="n" THEN 500
400 REM --- Print the indexed data base to the screen
410 FOR I=1 TO NX:RN = IR(IX(I)):GOSUB 1400:PRINT"RECORD: ";RN
420    FOR J=1 TO CX:PRINT "FIELD(";J;"): ";FP$(J):NEXT J
430    LINE INPUT"Press RETURN/ENTER to continue, END to end";IN$
440    IF IN$="END" OR IN$="end" THEN 500
450 NEXT I
500 REM --- End of Program
510 CLS:PRINT"All Done":CLOSE:END
600 REM --- input a character
610 C$=INKEY$:IF C$="" THEN 610
615 IF LEN(C$)>1 THEN GOSUB 700
620 RETURN
700 REM --- look for arrows
710 C = ASC(MID$(C$,2,1))
720 IF C=72 THEN C$=UP$ ELSE IF C=77 THEN C$=RT$
730 IF C=80 THEN C$=DN$ ELSE IF C=75 THEN C$=LF$
740 RETURN
800 REM --- GOTO XY ROUTINE
810 LOCATE X,Y:RETURN
900 REM --- break line
910 FOR K=1 TO 10:BL$(K)="":NEXT K
920 JN$=IN$:NB=1
930 K = INSTR(JN$,":"):IF K=0 THEN BL$(NB)=JN$:RETURN
940    BL$(NB) = MID$(JN$,1,K-1)
950    NB = NB + 1
960    JN$ = MID$(JN$,K+1)
970 GOTO 930
1000 REM --- Add the record to the index
1010 NX = NX + 1
1020 DA$(NX) = FP$(FX)
1030 IX(NX) = NX:IR(NX)=RN
1040 RETURN
1400 REM --- get record from data base
1410 IF RN<1 OR RN>FNLF(1) THEN RETURN
1420 GET 1,RN
1430 RETURN
2000 REM --- Sort the index
2010 DF = NX:PRINT "SORTING ..."
2020 IF DF = 1 THEN RETURN
2030    DF = INT(DF/2)
2040    SWP = FALSE
2050    FOR I=1 TO NX-DF
2060       IF DA$(IX(I))>DA$(IX(I+DF)) THEN GOSUB 2100:SWP = TRUE
2070    NEXT I
2080    IF SWP THEN 2040 ELSE 2020
2100 REM --- swap the data fields
2110 T = IX(I):IX(I) = IX(I+DF):IX(I+DF) = T
2120 RETURN
3000 REM --- Save the index
3010 PRINT"Saving the index"
3030 OPEN "R",3,FI$,2
3040 FIELD 3, 2 AS XX$
3050 FOR I=1 TO NX
3060    LSET XX$ = MKI$(IR(I))
```

```
3070     PUT #3,I
3080 NEXT I
3090 CLOSE #3
3100 RETURN
5000 REM --- read data map
5001 CX = 0
5005 OPEN"I",3,FM$
5010 IF EOF(3) THEN 5035
5015     LINE INPUT#3,IN$
5020     GOSUB 900
5025     GOSUB 5100
5030 GOTO 5010
5035 CLOSE#3
5040 RETURN
5100 REM --- decode map line
5110 IF BL$(1)="FIELD" THEN GOSUB 5200:RETURN
5120 RETURN
5200 REM --- define a field
5210 NF = VAL(BL$(2)):FL = VAL(BL$(4)):FP = VAL(BL$(5))
5220 XY(NF,0)=FL:XY(NF,3)=FP
5225 CX = CX + 1
5230 RETURN
5300 REM --- Map Fields
5310 FOR I=1 TO CX
5320     NL = XY(I,3)
5330     FIELD #1, NL-1 AS X$,XY(I,0) AS FP$(I)
5340 NEXT I
5350 RETURN
```

Sample indexing program

```
10 REM --- Sample index program
20 DIM LN$(10), IX(10)
30 FALSE=0:TRUE = NOT FALSE
40 IDX = FALSE
200 REM --- Main loop
210 CLS:PRINT"Index Demonstration":PRINT:PRINT
220 PRINT "0. End of Program"
230 PRINT "1. Enter Sample Data"
240 PRINT "2. Index the Data"
250 PRINT "3. Display the Data"
255 PRINT"4. Display Indexed Data"
260 PRINT:PRINT
270 INPUT"Option: ";CD:IF CD=0 THEN CLS:PRINT"All done":END
280 IF CD<0 OR CD>4 THEN PRINT"OOPS ... No such option, try again":
    GOTO 270
290 ON CD GOSUB 1000,2000,3000,4000
300 GOTO 200
1000 REM --- Enter Sample Data
1010 CLS:PRINT"Enter Sample Data":PRINT:PRINT:IDX = FALSE
1020 FOR I=1 TO 10
1030     PRINT "Enter Item ";I;":";
1040     LINE INPUT LN$(I)
1050     IF LN$(I)="" THEN RETURN
1060 NEXT I
1070 RETURN
2000 REM --- Index the Data and sort it
2010 PRINT "Indexing the Data"
2020 GOSUB 2100
```

Continues on page 40.

ORDER FORM

*(Please note that **unless** you have already renewed, **all** subscriptions **expire with this issue.**)*

WHAT?

Issues

- ☐ Renew Subscription $24.95
- ☐ NEW Subscription (starts with Nov/Dec 1987 issue)..... $24.95
- ☐ All 1st Year CodeWorks issues (Issue 1 through Issue 7) $24.95
- ☐ All 2nd year CodeWorks issues (Issue 8 through Issue 13) ... $24.95

Diskettes

- ☐ 1st year programs on disk (specify type below) $20.00
- ☐ 2nd year programs on disk (specify type below) $20.00

- ☐ PC/MS-DOS 40 track DSDD
- ☐ CP/M 5 1/4 inch (specify format and computer type here _____

_____

- ☐ TRSDOS Model I 35 track SSSD
- ☐ TRSDOS Model III 40 track SSDD
- ☐ TRSDOS Model IV 40 track SSDD

Clip or photocopy and mail to **CodeWorks,**
3838 S. Warner St.
Tacoma, WA 98409

Computer type:

Comments:

HOW?

- ☐ Check/MO enclosed
- ☐ Bill me later
- ☐ Charge to VISA/MC_____ *Exp* _____

TO: ***(Please print clearly)***

Name _____

Address _____

City _____ *State*_____ *ZIP*_____

Charge card orders may be called in (206) 475-2219 9 am to 4 pm weekdays, Pacific time.

```
2030 PRINT "Sorting the Data"
2040 GOSUB 2200
2050 IDX = TRUE
2060 RETURN
2100 REM --- Index the data
2110 FOR I=1 TO 10:IX(I)=I:NEXT I
2120 RETURN
2200 REM --- Sort the data
2210 N = 10:DF = 10
2220 IF DF = 1 THEN RETURN
2230    DF = INT(DF/2)
2235    SWP = FALSE
2240    FOR I=1 TO N - DF
2250       IF LN$(IX(I)) > LN$(IX(I+DF)) THEN GOSUB 2300:SWP = TRUE
2260    NEXT I
2270    IF SWP THEN 2235 ELSE 2220
2300 REM --- Swap the index pointers
2310 T = IX(I):IX(I) = IX(I+DF):IX(I+DF) = T
2320 RETURN
3000 REM --- Print Sample Data
3010 CLS:PRINT"Print Sample Data":PRINT:PRINT
3020 FOR I=1 TO 10
3030    PRINT"Item (";I;") = ";LN$(I)
3040 NEXT I
3050 PRINT"Press ENTER/RETURN to continue"
3060 LINE INPUT IN$
3070 RETURN
4000 REM --- Print Indexed Sample Data
4010 CLS:PRINT"Print Sample Data":PRINT:PRINT
4015 IF NOT IDX THEN PRINT"Data isn't indexed yet":GOTO 4050
4020 FOR I=1 TO 10
4030    PRINT"Item (";I;") = ";LN$(IX(I))
4040 NEXT I
4050 PRINT"Press ENTER/RETURN to continue"
4060 LINE INPUT IN$
4070 RETURN
```

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

CodeWorks

Issue 14 Nov/Dec 1987

CONTENTS

CODEWORKS

Issue 14 **Nov/Dec 1987**

Editor/Publisher
Irv Schmidt
Associate Editor
Terry R. Dettmann
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Advisor
Al Mashburn

CodeWorks, 3838 S. Warner St. Tacoma, WA 98409

Telephones
(206) 475-2219 (voice)
(206) 475-2356 (modem download)

300/1200 baud, 8 bits, no parity and 1 stop bit

Authors: We constantly seek material from contributors. Send your material (double spaced, upper/lower case) and allow 4 to 6 weeks for editorial review. You may send IBM-PC compatible diskettes (please save your programs in ASCII format.) Also send a hard copy listing of the program and article. Media will be returned if return postage is provided. Compensation will be made for works which are accepted for publication. **CodeWorks** pays upon acceptance rather than on publication.

Subscription price: $24.95 per year (six issues.) A subscription year runs from Nov/Dec through Sep/Oct. Anyone subscribing during the current subscription year receives all issues for that year. Not available outside the United States Zip codes. **VISA and Master Card orders are accepted by mail or phone (206) 475-2219. Charge card orders may also be left via our on-line download system (206) 475-2356.**

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in the United States of America. Bulk rate postage is paid at Tacoma, Washington.

Sample Copies: If you have a friend who would like to see a copy of CodeWorks, just send the name and address and we will send a sample copy at no cost.

Editor's Notes

The Mutual Fund tracker program promised in the last issue isn't here due to space considerations and the fact we are still trying to make it work on all machines. Also, since Randemo seemed to be sort of scattered, I have decided to try and fall back and regroup in this issue. If you have given up on it take another look, it has promise.

Magazines have two major aspects. One is the information they carry and the other is the vehicle that carries that information. I was appalled after seeing the last issue. The printer missed part of the border on the cover and most of the center pages were skewed. Aside from that, my layout and design on that issue had really gone to pot. I can do better than that, and you deserve better than that. So, starting with this issue I promise to pay more attention to the vehicle. Maybe a little decoration for visual impact would help it - we'll see.

The NFL people have really messed up everything this season. Maybe by the time you read this they will be back to some kind of schedule. In any case, we will put up the stats, for whatever games they decide to count, on the download by Tuesday afternoon of each week of play. We have finally decided on how to handle week three. If you are updating your stats just put all zeros for the week three scores and first downs. It throws off the average a bit, but all teams are being treated equally and we don't have to fix anything in the programs themselves. NFL87 will show the correct number of wins, but Stat87 will show one-half game too much for every team when you ask for the divisional standings.

Are you a "Power User" yet? I just love the hyped-up terms the industry is coming up with. Apparently, a power user is one who has (and uses) every piece of commercial application software and still needs more. And speaking of applications software, I just received an evaluation copy of a desktop publishing program. Not that we need it, we have a real typesetter and have no need to make excuses about our "dots per inch" resolution. The program came on no less than eleven diskettes, took over two megabytes of disk space and took four hours to install. A mouse is almost mandatory with these programs. I got one, and now I have "mouse cramp" in my right hand. Do you suppose that qualifies me as a Power User?

This issue is going to everyone on our list whether or not you have renewed your subscription. To those of you who have already renewed, thank you, we appreciate your support. If you have not renewed yet, please do so. We have the coming year reasonably well mapped out, and it looks very exciting. If you have some computing project in mind that you would like to see, let us know. It seems like more fun to work on something that was requested than to work on pure speculation. The family budget program several of you asked for is a case in point. I just couldn't warm up to the idea at first but last week, in a fit of inspiration, I started work on it and it may be something yet.

We have included a spot on our order form for gift subscriptions and diskettes, since it's getting around to that time of the year. Let us know who the recipient is and we will even send a card at the appropriate time, showing you as the giver.

As long as we are on that subject, please don't forget to tell us which computer you have, especially when you order diskettes. Bob can always look back in the files and find out, but we have no way of knowing whether or not you have finally broken down and moved into one of the more exotic new systems.

Enjoy the season, (in spite of the NFL) and let the only turkey you have to deal with be on the Thanksgiving table.

Irv

Forum

An Open Forum for Questions & Comments

Re: Issue 10, wanted to cast my vote for right-set numeric fields (in Card.Bas). It would be an interesting article to supplement a very popular program. And yes, we all should be following Terry's Random series anyway, even if space is not yet a problem. Also hope readers understand lines 2250 through 2360 would be a problem on the 1st year diskette but *not* in Issue 2 or the March 86 Sampler issue of CodeWorks.

John D Miller
Anderson, SC

Thanks John, you have just supplied me with an interesting topic for Beginning BASIC. Even though we touched on this lightly in Forum (Issue 11, page 4, letter from Howard W. Clother) a little expansion of the subject would seem to be in order.

...I have an IBM XT which I purchased used. It is a great machine, but I can't get into BASIC. If I type either BASIC or BASICA the machine just locks up and I have to reboot. That is, I have to turn the machine off, and then back on. The hot restart procedure (CTRL-ALT-DEL) does not work - all the keys are locked up... I really don't know where to turn, and hope you can point me in the right direction...

Paul Radke
Los Angeles, CA

First, we must assume that BASIC exists on the DOS diskette. If you are using a hard drive make sure that PATH is set so that it can find BASIC when you ask for it. Also does your DOS disk have an AUTOEXEC.BAT file that loads some other application program that would conflict with BASIC?

We would suggest you find a local user group and ask someone there about your problem. If you were close enough we would run over and see what the problem is, but unfortunately, there's too much distance between us.

There is also the possibility that you have a real hardware problem.

Please find my (renewal) for another year of the most enlightening computer magazine available in BASIC today. I am a "senior" senior citizen and do a great deal of traveling with my motorhome; with that and my penchant for procrastination, I do program "collecting" but little programming. However, I do study each and every one of the programs; enjoying the mental creativity as well as the practicality of same. One day I'll put some of them together and build a world beater!

Jack M. Farwell
Dowelltown, TN

The town you live in has an interesting name, and when I checked the files to see where it was I found that you also live at Quail Hollow Farm - darn, how rustic can you get? I once heard of a place on Long Island, New York called "Skunks Misery Road," none of which has anything to do with computing, but what the heck, it's interesting. And speaking of motorhomes, a reader from Sparks, Nevada recently stopped in to chat and have a cup of coffee. He was "roughing" it with his motorhome up in the woods around Mt. Rainier, Washington. While we sat and talked, he fiddled with my key ring which was laying on the desk. Sure enough, at 5 p.m. I went to close up and go home and my keys were missing. Panic! An hour and $50 later, a jiffy lock man had made a new set of keys to my pickup truck, so I could at least get home. On a slim chance, we sent our reader a letter in Sparks, and sure enough, a few days later he showed up with my keys. Luckily, he had a mail forwarding service, and was a hundred miles from here when he got our note. He, too, ran his computer in his motorhome using power from solar panels on top of the vehicle. In fact, he said the solar panels worked just fine and were manufactured right here in Seattle. 100 IRV - quit rambling and GOTO computing!

...It would be more helpful to me if you wrote a little more on running the programs. An easy example to let novices like me know I am doing it right and know it must then be something in the program (when something goes wrong). However, I do understand the limited space and I will overcome this small problem. I do enjoy your magazine lots and hope it stays around a long time for us old 8-bit people. Your personal touch is very difficult if not impossible to find these days. Please keep it up.

Judy Nolting
Bird Island, MN

(Ms. Nolting had sent in a problem listing wherein our CLS had been changed to CLEAR in every instance. This, of course, messed everything up, since CLEAR clears all variables and un-DIMensions all arrays. She is running a Heath 89 computer. The answer to her problem was to change the CLS to PRINT CHR$(12) instead of CLEAR. This problem is not uncommon among our readers, especially beginners.)

Thanks for the encouraging remarks, and yes, you are right - we should give more "how to run" information. The space in the magazine is wasted in any case if you can't get the programs to run, so why not use it wisely? And, in reference to your "8-bit" comment, may we reiterate that our programs are "bit-transparent," i.e., we don't care how many bits you have, BASIC is still BASIC and our programs are designed to run on 8, 16 and 32 bit machines which use Microsoft BASIC.

I found your program, Ticket.Bas, in Issue 13 very interesting but there is one puzzler. Numerous references are made to pressing the ESC key. I find no ESC key on my Model III (Tandy) keyboard, nor any mention in the owners manual. I tried David Lien's "Basic Handbook" and find no mention of ESC there either...

D. B. McRae
Grantsville, UT

They didn't tell you in their manuals and we failed to tell you in the last issue. You get ESCape on your Model III by pressing SHIFT and the UP ARROW together. That produces the ASCII 27, which is the escape code. While we are at it, you can press SHIFT and the DOWN ARROW together to get the CTRL function (ASCII 3). Also see this issue, in what will probably be a sidebar to Terry's Randemo article, where we repent for past sins and try to regroup and get all the details for all the machines together in one spot.

In response to Mr. Raymond Jasek's problem concerning the Randemo series (Issue 13, page 3), the problem is the Model III computer requires changes to the program listings. Most of these changes are noted in the articles but some are not sufficiently detailed. I changed all the word length variable names to their first two letters as the Model III won't accept DONE as it contains the key word "ON" and in line 60 I set TR=1. I also changed CHR$(3) to CHR$(31) in line 1831. This allows the CLEAR key to be used in place of CTRL-C and requires that data input to the /SCN file be changed from "USE CTRL-C to QUIT the program" to "USE CLEAR key to end input." This change should also work on the Model I which has no control key simulation...

Robert Hood
Bremerton, WA

See the answer to the previous letter from Mr. McRae. Our Model I gives the CTRL and ESC functions as described above.

This is in regard to the Pay.Bas program. I am using this program in a small business. This is the problem. It has a limit of 10 employees. As you add employees, N1 builds up to the limit. Now if an employee quits and one hires a new employee to take his place, you want to delete the old employee and add the new one. I can't do that because the N1 count is 10 and when I delete the old employee the N1 count doesn't go down. The only way I can then do it is to start with a new program and copy all the information for the old employees, leaving out the one that quit, and adding the new one... I sure would appreciate it if you could tell me how I can lower the count in N1 when I delete an employee.

Normal C. Buehler
Scott City, KS

Since yours is not the first such problem with Pay.Bas, we have made some minor changes to it which you can incorporate into your present program without having to re-enter all your employees. It's a one-page article and it's in this issue. Since Murphy's Laws always seem to be around we would suggest that you backup your disk first, then make the changes on a copy to make sure they are going to work. After making the changes here, we tried all afternoon to get it to hang up and it wouldn't.

In connection with finding ways to speed up Rcard.Bas (Issue 8, page 34), may I suggest you consider opening a dummy buffer and fielding the string variables you need? This will also avoid the garbage collector. The FIELD statement is an extremely powerful tool, since it can also be used to rename or re-specify the length of variables. The change in string pointers is invisible to the programmer, and that's a plus too, since many beginners have no clear notion of string pointers.

Jerry Bails
St. Clair Shores, MI

Thanks, that's an idea we need to explore in more detail. It sounds like there might be something really worthwhile there.

(Re: Mr. E. L. Higdon's query about the Leading Edge Model D in Issue 13.)

Sincerely hope that this gets to you before you do something foolish like buying something *other* than the Leading Edge Mod D!

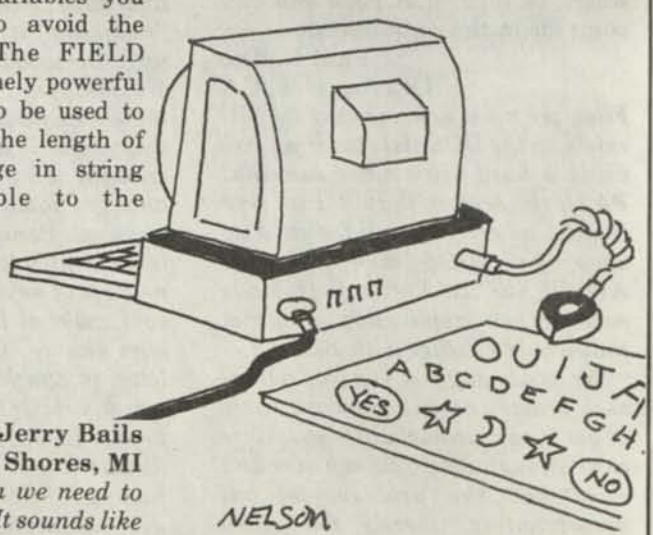
I have one through the same kind of deal you described, and am tickled purple with it. My package included a Star NX-10 printer, a bunch of disks, mucho paper, etc., for $1300. The best part of all is that with two clock speeds available it beats hell out of my friend's ATT 6300 similarly equipped at $4500, and has a much smaller footprint too...

A point that may interest you is that out here in the desert "boonies" we don't have electricity in the wall. Really, no AC power at all. I converted my Model D to run on deep-cycle batteries directly (no AC power supply) which I keep charged with photovoltaic solar panels. Works real good! Also runs the printer...

John Omohundro
Willcox, AZ

Talk about self-sufficiency! And no utility bills or power failures. Good for you!

How the computer decided to pick the NFL games this season.



Beginning Basic

ON GOTO & ON GOSUB

In this installment we are going to look at the ON GOTO, ON GOSUB statements and see what interesting things we can find out about them. Then, we'll go on to a somewhat related subject: limiting input to what you want it to be.

ON N GOTO, followed by a list of line numbers, will pick out the Nth number in that list and send program control to that line. It is interesting to note that in the expression ON N GOTO the language automatically assumes taking the integer value of N. You don't need to specify ON INT(N) GOTO; BASIC treats it that way naturally. This, as you may well have guessed, brings up a new problem. What if N is equal to 3.9 and you would like to round up anything over .5 to the next higher number? As it stands, BASIC would make a 3 out of 3.9 when it takes the integer value. The answer is an easy one. You can do simple arithmetic in the ON N GOTO statement. We can say, for example, ON N+.5 GOTO. Now, anything from 2.5 to 3.4 would be evaluated as 3; anything from 3.5 to 4.4 would become 4. In effect, you are *adjusting* the range in which INTeger works.

Since we can do arithmetic in the statement, we can take care of the problem of having so many GOTO line numbers that they won't fit on one program line. It may not be the best form to have that many in the first place, but it does happen occasionally. Let's say that we have 30 five-digit line numbers in an ON GOTO. You couldn't fit them on one program line, so here's how to do it:

10 ON N GOTO 10000, 10010, etc. to the 10th number
20 ON N-10 GOTO 11000, 11010, etc. to the 20th number
30 ON N-20 GOTO 12000, 12010, etc. to the last number.

In this example, if you entered 26 for N, the third line (line 30) would find the correct line number to go to.

While we are still on the subject of arithmetic in the ON GOTO statement, let's look at the following:

10 IF N < 0 then *linenumber*
20 IF N > 0 then *linenumber*
30 IF N = 0 then *linenumber*

We have probably all seen that many times when a test is being made for positive, negative or zero. All of the above can easily be written in one line, like this:

10 ON SGN(N)+2 GOTO *line, line, line*

Ok, what's this new thing called SGN? It's a function that determines if a number is negative, positive or zero. If N is negative it returns a minus 1, if N is positive it returns a one and if N is zero it returns a zero. Now you can see why we needed to add 2 to it in the above example. It moves -1, 0, +1 into the 1, 2, 3 range for our ON GOTO. Taking the SGN (sign) of N does not change N in any way; it simply tells what the sign of N is.

Everything we have just said about ON GOTO applies equally well to ON GOSUB.

An interesting fact about ON N GOTO and ON N GOSUB is that if N is smaller or larger than the number of items in the list of numbers to go to, *control falls through that statement to the next line.* In spite of that, you see many programs (ours too, unfortunately) that use a lumpy statement after the ON GOTO that says If N is less than the smallest number or N is larger than the largest number, etc. You don't need all that. Look at the example in Figure 1. If N is zero, or N is greater than three, control falls through to line 120 where we simply go back to line 100 and ask the question again.

Figure 1

```
100 INPUT"Enter a # from 1 to 3";N
110 ON N GOTO 130,140,150
120 GOTO 100
130 PRINT"You typed 1":END
140 PRINT"You typed 2":END
150 PRINT"You typed 3":END
160 PRINT"You should never get here."
```

One of the more annoying things about some input statements is that the cursor drops a line and prints a message, like "Redo from start" or some such when the wrong type of information is input. This can be a real pain when you have a formatted screen and the error message causes the screen to scroll up one line, taking everything out of alignment. Further, how do you limit input to numbers when you want only numbers?

In the statement: INPUT "Enter a number";N, if you enter a letter BASIC will drop down a line and print ?REDO, or on some machines, Redo from start. It may seem paradoxical, but to keep from accepting strings, you use string input. Consider the following, where the cursor is being positioned by X and Y and a GOSUB to do the positioning:

```
10 X=14:Y=1:GOSUB 260:PRINT"Enter a number";:INPUT N$
20 IF VAL(N$)=0 then 10 ELSE N=VAL(N$)
```

VAL is a function that returns the numerical value of a number that is in string form (as opposed to integer form.) If you take the VAL of a string and the string has anything in it but a number, the value returned is zero. So in line 20 of our example, if a letter was input, the VAL will be zero and we go back to where the cursor positions the input remark and simply put the remark back in the same place, in effect asking the question again. If the string we input was a number, the second part of line 20 makes an integer out of it and assigns it to variable N, and we got what we wanted - and the screen didn't scroll up a line.

All of which brings up another interesting point. In BASIC you don't need to declare your variable types; you can simply use them as they come up. This is a mixed blessing. It also causes a lot of "Type Mismatch" errors. You get that error when you try to use a string as an integer or vice versa. Most other languages force you to declare your variable types before you can use them. Worse yet, BASIC allows you to declare, say N, as a string at the beginning of the program. Then all occurrences of *any variable starting with N* will be treated as a string even without the accompanying dollar sign. Very nice, but very dangerous. You can spend days debugging such programs. The magicians among us like to use such devices to confuse the uninitiated, but we generally try to stay away from DEFSTR and the like to make more easily readable and understandable code.

It's a funny thing about programming. The more you learn about it, the shorter, more compact, (and hopefully, more elegant) your code becomes. And that's just one more kick you can get out of programming. ■

Computing Notes

We have changed Programming Notes to Computing Notes because we found we were covering more than just programming.

One reader was perplexed in reference to closing quotes in BASIC. He said sometimes we did and sometimes we didn't. If the last thing on a line of BASIC is a literal and starts with a quote, it need not be closed off with another quote. Apparently, BASIC takes the carriage return as the end of the literal string when the quote is missing.

If, however, there are more BASIC statemens following a literal string it must have closing quotes.

Is thirteen really an unlucky number? The Puzzler we published in Issue 13 has not generated even a single answer. Just in case you might be interested, the three things the defined function does is to change an integer into string form; take the absolute value of the integer (no sign); and remove the space on the left that is allocated for the sign.

Maybe Puzzlers have run their course.

When assigning a print format string (to be used later with a PRINT USING statement) you must use closing quotes. We find it difficult, when looking at programs to see how many spaces are between the quotes. So, most of the time in CodeWorks, we will add a "calibration" line above or below the line containing the string that contains spaces. See lines 250 and 260 on page 10 of this issue for an example.

This way, you can get the spacing exactly as it should be (and in the program starting on page 9 of this issue, spacing is everything.)

Pay.Bas Update

More employees and a fix

Back in March of 1986 (Issue 4) we published the program Pay.Bas. It was designed as a basic structure upon which you could build your personalized payroll program. Based on the mail and your telephone calls lately, many of you did just that.

One of the more serious problems you have pointed out to us is that you can't always get rid of an employee. (Calling him in and firing him is one thing; getting him out of the payroll program seems to be something else.) To make matters worse, when you delete an employee instead of leaving peaceably, he shows up twice in the paynames list!

The other most asked for improvement was to handle more than 10 employees without going through the entire program and fixing loop counters and such. The listing changes (below) show what needs to be done to fix both problems. We have added a new variable (NE, for number of employees) in line 165. Then we run the loops to NE instead of the 10 that was there originally. This way, you can change line 165 and all the loops will automatically behave nicely. Note that there is still no error check when you try to exceed the maximum number of employees. The program will simply not accept another employee after you have reached the maximum number. There is nothing wrong with setting NE to twice the number of employees you expect to have. It also turns out that with the program as it now stands, you must have at least one active employee in the paynames file for the program to work. Which sort of makes sense; if you have no employees there is no need to run the program.

The problem of getting rid of an employee is taken care of in the new added lines 2382 to 2388. Here, we took a cue from Terry's Stack manipulation routines in Random files and close up blank spaces in the paynames file any time there is an addition or deletion. This keeps all the valid paynames packed at the beginning of the file with all the blank spaces following. This way, the next person you add will take the first available blank space, and when you delete one from anywhere in the file, all the names following that one will move back to fill the vacant space. As in the original program, his actual record stays on the diskette until you remove it. The paynames file, if you will recall, is just a "pointer" file to the actual pay records on disk. You should be able to make these changes to your program and continue even though you already have an active payroll system running. You don't have to start all over because of these changes.

We were more than elated to see how many of you were actually using this program, and how even after almost two years, how few problems there were with it. ■

```
Added--->165 NE=15 ' sets the max number of employees you can have.
Added--->166 N1=NE
Changed->170 DIM E(18),E$(NE+1)
Changed->2130 FOR I=1 TO NE
Changed->2180 FOR I=1 TO NE
Changed->2360 FOR I=1 TO NE
Added--->2382 FOR I=1 TO NE
Added--->2384    L=I+1
Added--->2386    IF E$(I)="" THEN E$(I)=E$(L):E$(L)=""
Added--->2388 NEXT I
Changed->2940 FOR I=1 TO NE
Changed->3020 FOR I=1 TO N1
Changed->3090 FOR I=1 TO NE
Changed->3100    IF E$(I)=E$ AND NOT (E$="") THEN RETURN
```

Ecal.Bas

An Event Calendar for all occasions

Karl L. Townsend, Lansdale, Pennsylvania. By popular request, here is an event calendar program. We found it in our own files (it had been submitted serveral years ago for another publication we had.) You must have a printer capable of printing 132-character lines, even if it is in condensed mode on 8-inch paper.

An events calendar is a handy tool for organizational planning. Although these are available commercially, a customized version is an ideal project for your computer and can save (or make) money for your organization at the same time.

The events calendar program which accompanies this article (Ecal.Bas) accepts short notes (16 character limit) on coming events for a full year and prints a monthly calendar as shown in Figure 1. In addition to the scheduled events, you can customize the calendar with your own organization name, objectives or slogans. As an alternative, you can use the program to print a blank calendar without event listings to use as a worksheet or as an appointment calendar.

The type of printer is not critical other than it must have the capability of printing 132 characters per line. I am using the Okidata Microline 80, which can print 132 characters on a standard 8 1/2 inch sheet of paper using the condensed print mode. You may have to experiment with the printer commands if you have a printer other than those listed in Figure 2. *(See our notes in reference to printers at the end of this article. - Ed)*

The program is comprised of a number of independent routines accessed by menu to accomplish specific tasks such as INPUT, SAVE, PRINT, etc. I am a firm advocate of discrete routines in my programming both from the standpoint of being able to understand the program logic and for the ability to easily revise or add new functions.

Using the program

INPUT allows you to enter events into the calendar by date. A prompt appears requesting the date and the event. Enter the date in MMDD format followed by a comma. Next enter the event information limited to 16 characters (without a comma) and Enter. Your input is immediately sorted into position in the data array in date sequence and you will be asked if you have further entries. If so, enter a Y and the input prompt reappears. An N returns the program to the menu for your next selection. Note that you are limited to a maximum of three events for any given date.

Since the file used in this program tends to be rather small, a simplified Edit/Delete routine is provided. On selection of the Edit function, a prompt requests the date of the event to be edited or deleted. When the date is entered, the first record found for that date is displayed on the screen for your inspection. After verifying the correct record has been selected, press Y and Enter. You then write in your corrected entry. If it is not the record you were looking for, enter N and the program will search for another record for the selected date. If unable to find the desired record the program returns a notification that it can't be located. In the latter case, check your entry to see if you have entered the correct date. On completion of the editing of a record, the program asks if another record is to be edited. Answer Y or N as desired. On N the program returns to the menu.

To delete a record, request it by date just as if you are going to edit it. When the proper record is displayed on the screen, instead of entering a correction, simply press Enter. This nulls the record and effectively deletes it. This "deleted" record remains in the file but will not print. Since a small file is usual for a calendar function, no purge of deleted records has been provided.

FILE PRINT allows you to review the file records in rough form as stored in the file array. It lists the array to the printer in date sequence. I use this listing as a reference when editing the file.

CAL PRINT is the routine that actually prints your calendar. You are prompted for the desired month in numeric form, i.e., 7 for July, 11 for November, etc. Another prompt asks if you have loaded your events for that month, since the file for that month must have been loaded first in order to print. An N answer allows you to return to the menu to load the file, while a Y answer will commence printing. Some small delay before

printing begins is normal, as the program requires time to pull the event data from the file and format it. If the CAL PRINT routine is run with no stored events in file (*an uneventful month? -Ed*), a blank calendar is produced for the specified month. This can be used as a worksheet for initial calendar preparation.

SAVE and LOAD are standard sequential disk file operations enabling you to save your event data for later use. You can set up several different event files under separate file specifications if you wish to use the calendar program for more than one function or organization.

Several areas of the program logic deserve comment. To avoid having to re-sort the data each time new entries are made, the data is sorted into date order on initial input. Although the insertion sort used is not as efficient as many of the sort algorithms available today, for small files, it is a good choice due to the low overhead. Actually, I am using it more as a merge than a sort in this program. As each new record is entered via the input routine, it is inserted or merged into an already sorted file.

The logic of the insertion sort is extremely simple as is its implementation in BASIC. Essentially what happens is that each new record is compared against each old record in the file starting from the largest to the smallest until a record smaller than it is found. At this point the new record is inserted into the file. All records larger than the record to be inserted are moved up one place in the file to make space for the new record.

The second concept used in the calendar program is the use of pointers to the event data. Note that each time a new date and event are input, the event string is simply entered into the next available position in the DA array. At no time is this array sorted. Instead, each date has an additional piece of information appended. The second position in the two-dimensional date array serves as a pointer to the associated event data. On each input, this array position stores the current record number in the DA array where the event data is stored. No matter how often new data is sorted into the date array, this pointer can always find the event information for the related date.

As written, the program is designed to run for the year 1988. To adapt the program for other years, make the following changes:

1. Enter the desired year in line 320.
2. If the year is not a leap year, change the second pair of digits in line 2060 from 29 to 28.
3. Line 2070 will need to be changed completely. These characters indicate the day of the week for the first day of each month. For example, the first "6" in line 2070 says that January 1, 1988 is the 6th day of the week, or Friday.

There is an algorithm that will compute this data but for this program, a simple DATA line will suffice. You only have to change it once each year.

To customize the program, the strings in lines 290 to 330 should be changed to suit your particular needs. The program will automatically position these strings on the printout even if you change the number of characters. Be careful not to make them too long. Use the sample printout in Figure 1 as a general guide to maximum length. ■

We have moved the printer modes from Mr. Townsend's original program into the subroutine at line 1800. The code in lines 1810, 1870 and 1890 is for the Micronics SG-10 printer. You may need to change these to suit your particular printer. Some printers do not require that the width be set prior to entering some modes. If so, remark lines 1800 and 1880. If you have a full 132 character printer using 14 inch wide paper, the program will print a calendar 14 inches wide but the heading will be pushed to the far left. All the printer controls are in the subroutine at 1800. Also note the changes for some machines given at the end of the listing. The original program was developed under GW-BASIC. -Eds.

Main listing for Ecal.Bas for GW-BASIC
Changes for other machines follow this listing.

```
100 REM * Ecal.Bas * Event calendar by Karl L Townsend *
110 REM * CodeWorks Magazine, 3838 S. Warner St. Tacoma, WA 98409
120 REM * (206) 475-2219 voice, (206) 475-2356 modem download
130 'CLEAR 5000 ' only if you need to
140 DEFINT G-Z
150 DIM DA$(200),CO(200,2),DW$(7),DM$(12),DD$(12),DF$(12),FA(42,3)
```

```
160 RC=0
170 '
180 IF DATE$<>"" THEN DT$=DATE$ ELSE LINE INPUT "Enter today's date ";
    DT$
190 '
200 ' Read in data
210 FOR I=1 TO 7:READ DW$(I):NEXT I
220 FOR I=1 TO 12:READ DM$(I):NEXT I
230 FOR I=1 TO 12:READ DD$(I):NEXT I
240 FOR I=1 TO 12:READ DF$(I):NEXT I
250 ' 01234567890123456789012345678 9
260 A$=":\ \                    "
270 '
280 ' you can put your own info in the following lines.
290 B$="These things we do that others may live"
300 C$="Pennsylvania Wing"
310 D$="Civil Air Patrol"
320 E$="Activity - 1988 - Calendar"
330 F$="Published by Rescue Section Headquarters Squadron 3101"
340 '
350 ' print the heading and menu on the screen
360 CLS
370 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
380 PRINT"                E V E N T   C A L E N D A R
390 PRINT"              keeps track of important events
400 PRINT STRING$(60,45)
410 PRINT
420 PRINT TAB(15);"1 - Input events
430 PRINT TAB(15);"2 - Edit events
440 PRINT TAB(15);"3 - Print out an event file
450 PRINT TAB(15);"4 - Print out a calendar month
460 PRINT TAB(15);"5 - Save an event file to disk
470 PRINT TAB(15);"6 - Load an event file from disk
480 PRINT TAB(15);"7 - End session
490 PRINT
500 INPUT"Your choice ";ME
510 ON ME GOTO 540,670,870,940,1350,1470,1990
520 GOTO 360
530 '
540 ' Input routine
550 INPUT"Enter date (MMDD), Event";X1,DA$
560 IF LEN(DA$)>16 THEN DA$=LEFT$(DA$,16)
570 RC=RC+1
580 DA$(RC)=DA$
590 FOR I=RC-1 TO 0 STEP -1
600   IF X1>CO(I,1) THEN CO(I+1,1)=X1:CO(I+1,2)=RC:I=0:GOTO 630
610   CO(I+1,1)=CO(I,1)
620   CO(I+1,2)=CO(I,2)
630 NEXT I
640 INPUT"Another entry (y/n)";YN$
650 IF YN$="Y" OR YN$="y" THEN 550 ELSE 360
660 '
670 ' Edit/Delete
680 INPUT"Enter date of record (MMDD)";RD
690 FOR I=1 TO RC
```

Pennsylvania Wing
Civil Air Patrol
Activity — 1988 — Calendar

JANUARY

SUN	MON	TUE	WED	THU	FRI	SAT
					1 Happy New Year	2 Take down tree
3	4	5	6	7	8	9
10 Dad's Birthday	11	12	13	14 Order fuel Pay taxes Make dinner	15	16
17	18	19	20	21	22	23 Clean Garage
24	25	26	27 Anniversary!	28	29	30 Fix leaks
31						

These things we do that others may live

Special Notes:

Figure 1

```
700    RS=I
710    IF RD<>CO(I,1) THEN 730
720    I=RC
730 NEXT I
740 PRINT DA$(CO(RS,2))
750 INPUT"Is this the correct record? (y/n)";YN$
760 IF YN$="Y" OR YN$="y" THEN 810
770 RS=RS+1
780 IF CO(RS,1)=RD THEN 740
790 PRINT"There is no record with that MMDD."
800 GOTO 840
810 RN$=""
820 INPUT"Enter the correction ";RN$
830 DA$(CO(RS,2))=RN$
840 INPUT"Another record to edit? (y/n)";YN$
850 IF YN$="Y" OR YN$="y" THEN 680 ELSE 360
860 '
870 ' File print
880 FOR I=1 TO RC
890    LPRINT CO(I,1),CO(I,2),DA$(CO(I,2))
900 NEXT I
910 LINE INPUT"Press Enter to continue ";XX$
920 GOTO 360
930 '
940 ' Print the calendar month routine *
950 INPUT"Enter the number of the month to be printed ";MD
960 INPUT"Have you loaded your event file for this month (Y/N)";YN$
970 IF YN$="Y" OR YN$="y" THEN ELSE 360
980 GOSUB 1590
990 GOSUB 1790
1000 X=1-VAL(DF$(MD))
1010 FOR I=1 TO 6
1020    LPRINT TAB(5);STRING$(127,"-")
1030    LPRINT TAB(5);
1040    FOR J=1 TO 7
1050       X=X+1
1060       IF X>0 THEN DN$=STR$(X) ELSE DN$="    " 'three spaces
1070       IF X>VAL(DD$(MD)) THEN DN$="  " ' two spaces
1080       LPRINT USING A$;DN$;
1090       IF X>0 THEN EZ(J)=X ELSE EZ(J)=0
1100    NEXT J
1110    LPRINT":"
1120    FOR J=1 TO 5
1130       LPRINT TAB(5);
1140       FOR K=1 TO 7
1150          IF J>2 THEN ZZ=FA(EZ(K),J-2)
1160          01234567890123456789012345678901234567890123456789012345
1170          IF ZZ=0 OR J=1 OR J=2 THEN LPRINT":                    ";ELSE
              LPRINT USING ":\                    \";DA$(ZZ);
1180          01234567890123456789012345678901234567890123456789
1190       NEXT K
1200       LPRINT":"
1210    NEXT J
1220    IF X=>VAL(DD$(MD)) THEN I=6
1230 NEXT I
```

```
1240 LPRINT TAB(5) STRING$(127,"-")
1250 TB=INT(42-LEN(B$))/2
1260 LPRINT TAB(TB);B$
1270 LPRINT" ":LPRINT" ":LPRINT TAB(3) "Special Notes:"
1280 FOR I= 1 TO 10 STEP 2
1290   LPRINT TAB(3) STRING$(77,"-"):LPRINT" "
1300 NEXT I
1310 LPRINT TAB((80-LEN(F$))/2);F$
1320 FOR I=1 TO 9:LPRINT" ":NEXT I
1330 GOTO 360
1340 '
1350 ' Save to disk
1360 PRINT"Current file is ";FS$
1370 INPUT"Enter filename for save";S$
1380 IF S$="" THEN FS$=FS$ ELSE FS$=S$
1390 OPEN "O",1,FS$
1400 PRINT #1, RC, DT$
1410 FOR I=0 TO RC
1420   PRINT #1, CO(I,1),CO(I,2),",",DA$(I)
1430 NEXT I
1440 CLOSE 1
1450 GOTO 360
1460 '
1470 ' Load from disk
1480 INPUT"What filename ";FS$
1490 OPEN "I",1,FS$
1500 INPUT #1,RC,DT$
1510 PRINT FS$;" dated ";DT$;" with ";RC;" records"
1520 FOR I=0 TO RC
1530   INPUT #1,CO(I,1),CO(I,2),DA$(I)
1540 NEXT I
1550 CLOSE 1
1560 LINE INPUT"Press Enter to continue ";XX$
1570 GOTO 360
1580 '
1590 ' Format data entries
1600 PD=0
1610 FOR I=1 TO 31
1620   FOR J=1 TO 3
1630     FA(I,J)=0
1640   NEXT J
1650 NEXT I
1660 FOR I=1 TO 93
1670   IF I>RC THEN RETURN
1680   MO=INT(CO(I,1)/100)
1690   IF MO<>MD THEN 1760
1700   IF MO>MD THEN RETURN
1710   IF DA$(CO(I,2))="" THEN 1760
1720   ET=CO(I,1)-MO*100
1730   IF ET=PD THEN XD=XD-1 ELSE XD=3
1740   PD=ET
1750   FA(ET,XD)=CO(I,2)
1760 NEXT I
1770 RETURN
1780 '
```

Suggestion:

Name your event files JAN88.CAL, FEB88.CAL, etc., so you can tell what they are when you see them in your disk directory.

```
1790 ' Print header of calendar month
1800 WIDTH LPRINT 40
1810 LPRINT CHR$(27) "W" CHR$(1)  'set expanded print mode on
1820 LPRINT TAB((40-LEN(C$))/2);C$
1830 LPRINT TAB((40-LEN(D$))/2);D$
1840 LPRINT TAB((40-LEN(E$))/2);E$
1850 LPRINT TAB(5);STRING$(30,"-")
1860 LPRINT TAB((40-LEN(DM$(MD)))/2);DM$(MD)
1870 LPRINT CHR$(27) "W" CHR$(0); ' set expanded print mode off
1880 WIDTH LPRINT 132
1890 LPRINT CHR$(27) "B" CHR$(3)  ' set condensed print mode on
1900 LPRINT TAB(5); STRING$(127,"-")
1910 LPRINT TAB(5);
1920 FOR I=1 TO 7
1930 ' 012345678901234567890123456789012345678901234567890123456789
1940   LPRINT USING ":       \ \       ";DW$(I);
1950 NEXT I
1960 LPRINT":"
1970 RETURN
1980 '
1990 ' end of session
2000 ' set printer back to normal
2010 LPRINT CHR$(27) "B" CHR$(1)  ' return to normal print mode
2020 CLS:END
2030 '
2040 DATA SUN,MON,TUE,WED,THU,FRI,SAT
2050 DATA JANUARY,FEBRUARY,MARCH,APRIL,MAY,JUNE,JULY,AUGUST,SEPTEMBER,
     OCTOBER,NOVEMBER,DECEMBER
2060 DATA 31,29,31,30,31,30,31,31,30,31,30,31
2070 DATA 6,2,3,6,1,4,6,2,5,7,3,5
2080 END ' of program
```

Changes to Ecal.Bas for Tandy Models I/III

```
Changed->100 REM * Ecal/Bas * Event calendar by Karl L Townsend *
Changed->130 CLEAR 5000 ' only if you need to
Changed->260 A$=":% %                    "
Changed->1170        IF ZZ=0 OR J=1 OR J=2 THEN LPRINT":
   ";ELSE LPRINT USING ":%                    %";DA$(ZZ);
Changed->1800 'WIDTH LPRINT 40
Changed->1880 'WIDTH LPRINT 132
Changed->1940   LPRINT USING ":       % %       ";DW$(I);
```

Changes to Ecal.Bas for Tandy Model IV

```
Changed->100 REM * Ecal/Bas * Event calendar by Karl L Townsend *
Changed->130 CLEAR 5000 ' only if you need to
Changed->1800 'WIDTH LPRINT 40
Changed->1880 'WIDTH LPRINT 132
```

Event Calendar - Printer Codes

LINE	FUNCTION	STAR SG-10	MICRO LINE 80	EPSON MX 100	PAPER TIGER
1810	Expanded	ESC "W" CHR$(1)	CHR$(31)	CHR$(18)+CHR$(14)	CHR$(156)
1870	Standard	ESC "W" CHR$(0)	CHR$(30)	CHR$(20)	CHR$(157)
1890	Condensed	ESC "B" CHR$(3)	CHR$(29)	CHR$(15)	CHR$(159)
2010	Standard	ESC "B" CHR$(1)	CHR$(30)	CHR$(20)	CHR$(157)

Figure 2

```
101     1       Happy New Year
102     2       Take down tree
110     3       Dad's Birthday
114     6       Make dinner
114     5       Pay taxes
114     4       Order fuel
123     7       Clean Garage
127     8       Anniversary!
130     9       Fix leaks
```

This is the printout of the file (option 3 of the menu) that produced the calendar on page 11.

Index Update

Additions to CWindex.Dat

Card.bas, reference, issue 12, page 2
Misc, program, simple addressing, issue 12, page 3
Misc, program, select screen/printer in MS DOS, issue 12, page 4
Misc, program, swap 1st/last names, issue 12, page 4
Diff.bas, correction, swap statements in line 640, issue 12, page 5
Random files, article w/sample, issue 12, page 6
Beginning BASIC, page layout program, issue 12, page 12
Beginning BASIC, scaling organ pipes program, issue 12, page 12
Puzzler, simulate MID$ on left of equal sign, issue 12, page 16
Qtext1.bas, main program, issue 12, page 17, auto addressing to Qtext.bas
Card.bas, changes allow print of more than one label, issue 12, page 20
Amort.bas, main program, issue 12, page 21, prints amortization tables
Fract.bas, main program, issue 12, page 26, changes fractions to decimals
Typer.bas, main program, issue 12, page 28, fancy printer exerciser
Speed.bas, main program, issue 12, page 35, a tachistoscope program
Notes, break and verify on/off in MS DOS, issue 12, page 38
Notes, using the minus sign as exclusive OR operator, issue 12, page 38
Addressr.bas, correction, issue 12, page 3
Misc, POKE statement in Tandy 4 for special characters, issue 12, page 4
Poker.bas, reference, issue 12, page 5
Misc, protect programs with save P option, issue 12, page 5
Misc, tape machines lack MID$ on left of equal, issue 12, page 5
Misc, using dummy string in field statement, issue 12, page 7
Puzzler, more on Pi, issue 12, page 15
Card.bas, correction, extra lines, issue 12, page 20
Misc, if then else falls through to next line, issue 12, page 26
Misc, general purpose locate/print@ routine, issue 12, page 32
Wood.bas, reference, issue 13, page 2
Misc, conversion for MOD function, issue 13, page 3
Notes, more on minus sign as exclusive OR, issue 13, page 3
Fract.bas, correction, issue 13, page 4
Misc, mode and device timeout, issue 13, page 4
Misc, ARCHIVE and UNARC, issue 13, page 4
Misc, special characters on Tandy III/IV, issue 13, page 5
Rcard.bas, reference, speed of program, issue 13, page 5
Ticket.bas, main program, issue 13, page 7, makes show tickets
Puzzler, using defined function for STR$, issue 13, page 14
NFL87.bas, main program, issue 13, page 15, NFL for 1987-88 season
Stat87.bas, main program, issue 13, page 22, stats for NFL87.bas
Dcheck.bas, misc program, checks NFL schedules, issue 13, page 21
Beginning BASIC, find leap year program, issue 13, page 27
Beginning BASIC, find day of week program, issue 13, page 28
Beginning BASIC, find day of week/days between dates, issue 13, page 28
Menumake.bas, main program, issue 13, page 30, makes program menus
Notes, reference to line length and input #, issue 13, page 33
Notes, print@ wipes next line, correction, issue 13, page 33
Notes, MS DOS print screen and route to printer, issue 13, page 33
Random files, article w/sample index program, issue 13, page 34
Ranindex.bas, main program, issue 13, page 34, index for Randemo series.

If you are using Qkey.Bas to keep a running index of CodeWorks articles and notes, these are the changes to bring the index up to date through the last issue.

Slot.Bas

A One-armed Bandit Simulation

Staff Project. Of course, you don't like games. You have said so several times. But three games in over 80 published programs can't be too many, and besides, the kids might get a bang out of it.

Regardless of what you may think of games, programming them is always an interesting challenge. Usually, a computer game is a takeoff from some other game, so there are rules to follow. That's the case with the game we are presenting here. It simulates a slot machine (well, almost.)

Let's see what we need to make one. It should be simple, really, just assign some fruit and a bell or a bar and use the random number generator to pick one of them. Then keep track of what you got and assign some winning combinations and show a payoff of some kind or another.

But why stop there? Why not add some frills that would help the authenticity a little? Like wheels that look like they spin, and a handle that pulls down and some "sound" that's both audible and visual? Those things might just be the fun part. Shall we get started?

First, let's put our universal print@/locate subroutine in so that more than one type of machine can run it. We'll put it at the beginning of the program because it is used so often, and this way the program will always find it faster than if it were at the end somewhere. We'll put it between lines 150 and 210.

Now we have to make a decision. How are we going to make it look like wheels spinning? We could actually put some character strings on the screen and print them sequentially. Darn, they print so fast they look stationary, and we don't want that - they don't look real enough. Let's try putting three rows of number signs (#) out there and flashing them on and off. Hmmmm... still not right - makes you dizzy. Well, how about just flashing the center line of number signs for a little while and then have it stop and print the name of the fruit (or bell or bar) that happened to be picked? Now, that's not so shabby.

Actually, we can print all three wheels using the same string. It's in line 250 and we'll call it A$. Then, just for effect, we'll make a B$ made up of the letter X and we'll make it a couple of characters shorter on each end than the number signs. That's the one we will flash on and off to make it look like a spinning wheel.

What's a game without some irritating sound? No fun, that's what. So let's assign CHR$(7) to BL$ (in line 270). You know what that is, don't you? It's called BELL and is a holdover from teletype days when they would send a CHR$(7) down the line to wake up the operator at the other end to let him know something was about to come in. These days, it makes a raucous "BEEP" on most computers. Well, if that's all we have it will have to do.

Now let's see what we can do about making something that looks like the "arm" of the bandit. Keep in mind that it has to work on several different types of machine. So let's make it out of available characters, then it won't matter what machine is using it. Let's try some of the different characters to see what looks good. Hmmmm... the capital letter O looks real good, and in addition, gives the handle a "knurled" look. Talk about serendipity - some days you can't lose! What was the ASCII value for the capital O again? Oh, yes, it's ASCII 79. We can use STRING$ to build up all the different parts of the handle and then just print them when we want them. But we don't want a knurled look all the way down the handle; that wouldn't look right either. So let's use something else for the base part and the pivot of the handle. If you don't have any graphics characters available, you can use the capital letter M - it's not that nice,

but it works. With GW-BASIC we can use a graphic block whose ASCII value is 219. The earlier Tandy machines (I/III/IV) all have the same thing at ASCII 191.

Another question: Are we going to assign these strings every time we need to print the handle (which is quite often)? No, that would eat up time. If we assign the strings once, in an array, we can just print the array when we need it. We do that in lines 300 to 390, a section of code that only gets executed once per RUN of the program. But why, you may ask, does that array start at 3 and go to 15, instead of from 0 or 1 to something like most arrays? Well, it's because we are going to print the handle from row 3 through row 15 on the screen. We are using an X,Y screen location scheme, remember? So now in a loop we can say FOR X=3 to 15, assign Y, print array element L$(X) at X,Y and it puts the handle where we want it. It's almost too easy. We can even print the lever from the bottom up instead of top down (and we do, way down in line 1670, to make it look like the handle is coming back up after it was pulled down.)

Having done all that, we can clear the screen (line 410) and print some necessary information on it (lines 420 to 490). It tells all about what we get for the payoff. Next, from 500 to 540, we print the lever (like we just mentioned above). Then, from lines 570 to 590 we print the three simulated "wheels" using A$. All that has happened so far is that we have a complete display on the screen - nothing moves yet.

Line 620 positions our cursor on the prompt line at X=14, Y=1 and asks how many coins we want to start with. When we tell it, variable CU holds the number of coins used. We then do a bit of error trapping and in line 660 we go to the subroutine at 1620 to make the handle pull down. Making it pull down is simple. We just blank each line from 3 to 15 with STRING$(6,32), which is six blanks. (ASCII 32 is the space character.) Just before we went to that subroutine, we subtracted one from the number of coins because we have already used one. When we get back from the subroutine we blank out the prompt line in line 670.

Now we are going to spin our wheels for a while The J loop at line 740 goes from 1 to 3, once for each of the three wheels. Inside that loop, at line 750, we assign a different value for the Y location on the screen for each of the wheels (we have to because they each start at a different Y location.) Inside the J loop we have an I loop that is going to flash the center bar of each wheel on and off for a given amount of time. See lines 760 to 790. To make the simulation more realistic, the wheel on the left should run longer than the center wheel and the center should run longer than the right one. Serendipity strikes again! It just so happens that we have already (but for a different reason, explained above) assigned different values to the Y location on the screen. We can use that value for the "spin time" but it goes in the wrong direction, i.e., the right wheel would spin longer than the left two. To fix this, we simply start the spin loop at a number higher than Y and count backwards. See line 760, where we count from 35 to Y with a minus step, which gives us exactly what we want.

Now that we have spun the left wheel for a while the I loop exhausts itself and we fall into line 800. Here we go to the subroutine at 1130. Remember we are still in the J loop from 740 to 820, but let's go to that subroutine to see how we are going to decide what the left wheel stopped on.

At subroutine 1130 the first thing we do is seed the random number generator using DATE$. Some machines don't need to do this and they can remark this line (but don't remove it.) Having seeded the generator, we now pick a number between 1 and 5 in line 1160. Lemons are losers, and in order to lead the player on, we are going to retard their occurrence during the earlier part of the play. Sneaky, huh? Later we'll let lemons come up whenever they want to. The whole lemon thing takes place between lines 1190 and 1210. Variable A ends up being a number between 1 and 5 in any case, and the ON A GOTO in line 1220 sends us down to give a name to our number in lines 1250 through 1290. Here, we assign C$ to hold the literal name of the choice (with appropriate spaces around it), and then in the same line we "build" D$. D$ started out being a null string, and now we are going to add the string number 1 to 5 to D$ and return.

When we get back from the subroutine at 1130, we come back to line 810. There, we position the cursor at the center of the correct wheel (we are still on the left one) and then print C$ there. We then go on to the next wheel (the NEXT J in line 820 does it) and find out what the center wheel will stop on - and then go on to do the right wheel.

You can see that after all three wheels have been spun, D$ will hold three digits in string form, each digit ranging from 1 to 5. At line 840, where we are now that we have finished spinning all three wheels, we can take D$ apart to see what's in it. We put the digits from D$ into E$(1), E$(2) and E$(3). In lines 900 through 920 we can assign the proper (or no) win. Let's take line 900 first. It says that if all three numbers in D$ were the same and that the number was 1 we have the big jackpot and we

gosub 1320 to print out one hundred "klunks" and add $100 to your coins. Upon returning from the subroutine we immediately go to line 950 and try to entice the player into another pull of the lever.

If we didn't have the big jackpot (all three bars, where D$ would have been 111), program flow would automatically fall through to line 910. In 910 it says that if all three digits of D$ were the same and their value was greater than 2, we have a $10 payoff. Why greater than 2? Because the bars are value 1, the lemons are value 2, and we don't pay anything on lemons, and if they had been 1's we wouldn't even be at this line (line 900 would have caught it.) Again, we go to a subroutine (at line 1420 this time) and print the "klunks" and the noise and add the money.

The third check we make on D$ is in line 920, where we check for two plums, bells or cherries on the right. Line 920 says that if the center wheel equals the right wheel and the first wheel is not a lemon, and the value of the center and right wheels is greater than 22, then we pay off $3. The greater than 22 proviso insures that we don't pay off on two lemons on the right.

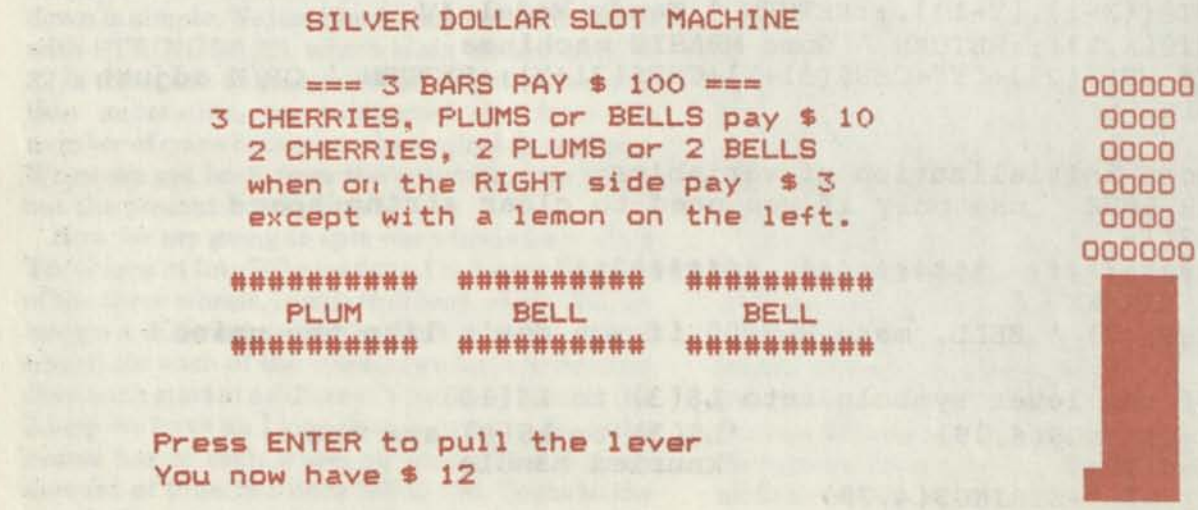
If all three of these cases fail (as they usually do) we drop right down to line 940, where we check to see if there is any money left and if there isn't we end the game. If there is, we print the prompt to pull the lever by pressing ENTER, update the screen to show how much money is left, keep track of how many times the lever was pulled and go back to repeat the entire wheel spinning process. At line 980 we gosub to 1620 to pull down the lever. This simply erases the lever display from top to bottom, making it look like the lever is being pulled towards you. Then before we return from this subroutine, we fall through to line 1680, where we let the lever come back up (print from bottom to top) and add a little delay to make it appear to come back up slowly.

It's the only slow thing about slot machines. You can lose your money rather quickly. But have fun anyway. ■

Main listing for Slot.Bas for GW-BASIC
Changes for other models follow this listing.

```
100 REM * Slot.Bas * Written for CodeWorks magazine
110 REM * 3838 S Warner St. Tacoma, WA 98409 (206) 475-2219
120 REM * placed in public domain 1987 (C)1987 80-NW Publishing Inc
130 '
140 ' General purpose locate/print@ routine. Unremark as needed.
150 GOTO 230
160 LOCATE X,Y:RETURN ' GW BASIC
170 'PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy Models I/III
180 'PRINT@((X-1),(Y-1)),;:RETURN ' Tandy Model IV
190 'PRINT@(X,Y),;:RETURN ' Some MBASIC machines
200 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN ' CP/M adjust
    to suit
210 '
220 'do some initialization of variables
230 'CLEAR 5000 ' use only if you need to clear string space
240 DIM L$(16)
250 A$="##########  ##########  ##########"
260 B$="  XXXXXX  "
270 BL$=CHR$(7) ' BELL, make BL$="" if you don't like the noise
280 '
290 'Stuff the lever symbols into L$(3) to L$(15)
300 L$(3)=STRING$(6,79)                'L$(3) to L$(8) are the
310 FOR I=4 TO 7                       'knurled handle
320   L$(I)=" "+STRING$(4,79)
330 NEXT I
```

```
340 L$(8)=STRING$(6,79)
350 FOR I=9 TO 14                    'L$(9) to L$(14) are the
360   L$(I)=" "+STRING$(4,219)       'base of the lever
370 NEXT I
380 L$(15)=STRING$(5,219)            'L$(15) is the pivot
390 '
400 'clear the screen and print the heading
410 CLS
420 PRINT TAB(9)"SILVER DOLLAR SLOT MACHINE"
430 PRINT
440 PRINT TAB(3)"      === 3 BARS PAY $ 100 ==="
450 PRINT TAB(3)" 3 CHERRIES, PLUMS or BELLS pay $ 10
460 PRINT TAB(3)"   2 CHERRIES, 2 PLUMS or 2 BELLS
470 PRINT TAB(3)"   when on the RIGHT side pay  $ 3
480 PRINT TAB(3)"   except with a lemon on the left.
490 '
500 'print the lever
510 FOR X=3 TO 15
520   Y=50:GOSUB 160
530   PRINT L$(X);
540 NEXT X
550 '
560 'print the wheels
570 X=9:Y=5:GOSUB 160:PRINT A$;
580 X=10:Y=5:GOSUB 160:PRINT A$;
590 X=11:Y=5:GOSUB 160:PRINT A$;
600 '
610 ' get started
620 X=14:Y=1:GOSUB 160:PRINT"How many coins will you start with";
630 INPUT CU
640 IF CU=<0 THEN 620 ' can't start without putting coins in
650 CU=CU-1
660 GOSUB 1620
670 X=14:Y=1:GOSUB 160:PRINT STRING$(48,32);
```



Screen dump of Slot.Bas

```
680 '
690 'start of main loop, play as long as CU is not zero
700 X=10:Y=5:GOSUB 160:PRINT A$;
710 '
720 ' spin the wheels and pick what they stop on.
730 ' note that locate Y is already defined in the J loop here.
740   FOR J=1 TO 3
750     IF J=1 THEN Y=5 ELSE IF J=2 THEN Y=17 ELSE Y=29
760     FOR I=35 TO Y STEP -1
770       X=10:GOSUB 160:PRINT B$; ' print the center bar
780       X=10:GOSUB 160:PRINT STRING$(10,32); ' blank the center bar
790     NEXT I
800     GOSUB 1130 ' to pick what it stops on
810     X=10:GOSUB 160:PRINT C$;
820   NEXT J ' end of wheel spinning loop
830 '
840 ' take D$ apart to see what is in it
850   FOR K=1 TO 3
860     E$(K)=MID$(D$,K,1)
870   NEXT K
880 '
890 ' find out if we have a win and if so, what kind.
900   IF E$(1)=E$(2) AND E$(1)=E$(3) AND VAL(E$(1))=1 THEN GOSUB 1320:
      GOTO 950
910   IF E$(1)=E$(2) AND E$(1)=E$(3) AND VAL(E$(1))>2 THEN GOSUB 1420:
      GOTO 950
920   IF E$(2)=E$(3) AND (NOT E$(1)="2") AND VAL(RIGHT$(D$,2)) >22
      THEN GOSUB 1520:GOTO 950
930 '
940   IF CU=0 THEN 1080
950   X=14:Y=1:GOSUB 160:PRINT"Press ENTER to pull the lever";
960   LINE INPUT X$
970   CU=CU-1
980   GOSUB 1620 ' to pull the lever
990   X=15:Y=1:GOSUB 160:PRINT STRING$(20,32);
1000   X=15:Y=1:GOSUB 160:PRINT"You now have $";CU
1010   D$=""
1020   X=14:Y=1:GOSUB 160:PRINT STRING$(40,32);
1030   PL=PL+1 ' keep track of number of pulls
1040 GOTO 700
1050 ' end of main loop
1060 '
1070 'end play when you have lost all your money.
1080 X=15:Y=1:GOSUB 160:PRINT"The one-armed bandit got you!
1090 PRINT"You have put $";PL+1;" into this machine.
1100 END
1110 '
1120 ' make randomizer if needed
1130 RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
1140 '
1150 ' pick a random number between 1 and 5
1160 A=INT(RND(1)*5)+1
```

```
1170 '
1180 ' retard lemons during early part of play
1190 IF PL>30 THEN PQ=1 ELSE IF PL>15 THEN PQ=2 ELSE PQ=3
1200 AA=INT(RND(1)*PQ)+1
1210 IF AA>1 THEN IF A=2 THEN 1160
1220 ON A GOTO 1250,1260,1270,1280,1290
1230 '
1240 ' D$ will be built up here to keep track of what we have
1250 C$="  ==BAR== ":D$=D$+"1":RETURN
1260 C$="   lemon  ":D$=D$+"2":RETURN
1270 C$="   PLUM   ":D$=D$+"3":RETURN
1280 C$="   BELL   ":D$=D$+"4":RETURN
1290 C$="  CHERRY  ":D$=D$+"5":RETURN
1300 '
1310 ' three bars win routine
1320 FOR JP=1 TO 100
1330   X=14:Y=17:GOSUB 160
1340   PRINT"K L U N K":PRINT BL$;
1350   X=14:Y=17:GOSUB 160:PRINT STRING$(10,32);
1360   CU=CU+1
1370   X=15:Y=1:GOSUB 160:PRINT"You now have $";CU
1380 NEXT JP
1390 RETURN
1400 '
1410 ' three of a kind win routine
1420 FOR JP=1 TO 10
1430   X=14:Y=17:GOSUB 160
1440   PRINT"K L U N K":PRINT BL$;
1450   X=14:Y=17:GOSUB 160:PRINT STRING$(10,32);
1460   CU=CU+1
1470   X=15:Y=1:GOSUB 160:PRINT"You now have $";CU
1480 NEXT JP
1490 RETURN
1500 '
1510 ' two on the right win routine
1520 FOR JP=1 TO 3
1530   X=14:Y=17:GOSUB 160
1540   PRINT"C L I N K":PRINT BL$;
1550   X=14:Y=17:GOSUB 160:PRINT STRING$(10,32);
1560   CU=CU+1
1570   X=15:Y=1:GOSUB 160:PRINT"You now have $";CU
1580 NEXT JP
1590 RETURN
1600 '
1610 ' make the lever pull down
1620 FOR X=3 TO 15
1630   Y=50:GOSUB 160
1640   PRINT STRING$(6,32);
1650 NEXT X
1660 '
```

```
1670 ' let the lever go back up slowly
1680 FOR X=15 TO 3 STEP -1
1690    Y=50:GOSUB 160
1700    PRINT L$(X);
1710    FOR TD=1 TO 50:NEXT TD ' time delay
1720 NEXT X
1730 RETURN
```

Changes to Slot.Bas for Tandy Models I/III

```
Changed->100 REM * Slot/Bas * Written for CodeWorks magazine
Changed->160 'LOCATE X,Y:RETURN ' GW BASIC
Changed->170 PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy Models I/III
Changed->230 CLEAR 5000 ' use only if you need to clear string space
Changed->360    L$(I)=" "+STRING$(4,191)    'base of lever
Changed->380 L$(15)=STRING$(5,191)          'L$(15) is the pivot
Changed->960    INPUT XX
Changed->1130 'RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->1160 A=RND(5)
Changed->1200 AA=RND(PQ)
```

Changes to Slot.Bas for Tandy Model IV

```
Changed->100 REM * Slot/Bas * Written for CodeWorks magazine
Changed->160 'LOCATE X,Y:RETURN ' GW BASIC
Changed->180 PRINT@((X-1),(Y-1)),;:RETURN ' Tandy Model IV
Changed->360    L$(I)=" "+STRING$(4,191)    'base of lever
Changed->380 L$(15)=STRING$(5,191)          'L$(15) is the pivot
Changed->960    INPUT XX
Changed->1130 'RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->1160 A=RND(5)
Changed->1200 AA=RND(PQ)
```

Random Files

A Report Generator and a summary

Terry R. Dettmann, Associate Editor. In this installment we fall back and tie the entire series together. Included here is the new report generator program, as well as the most recent version of Randemo5.Bas. If you have given up on the series, this would be an excellent place to jump back in. Randemo is now a fully functional database system, with more refinements and features to come.

The random file system now consists of three main programs: Randemo5.Bas, Ranindex.Bas and Ranprint.Bas.

Randemo5.Bas handles the input screen; allows for adding, editing and deleting records; and handles the disk file input/output. Since we have been merging files as the series progressed, and have never listed the complete Randemo5.Bas program, it is included in this issue along with change information for various machines.

Ranindex.Bas is the sort, or indexing, program for the database. It allows you to create one or more different sorts on your data and keeps these indices in files with the extension .IDX. These are used when printing out reports. The program listing for Ranindex was given in the last issue (Issue 13). Changes for various machines are given in this issue.

Ranprint.Bas (in this issue) is the report generator program for the series. It creates printed reports per your instructions (via a .PRT file which you create) using any one (or none) of the index files generated by Ranindex.Bas. Just as you can have as many index (.IDX) files as you wish; you can also have as many report format (.PRT) files as you wish.

These three programs form the basis for the entire system. Although we have refinements and new features to add, these three will remain as the foundation for the system.

Modification to Ranindex.Bas

Before going further, we need to apply a "hook" in Ranindex.Bas so that it will work with the report generator (Ranprint.Bas) program. While we are at it, we will incorporate the ability to select records as well as index them. We'll cover that in more detail later. First, get out your copy of Ranindex.Bas which was given in the last issue. There are some lines to add and one to change to make it interface properly with the report generator. The lines to add are:

```
231 INPUT"Select field number (enter 0 for none)";SX
232 IF SX=0 THEN 240
233 IF SX<1 OR SX>CX THEN PRINT"No such field number":GOTO 231
234 LINE INPUT"Select Criteria: ";SX$

255 IF SX>0 THEN IF INSTR(FP$(SX),SX$)=0 THEN 270
```

The line to change is 3060. Change it to:

```
3060  LSET XX$=MKI$(IR(IX(I)))
```

You already know from last issue that Ranindex.Bas will allow you to pick the field on which to sort. What the above code does is to further allow you to select what will be indexed. For example, you may elect to index on Zip code, and further limit the selection to people with the name "Smith" when the program asks for selection criteria. The field you index on and the field you select from need not be the same field, although they can be. Entering zero for the select field number will bypass the selection feature.

Setting it up

With that fix to Ranindex.Bas we can now go through the steps necessary to set up a database.

Let's set up a simple mail list. It will contain last name, first name, address, city, state and Zip code. We'll call it MAIL. The first thing we need to do is to decide how long each of the fields will be. Fifteen spaces should be enough for most last names, and ten spaces for first name. Note that keeping the fields as short as possible will facilitate printing reports later, but don't make them so short that you can't get the required information into them. Let's assign 20 spaces to the address field, 17 to the city field, two to the state field (all states can be abbreviated to two characters) and six spaces for the Zip code.

We now have enough information to build our input screen format file, MAIL.SCN and our map file for the random file, MAIL.MAP. Keep in mind that these two files will define the database called MAIL and need only be done once. Let's start with the MAIL.SCN file.

You can use any text editor or word processor to create this file. (Those of you with MS-DOS can even create it directly from DOS level by typing: copy con: MAIL.SCN <ENTER> and then entering the information, after which you would enter a CTRL-Z to close the file. Or, if you want to fight with it, use EDLIN.) When creating this file, do not use the TAB key; use the space bar to move the cursor instead. This is because Randemo5.Bas expects to count spaces and the TAB key won't count properly. You can be as imaginative as you like when making this file. For example, you can space down before you start, so that your input screen will start farther down the screen instead of way up on top. You can add comments and identification if you like. Whatever you enter into the MAIL.SCN file will appear on your screen. Be sure, however, to put the asterisks before the field numbers; they are important. See figure 1 for a sample layout of our MAIL.SCN file. Figure 2 shows how this file will appear when you add records with Randemo5.Bas. If you are using a word processor or text editor to create this file, be sure to save it and call it MAIL.SCN.

```
Last name:      *1
First name:     *2
Address:        *3
City:           *4
State:          *5
Zip:            *6

Use arrow keys to move up or down
Use CTRL-C to enter this record.
```

Figure 1 - MAIL.SCN

```
FIELD:1:LAST NAME:15:1
FIELD:2:FIRST NAME:10:16
FIELD:3:ADDRESS:20:26
FIELD:4:CITY:17:46
FIELD:5:STATE:2:63
FIELD:6:ZIP:6:65
```

Figure 3 - MAIL.MAP

Next, again using a text editor or word processor (or copy con:.. in MS-DOS), we need to create the MAIL.MAP file. See figure 3 to see how this file looks. Don't forget the colons between the various elements in each line; they are the delimiters that Randemo5.Bas will look for when setting up the random file fielding. In this file, we again identify our fields with the word "FIELD" in all-caps, the field name, the field length, and where it starts. It's not difficult, once you see it. For example, let's take field 1. It is 15 characters long and starts at position 1. Now going on to field 2, we find it is 10 characters long and starts at position 16. The 16 is simply the 15 plus the 1 from field 1. Field 3 is 20 characters long and starts at position 26. The 26 is

```
ADD RECORDS

Last name:      =>...............
First name:       ..........
Address:          ....................
City:             ................
State:            ..
Zip:              ......

Use arrow keys to move up or down
Use CTRL-C to enter this record.
```

Figure 2

the 10 plus the 16 from the previous field, and so on. That's all there is to the .MAP file. Be sure to save it and call it MAIL.MAP. We have now defined our input screen and the file layout in the random file for the MAIL database. You don't need to do anything with these two files again - unless you want to define an entirely new database.

Before we go into a test run of the Randemo5 program, and while we are still talking about using text editors or word processors, let's create just one of many possible print format files for the MAIL database. Figure 4 shows an example of the layout for a report that will show the date (at the position of the #D), the page number (at the #P position), a header line of dashes (at the H-------- etc. position), the column headings (HName etc.) and the field information that goes with each heading (R#1 #2 etc.). This is all followed by a footer line of dashes (F---- etc.). When laying out this report format be sure to leave enough space to contain the *field length* before starting the next field. Our first field (last name) is 15 characters long, so space over at least to the 17th position to start the next field in your report (the program adds one to each field length.) If your fields in the report overlap, it won't print the overlapping field. If you can't fit all the information you want into a single line, you can use a two-line report. See figure 5 for an example of this. Be sure to save your file and call it MAIL.PRT. You can also have MAIL1.PRT, MAIL2.PRT, for example, each with a different format and all working with the MAIL database. One of them could print in label format, another in directory format, etc.

If the report format looks familiar to some of you it's because it's the very same one we used on the Mini-Card system in Issue 8. In fact, Form.Bas from that issue can be used to create the report format files used here. If you use Form.Bas, however, you may want to change its line 1270 to: 1270 END so it won't try to call Mcard.Bas when you finish with it. Otherwise, you can use any text editor or word processor or "copy con: MAIL.PRT" (in MS-DOS) to create the report format file.

A test run

You can now run Randemo5.Bas. When it asks for FILENAME: enter MAIL. If you have looked at the listings for all three programs in the series, you will have noted that they all append the appropriate extension to file names. You can now select option 1 and start entering information into the MAIL database. During the add mode, your screen should look like figure 2 (or however you laid out your input screen in MAIL.SCN.) When you have entered as much information as you want, exit Randemo5.Bas by using menu option 0. Although you don't see it, during operation of Randemo5.Bas it creates the actual data file called MAIL.DAT and another small file called MAIL.STK. The .DAT file contains the data you entered. The .STK file is the stack file and keeps track of such things as blanks left by deleted records and where the next record you enter goes. These were covered in earlier installments and you may want to refer to them if you are hazy about what they do.

Using Ranindex.Bas

When you run Ranindex.Bas it will again ask you for FILENAME: and you enter MAIL, because that's the database we are using. Next it will ask you to tell it the name for the index file. It has to ask, because we can have many different index files. For our test run, we can simply tell it MAIL, and it will create an index file called MAIL.IDX. Next, it will ask: "Sort on what field number?" Here, you enter the number of the field you want to sort on.

The next question is: "Select on what field number (enter 0 for none)?" If you enter zero here, there will be no selection and the program will sort per the field number you gave it earlier. It will also sort every record in the file. If, however, you give it a field number to select on, it will then ask what the selection criteria is. If you were looking for all the names "Jones" and you wanted the list ordered by Zip code, you would first (in our example MAIL file) tell Ranindex.Bas to sort on field 6. Then tell it to *select* on field 1 and the selection criteria would be Jones. This would give you all the people in the file with the name "Jones" in Zip code order. You can play all sorts of variations on this theme.

Keep in mind that Ranindex.Bas will show you the list on the screen but will not print the list on the printer. It makes a file (MAIL.IDX) that contains pointers to the MAIL.DAT file in the proper order. To print, we will use Ranprint.Bas.

Using Ranprint.Bas

When you run Ranprint.Bas it will again ask for FILENAME: and again, we tell it our database name is MAIL. Next it asks for the name of the report format file, and since we have created one with the name MAIL, we answer MAIL. It then tells you that it is loading the report format file: MAIL.PRT. Next, it asks for an index name (or

none). If you simply press ENTER here or enter NONE, it will print the entire database per your report format. If you want your report per a sort or sort/selection you created with Ranindex.Bas, you tell it the name of the file (without extension) here and it will print the report per that index.

Some sample reports with fictitious names and addresses are included at the end of this article, along with the report format files which created them. From them, you should get an idea of how the Ranprint and Ranindex programs work.

You might think we're basically complete now. We can create files and print them. It's enough, isn't it? No. There is so much more we can do with these programs that we will all be having some fun for quite a while. Some of the things we can do are:

1. Expand the Ranindex program to handle more records than it can fit into memory for sorting.

2. Allow field math in the Ranprint program. This leads to wonderful things like making invoices and keeping inventory, among other things.

3. Expand our sort and search capability to multiple fields.

4. Speed up the Ranprint program to make the printing function faster.

And much more too. Let us know about features you would like to have. We'll do our best to add them. One reader is already using this system and has over 12,000 records on file, using a hard disk. ■

```
H    MAIL: sorted by Zip code  Date: #D          Page  #P
H-------------------------------------------------------
HLast name        City              State    Zip
H---------        -----             -----    ----
R#1               #4                #5       #6
```

```
    MAIL: sorted by Zip code  Date: 10-12-1987    Page   1
-------------------------------------------------------------
Last name         City              State    Zip
---------         -----             -----    ----
Peterson          Altoona           PA       17182
Larson            Charlottesville   NC       27102
Quimby            Birmingham        AL       35202
Evenson           Evansville        IN       47208
Buckingham        Fargo             ND       58001
Broncoson         Denver            CO       85103
Hawkins           Denver            CO       85201
Iverson           Denver            CO       85203
Hendricks         Garrison          CO       85405
```

```
    MAIL: sorted by Zip code  Date: 10-12-1987    Page   1
-------------------------------------------------------------
Last name         City              State    Zip
---------         -----             -----    ----
Broncoson         Denver            CO       85103
Hawkins           Denver            CO       85201
Iverson           Denver            CO       85203
Hendricks         Garrison          CO       85405
```

Figure 4

```
H   Report showing 2 lines per record              Date #D
H   Also showing footer and page number at bottom
H-------------------------------------------------------------
HLast name              First name      Address
HCity                   State           Zip
H-------------------------------------------------------------
R#1                     #2              #3
R#4                     #5              #6
F-----------------------------------------          Page #P
```

Figure 5

Figure 4 on the previous page shows (from top to bottom), the .PRT file layout, then a printout of the file in zip order without selection, and then the same .PRT file is used to print out the same list with selection on the state of Colorado, (CO). Note how you can put comments directly in the .PRT file which will print on every report.

Figure 5, above, shows how to handle the problem of not having enough space across the page. You could use a 132 character line printer, or as shown above, use an 80 column printer and print each record on two lines. The above example also shows that the date can be placed anywhere you like in the report. For that matter, so can the page number, which in this case will print at the lower right of the page.

See page 36 for a .PRT format that will print standard one-up labels. Note how you can set the page length, width and top and bottom margins.

Main listing for Randemo5.Bas for GW-BASIC
Changes for other models start on page 37.

```
10 REM - RANDEMO5.BAS - GW BASIC Random Files with Screen Control
20 REM --- Terry R. Dettmann for Codeworks Magazine
30 DIM FP$(20), SC$(24), XY(20,3)
40 DEF FNCTR$(X$)=STRING$((WD-LEN(X$))/2," ")+X$
41 DEF FNLF(X) = LOF(X)/128
50 WD=80:LN=24
51 CC=1:CX=1:CL=1
52 UP$=CHR$(5):DN$=CHR$(24):RT$=CHR$(4):LF$=CHR$(19)
53 CR$=CHR$(13):BS$=CHR$(8)
60 FALSE=0:TRUE = NOT FALSE
100 REM --- file setup
110 CLS:PRINT FNCTR$("RANDOM FILE DEMO"):PRINT:PRINT
120 LINE INPUT"FILENAME: ";FF$
125 FD$=FF$+".DAT":FS$=FF$+".STK"
130 OPEN "R",1,FD$:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
135 IF LOF(2)=0 THEN LSET SK$=MKI$(1):PUT 2,1
140 FM$=FF$+".MAP":FX$=FF$+".SCN"
150 GOSUB 5000: REM Read Map
160 GOSUB 6000: REM Read Screen
170 GOSUB 5300: REM Setup Fielding
200 REM --- main menu
210 CLS:PRINT FNCTR$("RANDOM FILE DEMO"):PRINT:PRINT
220 PRINT TAB(15)"1. Add Records to the File"
230 PRINT TAB(15)"2. Edit Records from the File"
240 PRINT TAB(15)"3. Delete Records from the File"
250 PRINT TAB(15)"4. Print the File"
260 PRINT
```

```
270 PRINT TAB(15)"0. End the Program"
280 PRINT:PRINT
290 PRINT TAB(15)"Command: ";
300 LINE INPUT CD$
310 CD = VAL(CD$)
320 IF CD<0 OR CD>4 THEN PRINT"Bad Command":GOTO 290
330 IF CD=0 THEN 500
340 ON CD GOSUB 1000,2000,3000,4000
350 GOTO 200
500 REM --- End of Program
510 CLS:PRINT"All Done":CLOSE:END
600 REM --- input a character
610 C$=INKEY$:IF C$="" THEN 610
615 IF LEN(C$)>1 THEN GOSUB 700
620 RETURN
700 REM --- look for arrows
710 C = ASC(MID$(C$,2,1))
720 IF C=72 THEN C$=UP$ ELSE IF C=77 THEN C$=RT$
730 IF C=80 THEN C$=DN$ ELSE IF C=75 THEN C$=LF$
740 RETURN
800 REM --- GOTO XY ROUTINE
810 LOCATE X,Y:RETURN
900 REM --- break line
910 FOR K=1 TO 10:BL$(K)="":NEXT K
920 JN$=IN$:NB=1
930 K = INSTR(JN$,":"):IF K=0 THEN BL$(NB)=JN$:RETURN
940     BL$(NB) = MID$(JN$,1,K-1)
950     NB = NB + 1
960     JN$ = MID$(JN$,K+1)
970 GOTO 930
1000 REM --- Add Records
1010 CLS:PRINT FNCTR$("ADD RECORDS"):PRINT:PRINT
1020 GOSUB 1600:REM Clear Buffer
1030 GOSUB 6100:REM Display Screen
1035 GOSUB 6500:REM Clear Data Fields
1040 GOSUB 1700:REM Enter Data
1050 REM
1060 X=24:Y=1:GOSUB 800
1065 RN=0:GOSUB 1200
1070 LINE INPUT"Add More (y/n)? ";YY$
1080 C$=MID$(YY$,1,1)
1090 IF C$="Y" OR C$="y" THEN 1010
1100 IF C$="N" OR C$="n" THEN RETURN
1110 GOTO 1070
1200 REM --- save current record to data base
1210 IF RN=0 THEN GOSUB 1300
1220 PUT 1,RN
1230 RETURN
1300 REM --- assign a record
1305 GOSUB 1550:IF RN<>0 THEN RETURN
1310 RN = FNLF(1) + 1:RETURN
1400 REM --- get record from data base
1410 IF RN<1 OR RN>FNLF(1) THEN RETURN
1420 GET 1,RN
1430 RETURN
1500 REM --- Stack current record number
```

```
1510 GET 2,1:SP=CVI(SK$):SP=SP+1
1520 LSET SK$=MKI$(RN):PUT 2,SP
1530 LSET SK$=MKI$(SP):PUT 2,1
1540 RETURN
1550 REM --- Get top of stack or 0
1560 GET 2,1:IF CVI(SK$)<=1 THEN RN=0:RETURN
1570 SP=CVI(SK$):GET 2,SP:RN=CVI(SK$)
1580 SP=SP-1:LSET SK$=MKI$(SP):PUT 2,1
1590 RETURN
1600 REM --- Clear Data Buffer
1610 FOR I=1 TO CX:LSET FP$(I)="":NEXT I
1620 RETURN
1650 REM --- Display Data Fields
1660 FOR I=1 TO CX:X=XY(I,1)+1:Y=XY(I,2):GOSUB 800
1670 PRINT FP$(I);
1680 NEXT I
1690 RETURN
1700 REM --- Enter Data
1710 CC=1:CL=1
1720 GOSUB 6150
1730 GOSUB 1800:IF DONE THEN RETURN
1740 GOTO 1720
1800 REM --- Enter Line of Data
1810 IN$="":DONE=FALSE
1820 GOSUB 600
1830 IF C$=BS$ THEN GOSUB 1900
1831 IF C$=CHR$(3) THEN DONE=TRUE:GOTO 1880
1832 IF C$=UP$ OR C$=RT$ THEN GOSUB 6300:GOTO 1880
1833 IF C$=DN$ THEN GOSUB 6400:GOTO 1880
1834 IF C$=CR$ THEN GOSUB 6400:GOTO 1880
1840 IF C$<" " THEN 1820
1850 IF IN$="" THEN GOSUB 1950
1860 IN$=IN$+C$:PRINT C$;:GOTO 1820
1880 IF IN$<>"" THEN LSET FP$(CL) = IN$
1890 RETURN
1900 REM --- Backspace over last character
1905 IF LEN(IN$)<1 THEN RETURN
1910 X=XY(CC,1)+1:Y=XY(CC,2)+LEN(IN$)-1
1915 GOSUB 800:PRINT ".";:GOSUB 800
1920 IN$=MID$(IN$,1,LEN(IN$)-1)
1930 RETURN
1950 REM --- Clear Data Field
1960 PRINT STRING$(LEN(FP$(CC)),".");
1970 X=XY(CC,1)+1:Y=XY(CC,2):GOSUB 800
1980 RETURN
2000 REM --- Edit Records
2010 CLS:PRINT FNCTR$("EDIT RECORDS"):PRINT:PRINT
2020 LINE INPUT"SEARCH FOR: ";SF$
2030 GOSUB 2100
2040 IF FOUND THEN GOSUB 2200 ELSE PRINT"NOT FOUND"
2045 X=LN:Y=1:GOSUB 800
2050 LINE INPUT"Edit More (y/n)? ";YY$
2060 C$=MID$(YY$,1,1)
2070 IF C$="Y" OR C$="y" THEN 2010
2080 IF C$="N" OR C$="n" THEN RETURN
```

```
2090 GOTO 2050
2100 REM --- search for record
2110 FOUND=FALSE
2120 FOR RN=1 TO FNLF(1):GOSUB 1400
2130    FOR I=1 TO CX:IF INSTR(FP$(I),SF$)<>0 THEN FOUND=TRUE:RETURN
2140    NEXT I
2150 NEXT RN
2160 RETURN
2200 REM --- edit record
2210 CLS:PRINT FNCTR$("EDIT RECORDS"):PRINT:PRINT
2220 GOSUB 6100:REM Display Screen
2230 GOSUB 1650:REM Display Data Fields
2240 GOSUB 1700:REM Enter Data
2250 GOSUB 1200:REM Save Record
2260 RETURN
2270 REM
2280 REM
2290 REM
2300 REM
2310 REM
3000 REM --- Delete Records
3010 CLS:PRINT FNCTR$("DELETE RECORDS"):PRINT:PRINT
3020 LINE INPUT"SEARCH FOR: ";SF$
3030 GOSUB 2100
3040 IF FOUND THEN GOSUB 3200 ELSE PRINT"NOT FOUND"
3045 X=LN:Y=1:GOSUB 800
3050 LINE INPUT"Delete More (y/n)? ";YY$
3060 C$=MID$(YY$,1,1)
3070 IF C$="Y" OR C$="y" THEN 3010
3080 IF C$="N" OR C$="n" THEN RETURN
3090 GOTO 3050
3100 REM --- delete the current record
3110 LSET FP$(1)="DELETED"
3120 GOSUB 1500:GOSUB 1200
3130 RETURN
3200 REM --- delete record
3210 CLS:PRINT FNCTR$("DELETE RECORDS"):PRINT:PRINT
3220 GOSUB 6100:REM Display Screen
3230 GOSUB 1650:REM Display Data Fields
3240 REM
3250 X=LN:Y=1:GOSUB 800
3260 LINE INPUT"Are you sure (y/n)? ";YY$
3270 C$=MID$(YY$,1,1)
3280 IF C$="Y" OR C$="y" THEN GOSUB 3100:RETURN
3290 IF C$="N" OR C$="n" THEN RETURN
3300 GOTO 3050
4000 REM --- Print File
4010 CLS:PRINT FNCTR$("PRINT FILE"):PRINT:PRINT
4020 FOR RN=1 TO FNLF(1):GOSUB 1400
4030    IF INSTR(FP$(1),"DELETED")<>0 THEN 4070
4040       FOR I=1 TO CX:PRINT "FIELD(";I;"): ";FP$(I)
4050       NEXT I
4060       PRINT
4070 NEXT RN
4080 RETURN
5000 REM --- read data map
```

```
5005 OPEN"I",3,FM$
5010 IF EOF(3) THEN 5035
5015    LINE INPUT#3,IN$
5020    GOSUB 900
5025    GOSUB 5100
5030 GOTO 5010
5035 CLOSE#3
5040 RETURN
5100 REM --- decode map line
5110 IF BL$(1)="FIELD" THEN GOSUB 5200:RETURN
5120 RETURN
5200 REM --- define a field
5210 NF = VAL(BL$(2)):FL = VAL(BL$(4)):FP = VAL(BL$(5))
5220 XY(NF,0)=FL:XY(NF,3)=FP
5230 RETURN
5300 REM --- Map Fields
5310 FOR I=1 TO CX
5320    NL = XY(I,3)
5330    FIELD #1, NL-1 AS X$,XY(I,0) AS FP$(I)
5340 NEXT I
5350 RETURN
6000 REM --- load screen display
6005 OPEN "I",3,FX$
6010 FOR I=1 TO LN:SC$(I)=""
6015    IF EOF(3) THEN 6030
6020       LINE INPUT#3,SC$(I):IN$=SC$(I):GOSUB 6050
6030 NEXT I
6035 CLOSE#3
6040 RETURN
6050 REM --- CHECK FOR FIELDS IN SCREEN LINE
6055 K = 0
6060 K = INSTR(K+1,IN$,"*")
6065 IF K<=0 THEN RETURN
6070 N = VAL(MID$(IN$,K+1)):XY(N,1)=I:XY(N,2)=K
6075 IF N>CX THEN CX = N
6080 GOTO 6060
6100 REM --- display screen
6110 FOR I=1 TO LN-1:X=I+1:Y=1:GOSUB 800
6115    PRINT SC$(I);
6120 NEXT I
6125 RETURN
6150 REM --- PLACE CURSOR
6155 X = XY(CL,1)+1:Y = XY(CL,2) - 2
6160 GOSUB 800:PRINT "  ";
6165 X = XY(CC,1)+1:Y = XY(CC,2) - 2
6170 GOSUB 800:PRINT "=>";
6180 RETURN
6200 REM --- interpret keystroke
6230 IF C$=UP$ OR C$=LF$ THEN GOSUB 6300
6240 IF C$=DN$ OR C$=RT$ THEN GOSUB 6400
6250 RETURN
6300 REM --- move up on page
6305 CL = CC:CC = CC - 1:IF CC<1 THEN CC = 1
6310 RETURN
6400 REM --- move down on page
6405 CL = CC:CC = CC + 1:IF CC>CX THEN CC = CX
```

```
6410 RETURN
6500 REM --- place blank fields on the screen
6510 FOR I=1 TO CX:X=XY(I,1)+1:Y=XY(I,2):GOSUB 800
6515    PRINT STRING$(XY(I,0),".");
6520 NEXT I
6525 RETURN
```

Main listing for Ranprint.Bas for GW-BASIC
Changes for other models start on page 37.

```
10 REM - RANPRINT.BAS - GW BASIC Random File Printing
20 REM --- Terry R. Dettmann for Codeworks Magazine
30 DIM FP$(20), SC$(24), XY(20,3)
40 DEF FNCTR$(X$)=STRING$((CL-LEN(X$))/2," ")+X$
41 DEF FNLF(X) = LOF(X)/128
42 DEF FNLX(X) = LOF(X)/2
50 CL=80:RW=24
51 NX=0
60 FALSE=0:TRUE = NOT FALSE
100 REM --- file setup
110 CLS:PRINT FNCTR$("RANDOM FILE REPORTS"):PRINT:PRINT
120 LINE INPUT"FILENAME: ";FF$
125 FD$=FF$+".dat":FS$=FF$+".stk"
130 OPEN "R",1,FD$:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
135 IF LOF(2)=0 THEN LSET SK$=MKI$(1):PUT 2,1
140 FM$=FF$+".MAP":FX$=FF$+".SCN"
150 GOSUB 5000: REM Read Map
170 GOSUB 5300: REM Setup Fielding
180 GOSUB 2000
185 IDX = FALSE
190 LINE INPUT"INDEX NAME (or NONE): ";FF$:IF FF$="" OR FF$="NONE"
    THEN 200
191 FI$=FF$+".IDX"
192 OPEN"R",3,FI$,2:FIELD 3,2 AS IX$
193 IDX = TRUE
200 REM --- main menu
210 CLS:PRINT FNCTR$("RANDOM FILE REPORTS"):PRINT:PRINT
220 IF IDX THEN MR=FNLX(3) ELSE MR=FNLF(1)
230 FOR RX=1 TO MR:IF IDX THEN GET#3,RX:RN=CVI(IX$) ELSE RN=RX
240    GOSUB 1400
250    IF FP$(1)="DELETED" THEN 270
260 GOSUB 2120
270 NEXT RX
280 IF LC>0 THEN GOSUB 2670
500 REM --- End of Program
510 CLS:PRINT"All Done":CLOSE:END
600 REM --- input a character
610 C$=INKEY$:IF C$="" THEN 610
615 IF LEN(C$)>1 THEN GOSUB 700
620 RETURN
700 REM --- look for arrows
710 C = ASC(MID$(C$,2,1))
720 IF C=72 THEN C$=UP$ ELSE IF C=77 THEN C$=RT$
```

```
730 IF C=80 THEN C$=DN$ ELSE IF C=75 THEN C$=LF$
740 RETURN
800 REM --- GOTO XY ROUTINE
810 LOCATE X,Y:RETURN
900 REM --- break line
910 FOR K=1 TO 10:BL$(K)="":NEXT K
920 JN$=IN$:ZB=1
930 K = INSTR(JN$,":"):IF K=0 THEN BL$(ZB)=JN$:RETURN
940     BL$(ZB) = MID$(JN$,1,K-1)
950     ZB = ZB + 1
960     JN$ = MID$(JN$,K+1)
970 GOTO 930
1400 REM --- get record from data base
1410 IF RN<1 OR RN>FNLF(1) THEN RETURN
1420 GET 1,RN
1430 RETURN
2000 REM --- Initialize Report
2010 HDR=TRUE:FTR=FALSE
2020 V=200:NF=8:ML=10
2030 PG=66:WD=80:TP=3:BT=3
2040 NT=0:NB=0:NC=0
2050 PN=0:SF=-1:SC$=""
2060 DIM HD$(ML),RC$(ML),FT$(ML)
2070 IF DATE$="" THEN INPUT"Enter the date of the report";DATE$
2080 PRINT
2090 LINE INPUT "NAME OF REPORT FORMAT FILE: ";RF$
2100 GOSUB 2780
2110 RETURN
2120 REM --- main loop
2130   IF HDR THEN GOSUB 2190
2140   GOSUB 2450
2150   IF FTR THEN GOSUB 2670
2160 RETURN
2170 REM ---
2180 '
2190 REM --- print header
2200 FOR I=1 TO TP:LPRINT" ":NEXT I
2210 PN=PN+1
2220 FOR I=0 TO NT-1
2230   LN$=HD$(I)
2240   GOSUB 2310
2250   GOSUB 2400
2260   LPRINT LN$
2270 NEXT I
2280 HDR=FALSE:LC=NT+TP
2290 RETURN
2300 '
2310 REM --- insert into header/footer lines
2320 IF INSTR(LN$,"#")<>0 AND LEN(LN$)<WD THEN LN$=LN$+STRING$(WD-
     LEN(LN$)," ")
2330 IF INSTR(LN$,"#")=0 THEN RETURN
2340   X = INSTR(LN$,"#")
2350   IF MID$(LN$,X+1,1)="D" THEN MID$(LN$,X)=DATE$
2360   IF MID$(LN$,X+1,1)="P" THEN MID$(LN$,X)=STR$(PN)
2370   IF MID$(LN$,X+1,1)="F" THEN MID$(LN$,X)=FF$
2380 GOTO 2330
```

```
2390 '
2400 REM --- strip off trailing blanks
2410 LN=LEN(LN$):IF LN=0 THEN 2430
2420 IF MID$(LN$,LN,1)=" " THEN LN$=MID$(LN$,1,LN-1):GOTO 2410
2430 RETURN
2440 '
2450 REM --- print data record
2460 FOR I=0 TO NC-1
2470   LN$=RC$(I)
2480   GOSUB 2560
2490   GOSUB 2400
2500   LPRINT LN$
2510   LC=LC+1
2520 NEXT I
2530 IF LC+NC+BT+NB>=PG THEN FTR=TRUE
2540 RETURN
2550 '
2560 REM --- put together a data line
2570 FS$=LN$
2580 IF INSTR(LN$,"#")<>0 AND LEN(LN$)<WD THEN LN$=LN$+STRING$(WD-
     LEN(LN$)," ")
2590 IF INSTR(LN$,"#")=0 THEN RETURN
2600   X = INSTR(LN$,"#")
2610   Y = VAL(MID$(LN$,X+1))
2620   MID$(LN$,X) = FP$(Y)+"   "
2630   IF X>1 THEN MID$(LN$,X-1)=" "
2640   MID$(FS$,X)=" "
2650 GOTO 2590
2660 '
2670 REM --- print footer
2680 FOR I=LC TO PG-BT-NB:LPRINT" ":NEXT I
2690 FOR I=0 TO NB-1
2700   LN$=FT$(I)
2710   GOSUB 2310
2720   GOSUB 2400
2730   LPRINT LN$
2740 NEXT I
2750 FOR I=1 TO BT:LPRINT" ":NEXT I
2760 HDR=TRUE:FTR=FALSE:LC=0:RETURN
2770 '
2780 REM --- load print file
2790 RF$=RF$+".PRT"
2800 PRINT "Loading Report Format File ";RF$
2810 OPEN "I",3,RF$
2820 IF EOF(3) THEN 2870
2830   LINE INPUT#3,LN$
2840   GOSUB 2900
2850 GOTO 2820
2860 '
2870 REM --- declare data area
2880 CLOSE#3:RETURN
2890 '
2900 REM --- decode the line
2910 REM DEBUG: PRINT LN$
2920 IF LEFT$(LN$,1)="D" THEN GOSUB 2990:RETURN
2930 IF LEFT$(LN$,1)="H" THEN GOSUB 3080:RETURN
```

```
2940 IF LEFT$(LN$,1)="R" THEN GOSUB 3120:RETURN
2950 IF LEFT$(LN$,1)="F" THEN GOSUB 3160:RETURN
2960 IF LEFT$(LN$,1)="S" THEN GOSUB 3200:RETURN
2970 RETURN
2980 '
2990 REM --- declare report parameters
3000 IF MID$(LN$,2,5)="LINES" THEN NF=VAL(MID$(LN$,8)):GOTO 3050
3010 IF MID$(LN$,2,4)="PAGE" THEN PG=VAL(MID$(LN$,7)):GOTO 3050
3020 IF MID$(LN$,2,5)="WIDTH" THEN WD=VAL(MID$(LN$,8)):GOTO 3050
3030 IF MID$(LN$,2,3)="TOP" THEN TP=VAL(MID$(LN$,6)):GOTO 3050
3040 IF MID$(LN$,2,6)="BOTTOM" THEN BT=VAL(MID$(LN$,9)):GOTO 3050
3050 REM DEBUG: PRINT NF,PG,WD,TP,BT
3060 RETURN
3070 '
3080 REM --- header line
3090 HD$(NT)=MID$(LN$,2):NT=NT+1
3100 RETURN
3110 '
3120 REM --- record line
3130 RC$(NC)=MID$(LN$,2):NC=NC+1
3140 RETURN
3150 '
3160 REM --- footer line
3170 FT$(NB)=MID$(LN$,2):NB=NB+1
3180 RETURN
3190 '
3200 REM --- selection criteria
3210 SF=VAL(MID$(LN$,2))
3220 X = INSTR(LN$,"=")
3230 IF X=0 THEN SF=-1:RETURN
3240 SC$=MID$(LN$,X+1)
3250 RETURN
5000 REM --- read data map
5001 CX = 0
5005 OPEN"I",3,FM$
5010 IF EOF(3) THEN 5035
5015    LINE INPUT#3,IN$
5020    GOSUB 900
5025    GOSUB 5100
5030 GOTO 5010
5035 CLOSE#3
5040 RETURN
5100 REM --- decode map line
5110 IF BL$(1)="FIELD" THEN GOSUB 5200:RETURN
5120 RETURN
5200 REM --- define a field
5210 NF = VAL(BL$(2)):FL = VAL(BL$(4)):FP = VAL(BL$(5))
5220 XY(NF,0)=FL:XY(NF,3)=FP
5225 CX = CX + 1
5230 RETURN
5300 REM --- Map Fields
5310 FOR I=1 TO CX
5320    NL = XY(I,3)
5330    FIELD #1, NL-1 AS X$,XY(I,0) AS FP$(I)
5340 NEXT I
5350 RETURN
```

```
DPAGE=5
DWIDTH=36
DTOP=0
DBOTTOM=3
R#2          #1
R#3
R#4                    #5    #6
```

A .PRT format for making one-up standard labels.

```
Pete          Peterson
123 Oak Street
Altoona              PA    17182
```

```
Phinneas      Quimby
1432 Larkspur Lane
Birmingham           AL    35202
```

```
Lars          Larson
56102 Vista View Dr
Charlottesville      NC    27102
```

Change lines for Tandy Models I/III

Lines to change in Randemo5.Bas

```
Changed->10 REM - RANDEMO5.BAS - Model III Random Files with Screen C
ontrol
Added--->15 CLEAR 1000
Changed->30 DIM FP$(20), SC$(16), XY(20,3)
Changed->41 DEF FNLF(X) = LOF(X)
Changed->50 WD=64:LN=16
Changed->52 UP$=CHR$(91):DN$=CHR$(10):RT$=CHR$(9):LF$=CHR$(8)
Changed->125 FD$=FF$+"/DAT":FS$=FF$+"/STK"
Changed->140 FM$=FF$+"/MAP":FX$=FF$+"/SCN"
Changed->810 PRINT@((X-1)*64)+(Y-1),;:RETURN
Changed->1060 X=16:Y=1:GOSUB 800
Changed->1730 GOSUB 1800:IF DUN THEN RETURN
Changed->1810 IN$="":DUN=FALSE
Changed->1831 IF C$=CHR$(3) THEN DUN=TRUE:GOTO 1880
```

Lines to change in Ranindex.Bas

```
Changed->10 REM - RANINDEX.BAS - Model III Random File Indexing
Added--->15 CLEAR 1000
Changed->30 DIM FP$(20), SC$(16), XY(20,3)
Changed->41 DEF FNLF(X) = LOF(X)
Changed->50 WD=64:LN=16
Changed->125 FD$=FF$+"/DAT":FS$=FF$+"/STK"
Changed->140 FM$=FF$+"/MAP":FX$=FF$+"/SCN"
Changed->215 LINE INPUT"Name of the index: ";FI$:FI$=FI$+"/IDX"
Changed->231 INPUT"Select on what field (0 for none)";SX
Changed->233 IF SX<1 OR SX>CX THEN PRINT"NO SUCH FIELD":GOTO 231
Changed->234 LINE INPUT"SELECT CRITERIA: ";SX$
Changed->255 IF SX>0 THEN IF INSTR(FP$(SX),SX$)=0 THEN 270
Changed->810 PRINT@((X-1)*64)+(Y-1),;:RETURN
```

Lines to change in Ranprint.Bas

```
Changed->10 REM - RANPRINT.BAS - Model III Random File Printing
Added--->15 CLEAR 1000
Changed->41 DEF FNLF(X) = LOF(X)
Changed->42 DEF FNLX(X) = LOF(X)
Changed->50 CL=64:RW=16
Changed->125 FD$=FF$+"/DAT":FS$=FF$+"/STK"
Changed->140 FM$=FF$+"/MAP":FX$=FF$+"/SCN"
Changed->191 FI$=FF$+"/IDX"
Changed->2790 RF$=RF$+"/PRT"
```

The Control key (CTRL) on the Models I and III is the SHIFT and DOWN ARROW keys pressed together, i.e., to get CTRL-C you hold down the shift key and the down arrow key and press C. Although not used in these programs, but for your information, the Escape (ESC) key for these machines is the SHIFT key and UP ARROW key, pressed together.

Because of the clock speed on the Model I, data input with Randemo5.Bas is rather sluggish. It would help considerably if you are using a doubler board or some other means of speeding up the clock on this machine. The speed on the Model III is considerably better than that on the Model I,

however, it is still possible to out-type the program.

On the Model III only: When you enter BASIC from DOS you will get the "How many files" question. Answer this with 3V (for three variable files) or you will get a "Bad File Mode" error when you try to run Randemo5.

On both models, the variable "DONE" was unacceptable. This is because it contains the keyword "ON." This restriction will also apply to any machine with BASIC prior to version 5.0, where keywords, even when embedded, were rejected. (After version 5.0 of BASIC, keywords may be embedded in variable names.) Consequently, for both of these machines, we have changed "DONE" to "DUN."

Both models allow the use of more than two characters as variable names - they simply ignore anything after the first two. The programs were written with this in mind, so there is no conflict with variable names.

Change lines for Tandy Model IV

Lines to change in Randemo5.Bas

```
Changed->10 REM - RANDEMO5.BAS - Model IV Random Files with Screen Co
ntrol
Changed->41 DEF FNLF(X) = LOF(X)
Changed->52 UP$=CHR$(11):DN$=CHR$(10):RT$=CHR$(9):LF$=CHR$(8)
Changed->125 FD$=FF$+"/DAT":FS$=FF$+"/STK"
Changed->140 FM$=FF$+"/MAP":FX$=FF$+"/SCN"
Changed->810 PRINT@((X-1),(Y-1)),;:RETURN
```

Lines to change in Ranindex.Bas

```
Changed->10 REM - RANINDEX.BAS - Model IV Random File Indexing
Changed->41 DEF FNLF(X) = LOF(X)
Changed->125 FD$=FF$+"/DAT":FS$=FF$+"/STK"
Changed->140 FM$=FF$+"/MAP":FX$=FF$+"/SCN"
Changed->215 LINE INPUT"Name of the index: ";FI$:FI$=FI$+"/IDX"
Changed->231 INPUT"Select on what field (enter 0 for none)";SX
Changed->233 IF SX<1 OR SX>CX THEN PRINT"NO SUCH FIELD":GOTO 231
Changed->234 LINE INPUT"SELECT CRITERIA: ";SX$
Changed->2110 SWAP IX(I),IX(I+DF)
```

Lines to change in Ranprint.Bas

```
Changed->10 REM - RANPRINT.BAS - Model IV Random File Printing
Changed->41 DEF FNLF(X) = LOF(X)
Changed->42 DEF FNLX(X) = LOF(X)
Changed->125 FD$=FF$+"/DAT":FS$=FF$+"/STK"
Changed->140 FM$=FF$+"/MAP":FX$=FF$+"/SCN"
Changed->191 FI$=FF$+"/IDX"
Changed->2790 RF$=RF$+"/PRT"
```

This machine is very much like the MS-DOS machine on which these programs were written. Unless you are a speed-demon typist, the program can easily keep up with your input.

The Model IV has a control key, but ESC is SHIFT and UP ARROW, held down together.

Like the Models I and III, the Model IV uses the slash (/) for the file extension. CP/M and MS-DOS use the period.

ORDER FORM

This is the last issue of CodeWorks you will receive *(unless you have already renewed your subscription.)*

Issues

- ☐ Renew Subscription $24.95
- ☐ NEW Subscription (starts with Nov/Dec 1987 issue)..... $24.95
- ☐ All 1st Year CodeWorks issues (Issue 1 through Issue 7) $24.95
- ☐ All 2nd year CodeWorks issues (Issue 8 through Issue 13) ... $24.95

☐ GIFT Subscription

Please give both your name and the name and address of the person who will receive the gift.

Clip or photocopy and mail to **CodeWorks,**
3838 S. Warner St.
Tacoma, WA 98409

Computer type:

Comments:

Diskettes

- ☐ 1st year programs on disk (specify type below) $20.00
- ☐ 2nd year programs on disk (specify type below) $20.00

- ☐ PC/MS-DOS 40 track DSDD
- ☐ CP/M 5 1/4 inch (specify format and computer type here _____

_____

- ☐ TRSDOS Model I 35 track SSSD
- ☐ TRSDOS Model III 40 track SSDD
- ☐ TRSDOS Model IV 40 track SSDD

HOW?

- ☐ Check/MO enclosed
- ☐ Bill me later
- ☐ Charge to VISA/MC_____ *Exp* _____

TO: ***(Please print clearly)***

Name _____

Address _____

City _____ *State*_____ *ZIP*_____

Charge card orders may be called in (206) 475-2219 9 am to 4 pm weekdays, Pacific time. 1187

Download

What's happening on the download

We haven't mentioned the download lately, probably because there haven't been that many problems with it.

Except one or two. In July, almost exactly one year to the day that it had previously failed, the computer blew a power supply. Not bad, considering that the download is up and running 24 hours per day every day of the year. It gave us the opportunity to clean the dust off the table where it sits.

In August, after three months of no rain (in the Seattle area that's a record) we got a light sprinkle which turned the streets slick. Sure enough, someone connected his car to a utility pole and knocked out our power. It didn't hurt anything, except for the car and the pole, but afterwards our modem quit answering. Or, more correctly, it answered with a carrier so weak you could hardly hear it.

Off it went to repair, while we hooked our old 300 baud only job back on. A few days later we were informed that the modem had to go back to Hayes in Georgia for repairs. So our ever resourceful technical editor, Al Mashburn, loaned us a 300/1200 Hayes to use for a while. That's what is on now, and seems to be working ok.

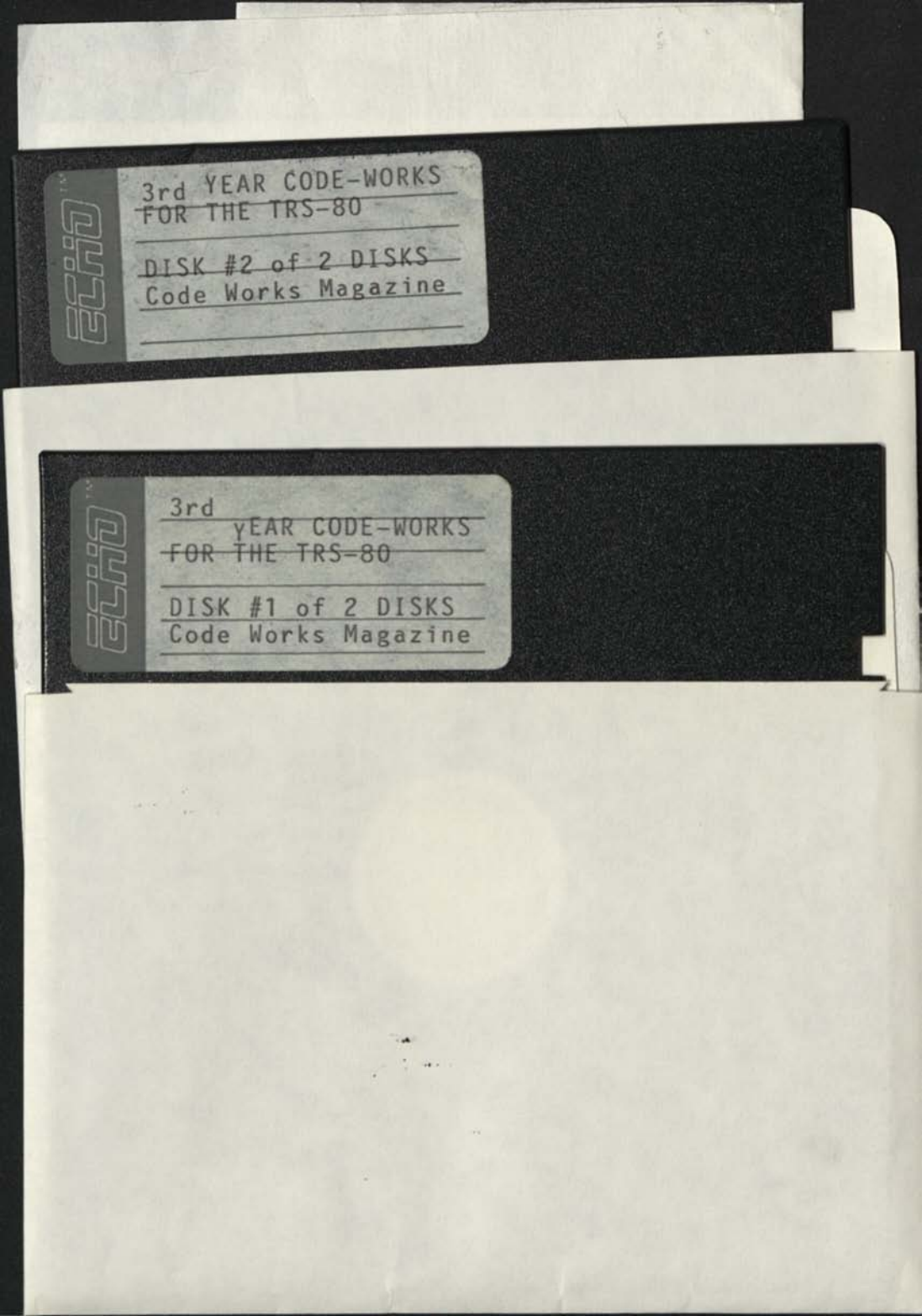
Remember the rash of problems we used to have with you not being able to get on? What happened to them? We haven't had any complaints about that for months. We would hate to think it was one of those problems that "fixed itself," since those kinds of problems always come back to haunt you in the middle of the night. Or could it be tied to that power supply failure in July? Oh, well. Let's not look a gift horse in the mouth - at least not while the horse is still alive.

In answer to several requests: Yes, you can renew your subscription via the download. Just leave your last name, subscriber number and credit card number (and exp. date) in the comments section and say "renew" or something like that. We'll pick it up from there.

It happened again. Last year when we ported NFL86 over to the download it lost line 1340 for some unknown reason. We fixed it and forgot about it. This year, we moved NFL87 over and didn't give it another thought. Sure enough, a couple of days later, reader Jerry Mallard left a message saying that line 1340 was missing. It was, and we have fixed it again, in all three versions on the download in the Issue.13a menu. We're still trying to figure out what the Xenix system doesn't like about that particular line. Maybe it's just too long for it to handle in one gulp. ■

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
US Postage
PAID
Permit No. 774
Tacoma, WA


3rd YEAR CODE-WORKS
FOR THE TRS-80
DISK #2 of 2 DISKS
Code Works Magazine
3rd
YEAR CODE-WORKS
FOR THE TRS-80
DISK #1 of 2 DISKS
Code Works Magazine

CodeWorks 3rd Year Programs for Tandy Models I/III

The diskette provided contains the following programs. The number after the file name indicates the size of the program in bytes.

We have made every effort to make these programs work on the Models I/III "as is". Ranidx.Bas is included for your information only. It will not run on the I/III. Use Ranindex.Bas instead.

Refer to the appropriate issue for programs that require initialization. Please do not remove our identification lines from the programs, they serve to show where these programs come from and will insure that we stay in business and continue to produce more good and economical software.

BIO	BAS	4785	Issue 17	A Personal biorhythm plotting program
BROKER	BAS	13121	Issue 16	An investment simulation program
CORREL	BAS	4509	Issue 19	A correlation program with lead/lag
CURSOR1	BAS	454	Issue 17	Direct cursor positioning program 1
CURSOR2	BAS	717	Issue 17	Direct cursor positioning program 2
CURSOR3	BAS	781	Issue 17	Direct cursor positioning program 3
CWINDEX	DAT	21077	All	Cumulative 3 year index of CodeWorks
DMAKER	BAS	5188	Issue 17	A decision making/helper program
EASYDATE	BAS	752	Issue 17	A standardized date input routine
ECAL	BAS	6203	Issue 14	An event calendar making program
ETAX88	BAS	7952	Issue 17	An estimated income tax program
LEDGER	BAS	9615	Issue 15	An 11-column expense ledger program
LINK	BAS	768	Issue 18	Demo program for Outline.Bas
LIST	BAS	1792	Issue 18	Demo program for Outline.Bas
LPICK	BAS	726	Issue 15	Picks Lotto numbers
LSTAT	BAS	4312	Issue 16	A lotto winner statistics program
MEDIATOR	BAS	4640	Issue 18	A dispute mediator program
NFL88	BAS	6873	Issue 19	NFL 1988-89 prediction program
OUTLINE2	BAS	5632	Issue 19	Demo program for Outline.Bas program
PUMP	BAS	2734	Issue 16	A hydraulic pump simulation program
RANDEMO5	BAS	7746	Issue 14	Randemo add/edit/delete program ver.5
RANDEMO7	BAS	8488	Issue 15	Randemo add/edit/delete progrma ver.7
RANDIST	BAS	1665	Issue 15	Distribution of random number generator
RANIDX	BAS	5544	Issue 18	Sorting for Randemo series - MS DOS
RANINDEX	BAS	3707	Issue 13	Sorting for Randemo Tandy I/III or MSDOS
RANPRINT	BAS	6016	Issue 14	Print formatting for Randemo series
RANPRNT2	BAS	6784	Issue 19	Column totals for Ranprint.Bas
REPL	BAS	3336	Issue 15	An improved Search/Replace Utility
SLOT	BAS	5498	Issue 14	A slot machine simulation program
STAT	DAT	12859	Issue 19	The statistics file for Stat88.Bas
STAT88	BAS	5801	Issue 19	Statistics keeping program for NFL88.Bas

80-NORTHWEST PUBLISHING, INC.

3838 South Warner Street, Tacoma, WA 98409 (206) 475-2219

- Technical Writing
- Publications Consultants
- Print Brokerage
- CodeWorks Magazine
- 80-NW Books

Mr. Mike Erickson
Radio Station KRJB
Box 250
Monte Rio CA 95462

Date January 6, 1989

Terms: Net 30

Invoice No. 8903

Qty	Description	Amount
3	Sets of CodeWorks Back Issues @ $24.95	$74.85
3	Sets CodeWorks Program Disks for MSDOS	$30.00
3	Sets CodeWorks Program Disks for TRS-80-III	$30.00
	Total amount due	$134.85
	Thank you very much.	

CodeWorks 3rd Year Programs for PC/MS-DOS GW-BASIC

The diskette provided contains the following programs. The number after the file name indicates the size of the program in bytes.

Refer to the appropriate issue for programs that require initialization. Please do not remove our identification lines from the programs, they serve to show where these programs come from and will insure that we stay in business and continue to produce more good and economical software.

BIO	BAS	4785	Issue 17	A Personal biorhythm plotting program
BROKER	BAS	13121	Issue 16	An investment simulation program
CORREL	BAS	4509	Issue 19	A correlation program with lead/lag
CURSOR1	BAS	454	Issue 17	Direct cursor positioning program 1
CURSOR2	BAS	717	Issue 17	Direct cursor positioning program 2
CURSOR3	BAS	781	Issue 17	Direct cursor positioning program 3
CWINDEX	DAT	21077	All	Cumulative 3 year index of CodeWorks
DMAKER	BAS	5188	Issue 17	A decision making/helper program
EASYDATE	BAS	752	Issue 17	A standardized date input routine
ECAL	BAS	6203	Issue 14	An event calendar making program
ETAX88	BAS	7952	Issue 17	An estimated income tax program
LEDGER	BAS	9615	Issue 15	An 11-column expense ledger program
LINK	BAS	768	Issue 18	Demo program for Outline.Bas
LIST	BAS	1792	Issue 18	Demo program for Outline.Bas
LPICK	BAS	726	Issue 15	Picks Lotto numbers
LSTAT	BAS	4312	Issue 16	A lotto winner statistics program
MEDIATOR	BAS	4640	Issue 18	A dispute mediator program
NFL88	BAS	6873	Issue 19	NFL 1988-89 prediction program
OUTLINE2	BAS	5632	Issue 19	Demo program for Outline.Bas program
PUMP	BAS	2734	Issue 16	A hydraulic pump simulation program
RANDEMO5	BAS	7746	Issue 14	Randemo add/edit/delete program ver.5
RANDEMO7	BAS	8488	Issue 15	Randemo add/edit/delete progrma ver.7
RANDIST	BAS	1665	Issue 15	Distribution of random number generator
RANIDX	BAS	5544	Issue 18	Sorting for Randemo series - MS DOS
RANINDEX	BAS	3707	Issue 13	Sorting for Randemo Tandy I/III or MSDOS
RANPRINT	BAS	6016	Issue 14	Print formatting for Randemo series
RANPRNT2	BAS	6784	Issue 19	Column totals for Ranprint.Bas
REPL	BAS	3336	Issue 15	An improved Search/Replace Utility
SLOT	BAS	5498	Issue 14	A slot machine simulation program
STAT	DAT	12859	Issue 19	The statistics file for Stat88.Bas
STAT88	BAS	5801	Issue 19	Statistics keeping program for NFL88.Bas

CodeWorks

Issue 15 **Jan/Feb 1988**

CONTENTS

CODEWORKS

Issue 15 Jan/Feb 1988

Editor/Publisher
Irv Schmidt
Associate Editor
Terry R. Dettmann
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Advisor
Al Mashburn

Editor's Notes

A reader recently renewed his subscription to CodeWorks but said he didn't know why, since there was such a rich assortment of applications programs available today.

Naturally, this made us sit back and take a look. Why, indeed?

There is application software available today for just about every imaginable purpose. Frankly, I don't know how I would get along without WordStar and my spelling and Thesaurus programs. But I remember the price of those programs, the learning curve (learning to jump through someone else's hoops) and the incompatibility of most such applications with others. I still use WordStar although I would rather use Microsoft Word, but the Turbo Lightning spelling checker won't work with Word. And that's the way it goes, it seems as though you can never quite get the combination you really want.

Is CodeWorks intended to replace applications software? Heck, no. Its purpose is to make entertainment out of learning how programming works and as an "idea generator." As a bonus, you get useful programs, very economically, that do something and are explained in such detail that you can create your own, personalized, versions which do exactly what you want them to. And that is where creativity comes in. We all like to point with pride to something that we have accomplished. It is always a joy to be faced with a seemingly insurmountable problem and then find the solution. Why settle for someone else's solution?

It's the same in many other fields. Take woodworking. Furniture stores are full of fine, reasonably priced, furniture. Yet woodworking magazines abound, and the craftsmen among us would rather build from scratch than buy "store bought" furniture. Several "idea generator" magazines come to mind: Mechanix Illustrated, Popular Mechanics and others all appeal to the creative instinct. Doing it better, cheaper and getting the exact results you want seem to be the draw to such publications.

In today's world of "genuine Baroque Plastic" it is refreshing to see a one-of-a-kind creation now and then.

Another aspect of the applications producers is that the crowd goes wild with every announcement of a new system and abandons the current generation of users. There are probably a few million earlier model machines out there and no one supports them, having left them to fend for themselves while they switch support to the latest announced (and yet to be seen) models. Not all of us can afford to move up to the latest model as soon as it becomes available. Some of us are still making excuses to the little lady about the expenditure for a computer that is already several years old. (That could have been our trip to Hawaii, and all you do is sit there - staring at that screen!) Such users, especially, deserve to continue to get something from their investment.

Your BASIC manual lists the name and a brief description of every tool in the arsenal. It is CodeWorks' goal to show you how to use those tools, how to mix and match them, so that you can create your own masterpiece. If we can satisfy your curiosity, whet your appetite, give you some enjoyment and entertainment along the way and give you a sense of accomplishment too, we have done our job.

Irv

Forum

An Open Forum for Questions & Comments

Even though our printer advanced our printing schedule, this issue should arrive to you at about the same time as it would have because of the Christmas holidays. We'll assume it arrives after New Years, and therefore, wish each of you a very merry Christmas and a happy New Year - belatedly. Thank you all for the kind support you have shown during the past year. By way of appreciation, we can only say that we intend to keep up our end of it and produce another fun year of recreational and educational computing.

It's early December as I write this and the renewal percentage is at about the 50 percent mark. Diskette sales have been very encouraging this year. Perhaps many of you have found that telephone connect time to our download is as, or more, expensive than ordering the diskette. We like to think that we can put those revenues to better use than the phone company can - thank you.

Week 12 of the NFL season has just passed and we are picking at just around the 60 percent mark. That may not seem to be too good, but considering the season this year, with the strike and all, it may not be too bad either. It will be interesting to finally see who goes to the Super Bowl. At this point, it's hard to make any kind of prediction.

80 Micro magazine, in their November 1987 issue, announced they would no longer support the early Tandy machines. We are guessing this covered almost half of their readers. In their January issue they suggested CodeWorks as an alternative. We already have heard from many ex-80 Micro readers and expect to hear from more as time progresses. We welcome you all and hope you find in CodeWorks the support you were denied in your previous magazine.

One thing is certain: we won't sell you short because of lack of advertising support. We don't carry advertising. Where you may previously have been just a number to be held up to some prospective advertiser, here you count as an individual whose cares and needs are our primary concern. If our programs don't work "as is" on your machine, then you will always find change lines at the end of the listing. We also use a generalized locate/print@ subroutine, where all you need do is un-remark a certain line and presto, it works for you too.

We take this opportunity to thank Eric Maloney, Michael Nadeau and the others at 80 Micro for giving us such a generous plug.

Renewal time always seems to bring about a rash of Yay's and some Boo's. We appreciate both, so here goes:

As a psychologist and sometime teacher of programming, I think your publication is the best in its field. I've tried numerous texts on BASIC programming and found them woefully inadequate. For the most part the writing is terrible, concept learning is missing, and the examples are trite and superficial. Altogether, the transfer-of-training potential offered by most BASIC programming texts and manuals that I've seen is weak at best...

CodeWorks is well written, deals effectively and at the appropriate level with concepts, and the material is interesting and germane. I've seen how the material is providing students with conceptual understanding of computer programming in general (we have Pascal ahead of us.) In addition, many examples in CodeWorks seem to transfer well to the job of getting programs written properly. I congratulate myself on having the foresight to subscribe at the very beginning. Keep up the very excellent work.

William F. Dossett
Austin, TX

Thank you, we stated in Issue 1 that our intent was to take what could be a dry subject and make it both interesting and exciting.

(This) may be the last time I will renew. $24.95 is a lot for this magazine! I get PC Resource for $19.95 a year, 12 issues!

K. D. Wentzel
Charlotte, NC

You're supposed to think quality, not quantity!

Thank you for many hours of fun and pleasure. I happen to enjoy typing in code and commend you upon the ease of reading your code and lack of confusion between some letters and numbers. It is interesting to note that there has been a program in each issue that was of concern to me. Your utility programs are much needed and to my surprise - work. Thank you for a great magazine at a bargain price!

W. A. Dahlstrom
Lincoln, NE

You said also that you had a Tandy 2000, which is interesting because we have heard from several 2000 owners who claimed our programs would not work on that computer. We knew they would, thanks for reinforcing our belief.

As a non-programmer, I enjoy reading CodeWorks. You begin on my level and, elegantly, lead me deeper into BASIC. Thank you for your fine "works."

Gene Sutherland
Chesterfield, VA

You can tell Paul Radke (Issue 14) that I too had the same problem in loading BASIC from the IBM. The reason he can't load BASIC is because it's not there! The filename "BASIC" is there but nothing else.

All he has to do is to go to an IBM or IBM compatible dealer and ask for a copy of BASIC and that's all there is to it. At least it worked for me. Apparently IBM does not tell you that the file name is there but nothing else so it just keeps on searching forever unless you reboot the system.

Marv Rozeboom
Rock Valley, IA

...I believe you have said that CodeWorks is directed toward the "average" user and therefore try not to be too simple or complicated. However, each issue has a beginning BASIC column. Since that is going to one extreme, how about also going the other direction with a small section devoted to advanced BASIC. I would be very interested to learn more about compilers and Microsoft Quick-BASIC - just a thought... P.S. I did buy the Leading Edge Model D (ref. John Omohundro's letter in Issue 14.) It is an excellent machine.

Ermon L. Higdon
Grain Valley, MO

We think that Terry's Randemo series is rather advanced, and there will be more like that. Also, in the next issue there will be an introductory article on compilers.

I need help deleting hex code string "200D0A0D0A0D0A" from an ASCII file so it can be imported to a DBase program. Has anything like this been addressed in past issues? The space taken up by the code should be deleted altogether from the file.

Charles F. Coe
Canandaigua, NY

Any word processor or text editor will read an ASCII file. That being the case, you can load your file and use it to delete the hex code string. Then save it back again, making sure you are saving it in unformatted (ASCII) format. Hex 20 is a space and 0A and 0D are line feed and carriage return, so you will have to watch for their effects in the file, not the actual code.

I have a question regarding APPEND and MERGE which has been bothering me for some time. This concerns my TRS Model III. The instructions say that to APPEND or MERGE two programs, they must be saved in ASCII format. I have quite a lot of programs, some written by me, others purchased. How can I determine whether they were saved in ASCII? They load and run without having to know. Most of them are probably in ASCII.

How can they be converted to ASCII format? Could I load and then save again but this time use SAVE"PROG",A? This would not solve the problem next month of trying to remember whether I saved in ASCII. Any suggestions? Thank you. I enjoy CodeWorks. It is the first mail I read on the day it arrives.

D. B. McRae
Grantsville, UT

There is no way to tell by looking at a directory if a program has been saved in ASCII. However, from the DOS Ready prompt on a Model III, you can type: LIST PROG/BAS (ASCII) and it will list the program for you. If you can read the program listing as though it were in BASIC, it is in ASCII format. Otherwise, you will see all sorts of hex code and what looks like missing parts in the lines. You are right about converting programs to ASCII format. Save them with the comma A, as you stated, and they will be saved in ASCII. Normally, BASIC saves in compressed binary format which, among other things, takes the keywords out and replaces them with a token character. ASCII files take up about 20 percent more disk space than compressed binary files do. Also, it takes slightly longer to save or load a file that has been saved in ASCII. The plus side of saving in ASCII is that many utilities that work on program files need the program file to be in ASCII. You can also load and make changes to an ASCII program file with a text editor or word processor; something you cannot do if the file is in compressed binary format. ASCII is almost universally understood, while compressed binary is tied to the BASIC and the machine type it is intended for. For this reason, the CodeWorks diskettes have all programs saved in ASCII format. As for knowing if programs are in ASCII or not: we have a very simple way of determining that here at CodeWorks. We save every BASIC program, without exception, in ASCII. That way, when we move a program from one machine type to another, and run into the "direct statement in file" error, we can fire up the word processor and fix it. Yet another way to tell is to save your ASCII files with a "PROG/ASC",A extension instead of /BAS extension.

...got around to playing with (your) first year diskette and found it would not boot up on my IBM PCJr. What to do?

Virginia M. Erickson
Erie, PA

The diskettes we provide do not have a system on them. We are not authorized to do that by the people who own copyright to the systems programs. To use our disks, you boot up with your own system disk, then load BASIC from your system disk. At this point you can remove your disk and insert ours and load and run the programs on it. Or, if you have two drives, put ours in the second drive and call the programs from it after having booted up with your own disk and called BASIC from the first drive.

How do I get (what switches do I set) on my printer to print graphics? I have a DMP 430 printer and a Tandy 1000...

Joseph Zabka
South Windsor, CT

We don't know what switches to set on your DMP 430. Check the manual that came with it, it should tell how to set it for different modes. Also, with the Tandy 1000 (and some other MS-DOS machines too) there is a program on the DOS disk called GRAPHICS. When you type GRAPHICS at the DOS Ready prompt, it comes up and asks what kind of printer you are using. If your printer has a graphics mode, this program will adjust things for you. There is an extremely wide variation between printers billed as graphics printers. The most popular are the Epson and IBM Graphics Printer, and many other printers emulate

them.

...In Issue 2 there is a listing for Card.Bas. I have modified that program to fill a need I have, namely for a program to keep track of my Ham Radio contacts... Here's the problem. In the program I have used a series of LPRINTs to make a formfeed every 55 lines. I am sure that there must be an easier way to do this and also I am not sure how to make a counter that will produce a formfeed every 55 lines. If you can help me out on this I would appreciate it greatly...

Norris Bundy
Alsea, OR

We thought your problem of paging when you LPRINT was universal enough to write about. Look for it in Beginning BASIC in this issue.

I have been with you guys since pretty close to the very beginning and you are getting better - hang in there. I see a lot of your programs on GEnie - that must be flattering and telling that you are doing something right. ...This item is one that I don't seem to (be able to) solve with Card.Bas or the Randemo.Bas programs. Here is what I want the printout to look like:

1. Able 3. Charlie
2. Baker 4. Delta

or, I could accept (but really don't want to):

1. Able 2. Baker
3. Charlie 4. Delta

Joe D. Lybrand
Osceola, AR

Your problem making the printer print the way you want it to is one of those nasty ones that do take a whole lot of code to get done. When you want to put names in two columns like that you have to read the entire file in first and then format it somehow. Otherwise, how would it know which name to start with at the top of the second (right hand) column? The second way, the one you don't like so much, is easier. You print the first name followed with a semicolon to keep the printhead on the same line. Then tab over and print the second name and follow that with an LPRINT command to move to the next line. This whole idea may just end up as an article in CodeWorks. Thanks for the suggestion.

I am still enjoying CodeWorks and am learning a lot. I have a quick question. We have an address program that I picked up somewhere some years ago and my daughter-in-law did a little tinkering around and it now allows me to address envelopes or print labels. I use it to keep my addresses that I mostly use for our annual Christmas letter. When I set it up I put about 50 lines, to allow for expansion, between each letter of the alphabet since I have the names entered alphabetically. The thing is, I can see that I am going to have to go through and renumber lines to accommodate more names when I add them. Is there some program I can use to renumber the lines without going through and painstakingly renumbering each line?

Jim Lopez
Pasadena, TX

If I understand your problem correctly, all you need to do is renumber the program with a big increment, like 100. Then in the spaces created, you can put many more names and addresses. The syntax for renumber is NEWLINE, STARTLINE, INCREMENT which means that if you want to renumber a program with 100 between lines and it currently starts at line 10, you would say: RENUM 10,10,100 and that should do it. Have you considered using Card.Bas to do this job?

I have a Heath H89 and added a Magnolia 10 meg. soft sector. I had two DSDD hard sector drives, which can now be used for hard or soft sector. My problem is I can run all my old hard sector disks, but can't get a printout in the programs I did before. I've checked the configurations and everything seems to be okay. The programs I have tried were HDOS or MBASIC. Also, I haven't figured out how to copy my hard sector disks to soft sector - or can't one do that? I'd really appreciate someone helping me as I live in an area where no one has a Heath computer.

Judy Nolting
550 S. 10th St.
Bird Island, MN 55310

Since we are totally unfamiliar with your problem, we are printing your complete address so that other Heath readers who may have an answer may contact you directly.

(Your) programs are great. I have converted many over to the Color Computer 3. Really like Wood.Bas, it comes in handy. On the Color Computer 3 it does your sample program in 11 seconds. Not bad for a "game machine." Keep up the good work.

Barry J. Baylis, Sr
Freeport, NY

Like we keep saying, Microsoft BASIC is Microsoft BASIC, even though you may find a couple of small differences from one version to the other.

Thank you all for the interesting input. Enjoy what's left of the holidays.

Irv

"... and he's just removing unwanted zeros."

Beginning Basic

Three reader-requested programs

Your questions and comments triggered this particular installment of Beginning BASIC. Since we wrote it there have been even more requests for a Lottery program, so we have moved that up on our priority list and are working on it.

In this installment we will deal with three different items. All three were suggested by either letters or telephone conversations. They are: right-setting a number in string form, making your printer "page" the output correctly and a little program that picks lottery numbers.

Let's take the right-set problem first. Before we get into it, however, we need to know a couple of things about two BASIC functions that complement each other. They are the STR$ and VAL functions. The STR$ function takes an integer value and changes it into string form. In string form, the value can be manipulated just like any other string, which is to say that you can take digits off the left end with LEFT$, off the right end with RIGHT$ and from the middle with MID$. That opens up a whole bunch of interesting possibilities. One of the possibilities is that we can now take a column of digits and right-set them so that they will sort correctly. You have probably already noted that numbers will not always sort properly (i.e., the number 1000 will sort ahead of the number 20 because the digits are left-set.) If we can get the values to be right-set, they will sort in the proper order because the leading spaces will sort ahead of anything else. Now, the number 20 will sort ahead of the number 1000, like it should.

But when we enter a number it is usually in integer form and we cannot control whether or not it will be right or left-set. We could, of course, enter it as a string in the first place; but even then it would still be left-set. If we are dealing with integers and want to change them to string form, we use the STR$ function to do it. If variable A is an integer and we want to change it to a string, we can say: A$=STR$(A). And, as you have probably guessed by now, if we want to change a numerical value that is in string from back to an integer, we use the VAL function: A=VAL(A$). In this case, if A$ does not contain numbers, the value returned by VAL will be zero. Not only that, but if A$ does contain numbers, but they are preceded with letters or symbols, VAL will return a zero.

Ok, so how are we going to right-set our string number? Since the number is in string form, we can use the string concatenation function to append blanks in front of it. Then we can use the RIGHT$ function to return our number with spaces in front of it. See line 60 of figure 1 for an example. In fact, if you study figure 1, you will see how it happens. In line 60 we have attached five spaces in front of A$, which in this case happens to be 10.23. Now when we take the right nine characters of this string, we get four spaces, followed by our 10.23. If you expect your numbers to get larger than nine places, append more spaces and take a larger RIGHT$. If you do this to a whole series of numbers, they will all be right-set, and if you sort them, they will sort in the correct order. And, if you later want these numbers to be integers again, use the VAL function. Yes, the VAL function will overlook those spaces we stuck in front of the number. It will even tolerate the decimal point and return the entire value (10.23 in our example case.) According to our manual, "VAL terminates its evaluation upon encountering any value that has no numeric meaning."

For a complete understanding of right-setting, type in the program in figure 1 and play with it. Try making A$ 11 characters long and see what happens. Change the number of spaces in line 60 and see what happens. The string of numbers in line 70 are used as a guide so you can see where A$ starts and ends when you run the program. Try making A$ equal to 1,234.56 and see what

happens. Also try making A$ equal to $123.45. After this, you will appreciate the power of both STR$ and VAL.

Our next item is a little sample to show how to make your printer skip to the next page after printing a given number of lines. See figure 2 for one way to do it. Basically, we want to set up a counter that gets kicked up one every time we have printed a line. Then after printing each line we check that counter to see if it has reached our desired value. If it has we first reset the counter to zero and then issue a page eject (top of form) to the printer. If it hasn't, we simply keep printing lines.

In figure 2, we initialize our line counter (NL) to zero before getting into the printing loop. That's just to make sure that NL does not have some residual value left in it from a previous operation. Then we get into our printing loop and print the value for I (it could just as well be A$(I)). Having printed I one time (one line), we increment NL by one. Then in line 50 we check to see if NL is at 55 yet. The 55 there is presumably the number of lines we want to print on the page. If NL is not 55 yet, we go on printing more lines. If it is, we set NL to zero to get it ready for the next page, and then issue an LPRINT CHR$(12), which is almost universally the printer control to page eject, or move to the top of the next form. When the loop is all done and we have printed all the lines there are to print, we issue another page eject; this one is to finish off the last partially filled page. You need to be careful of *where* in the loop you do the increment for NL, as well as where you check. Because we initialized NL at zero before getting into the loop, and because of where we increment NL and check for 55 lines, this code would actually print 55 lines on a page and then skip to the next page. One of the most common problems with this type of code is that it will print 56 lines or 54 lines instead of 55 because of how we initialize the counter and place the increment for that counter in the loop.

It should be obvious that the number 55, hard coded into line 50, could just as well be a variable which could have been set with an input prompt asking how many lines to print per page. Also, if you were printing A$(I) instead of just I, and if A$(I) in our example exceeds 80 characters in length, you may want to check for the length of A$(I) and increment NL by two instead of one in that case. This assumes you have an 80-column printer. Line 40 would then say: IF LEN(A$(I))>79 THEN NL=NL+2 ELSE NL=NL+1.

Our third goodie (see figure 3) is a little program that picks lottery numbers. At first glance, it doesn't seem like such a difficult thing to do. We thought that too, and when we tried it we got zero values and duplicate numbers. Well, the lotteries don't work like that, as you well know, so we had to find a way around those little problems.

As a matter of fact, picking random numbers and eliminating duplicates is a neat little problem all on its own. But let's start at the top of the program and work down. First, we need to know the total number of possible numbers to pick from. Some states pick six from 36, others from 40 and others from 44 or 48. We ask this question and put the value in variable H in line 120. Next, we want to know how many groups of six digits you want to see. This becomes S in line 130. In line 150 we dimension the B array at the value of H because it will most certainly be more than the default ten that BASIC allows in an array.

Now, because you want S groups of digits, we set up a loop to count from 1 to S in line 160. The next thing we need to do is seed the random number generator. We do that in line 170. Some computers don't need to do this, and for them you should remark this line. In line 190 we fill the B array with the numbers from one to H. Later we will pick numbers from this array and zero out the location where the number was so that we don't get duplicate digits.

The code from lines 210 to 250 is where we pick six numbers at random. In line 230, if the random generator has picked a number already chosen, then the B array for that location will contain a zero. If this is the case, we go back to line 220 and pick another number. When we find a number that has not been picked we zero out that position in the B array so that it won't be picked again. We then assign the number picked to the A array in line 240.

Lines 270 to 330 sort the six numbers picked into ascending order. The sort is a simple bubble sort, covered several times in earlier issues.

Finally, lines 350 to 380 print the six numbers neatly on the screen. The NEXT J in line 390 sends us back to do it all again for another group of six numbers, until the J loop is exhausted.

Will it pick winners? It will pick groups of six numbers, and is probably just as good as any other method of picking lottery numbers. And then again, it may not be. Do you suppose there is a better method of picking numbers? We'll think about that and see what comes up. ■

```
10 REM * Rset.Bas * for Beginning BASIC
20 '
30 A$="10.23"
40 PRINT"The length of A$ is now ";LEN(A$)
50 PRINT A$
60 A$=RIGHT$("      "+A$,9)
70 PRINT"1234567890"
80 PRINT A$
90 PRINT"The length of A$ is now ";LEN(A$)
```

Figure 1

```
5 REM * LC.Bas * Line counter
10 NL=0
20 FOR I=1 TO 200
30   LPRINT I
40   NL=NL+1
50   IF NL=>55 THEN NL=0:LPRINT CHR$(12)
60 NEXT I
70 LPRINT CHR$(12)
```

Figure 2

```
100 REM * Lpick.Bas * picks lottery numbers
110 CLS
120 INPUT"Pick six from how many possible numbers ";H
130 INPUT"How many sets of six do you want to see";S
140 PRINT
150 DIM B(H)
160 FOR J=1 TO S
170   RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
180 '
190   FOR X=1 TO H:B(X)=X:NEXT X
200 '
210   FOR I=1 TO 6
220     A=INT(RND(1)*H)+1 ' A=RND(H) on some machines.
230     IF B(A)=0 THEN 220 ELSE B(A)=0
240     A(I)=A
250   NEXT I
260 '
270   FOR I=1 TO 5
280     T=I+1
290     IF A(I)<A(T) THEN 320
300     Z=A(I):A(I)=A(T):A(T)=Z
310     F=1
320   NEXT I
330   IF F=1 THEN F=0:GOTO 270
340 '
350   FOR L=1 TO 6
360     PRINT USING " ###";A(L);
370   NEXT L
380   PRINT
390 NEXT J
```

Figure 3

Randist.Bas

Random distribution and a graph

Two years ago (yes, it has been that long) we did a piece on the random function. In Issue 3 we used it to determine the area under a curve. In that article, we hinted that there were other nice things we could do with random. This still isn't it, but we are finally working on a probability program using the random function.

In this article we will include a program that shows, via a graph, the distribution of your random number generator. It has two purposes: to show how to make a normal bar graph; and to give you some idea of how well your computer generates random numbers.

When you ask most programmers how many different ways there are to get data into a program they usually overlook the random number generator. It is a way of automatically generating data. On some computers, those using seed numbers, you can even get a random series that can be repeated. This sounds like something you wouldn't want from a random generator, but it is very desirable in scientific analysis and in modeling economic situations.

The program draws a nice bar graph which for our purposes works well (see figures 1 to 3.) Since the resolution of the bars is tied to the number of screen lines all data must be scaled down to either a 24 or 16 line screen. This may make this type of graph unacceptable for most other applications. On the MS-DOS machine where this program was written, we used a shaded bar of the same length next to the regular bar to give a three-dimensional effect. On the other models we just repeat the same graphics character, making the bar look twice as wide. It is interesting to note that with very little modification, you could graph two different variables, side by side on the same graph.

Picking and tabulating the random numbers takes place in the little loop between lines 190 and 230. Here, we pick a random number, print it on the screen, and at the same time (in line 220) count up how many times that number has been selected by the generator. Note that we are not putting the actual number into the B array in line 220, but are just adding one to that array position to indicate that the number was picked.

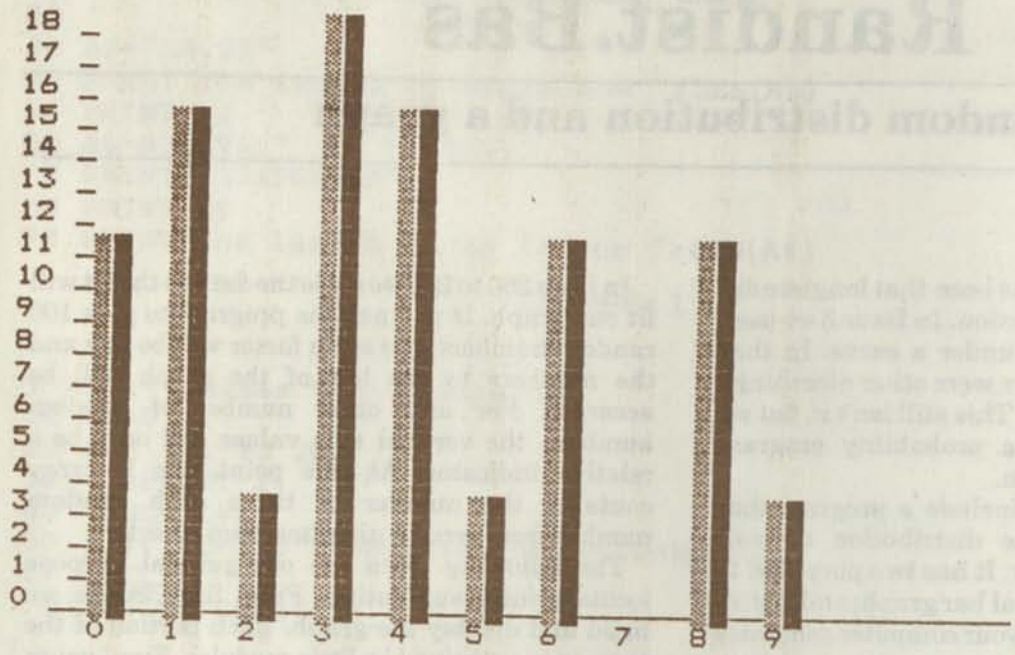
In lines 260 to 280 we scale the data so that it will fit our graph. If you ask the program to pick 100 random numbers, the scale factor will be one and the numbers to the left of the graph will be accurate. For any other number of random numbers, the vertical axis values will only be a relative indicator. At this point, the B array contains the number of times each random number from zero to nine has been selected.

The following lines are our general purpose locate/print@ subroutine. From line 380 on we build and display the graph. Each portion of the process is contained in little modules. First, using the underscore character, we draw the vertical axis of the graph (lines 400-430.) Then, in lines 450 to 480, again using the underscore character we draw the horizontal axis. The two sections from 510 to 580 and 600 to 670 put the bars on the graph. Looking at the one from 510 to 580 we see that variable I is initialized to zero. Then for the horizontal position (Y) from column six to column 50 with a step of four, we position the cursor on the Y axis. Then, counting backwards (up the screen), the loop from 540 to 560 prints the bar. The line of code at 530 simply says that if the bar is going to run off the top of the screen, to make it end at the top of the screen instead. The code from line 600 to 670 does virtually the same thing all over again, except that the horizontal position is displaced by one, making fatter (or shaded) bars.

The last two little sections of code put numbers on the axis; first the vertical and then the horizontal axis.

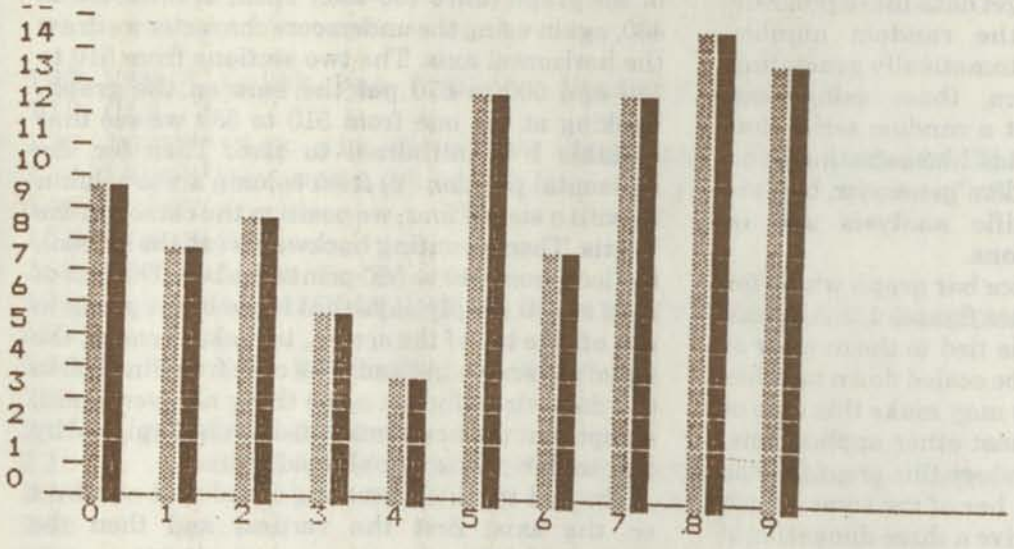
Don't expect to get identical graphs to the ones in the figures. It will probably be different every time you run it. Notice, however, that the more numbers you generate the flatter the tops of the bars become. In figure 1, where we only let it pick 25 numbers, it never picked the number seven once. The figures were dumped from the screen to a graphics printer. Line changes for other machines are given at the end of the listing.

After the graph programs we presented in earlier issues in Beginning BASIC, we thought this would be a good addition to round out the subject of graphing without the use of special graphics modes. ■



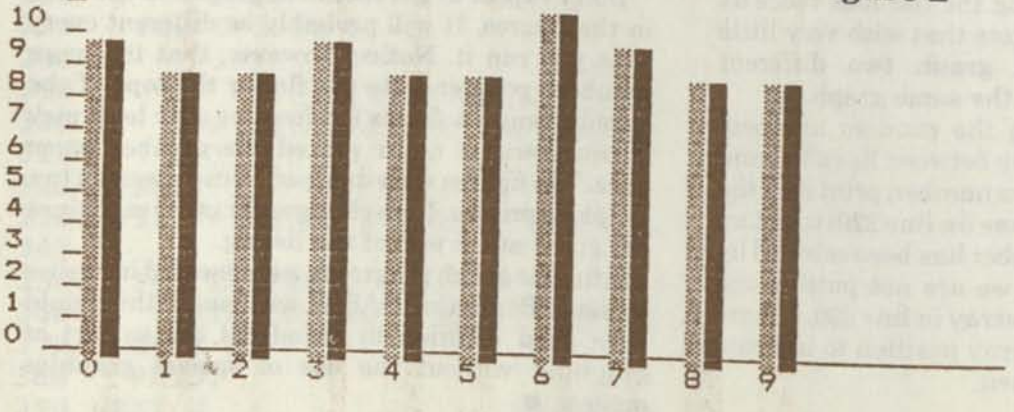
Distribution of
25 random numbers

Figure 1



Distribution of
100 random numbers

Figure 2



Distribution of
500 random numbers

Figure 3

Listing for GW-BASIC
Changes for others follow.

```
100 REM * Randist.Bas * Shows random distribution
110 '
120 RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
130 INPUT"How many random numbers do you want to generate ";R
140 IF R<20 THEN R=20
150 H=21 'set # of screen lines to use. Use 14 for 16 line screens.
160 DIM B(R)
170 '
180 'generate the random numbers
190 FOR I=1 TO R
200   A=INT(RND(1)*10)
210   PRINT A;
220   B(A)=B(A)+1
230 NEXT I
240 '
250 'scale the data
260 FOR I=0 TO 9
270   B(I)=B(I)/(R/100)
280 NEXT I
290 '
300 ' General purpose locate/print@ routine. Unremark as needed.
310 GOTO 380
320 LOCATE X,Y:RETURN ' GW BASIC
330 'PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy Models I/III
340 'PRINT@((X-1),(Y-1)),;:RETURN ' Tandy Model IV
350 'PRINT@(X,Y),;:RETURN ' Some MBASIC machines
360 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN ' CP/M
370 '
380 CLS
390 ' draw the axis
400 Y=5
410 FOR X=H TO 1 STEP -1
420   GOSUB 320:PRINT CHR$(95)
430 NEXT X
440 '
450 X=H-1
460 FOR Y=1 TO 54
470   GOSUB 320:PRINT CHR$(95)
480 NEXT Y
490 '
500 'put in the data
510 I=0
520 FOR Y=6 TO 50 STEP 4
530   IF B(I)>H-1 THEN B(I)=H-1
540   FOR X=H-1 TO H-B(I) STEP -1
550     GOSUB 320:PRINT CHR$(219)
560   NEXT X
570   I=I+1
580 NEXT Y
590 '
```

```
600 I=0
610 FOR Y=7 TO 51 STEP 4
620   IF B(I)>H-1 THEN B(I)=H-1
630   FOR X=H-1 TO H-B(I) STEP -1
640     GOSUB 320:PRINT CHR$(177)
650   NEXT X
660   I=I+1
670 NEXT Y
680 '
690 'put numbers on the axis
700 Y=1:I=0
710 FOR X=H-1 TO 1 STEP -1
720   GOSUB 320:PRINT I
730   I=I+1
740 NEXT X
750 '
760 X=H:I=0
770 FOR Y=5 TO 42 STEP 4
780   GOSUB 320:PRINT I
790   I=I+1
800 NEXT Y
810 PRINT
```

Change lines for Tandy I/III

```
Changed->100 REM * Randist/Bas * Shows random distribution
Changed->120 'RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->150 H=14 'set # of screen lines to use. Use 14 for 16 line s
creens.
Changed->200   A=RND(10)-1
Changed->320 'LOCATE X,Y:RETURN ' GW BASIC
Changed->330 PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy Models I/III
Changed->420   GOSUB 320:PRINT CHR$(95);
Changed->460 FOR Y=1 TO 43
Changed->470   GOSUB 320:PRINT CHR$(95);
Changed->520 FOR Y=6 TO 43 STEP 4
Changed->550     GOSUB 320:PRINT CHR$(191);
Changed->610 FOR Y=7 TO 43 STEP 4
Changed->640     GOSUB 320:PRINT CHR$(191);
Changed->720   GOSUB 320:PRINT I;
Changed->780   GOSUB 320:PRINT I;
```

Change lines for Tandy II/IV

```
Changed->100 REM * Randist/Bas * Shows random distribution
Changed->120 'RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->200   A=RND(10)-1
Changed->320 'LOCATE X,Y:RETURN ' GW BASIC
Changed->340 PRINT@((X-1),(Y-1)),;:RETURN ' Tandy Model II/IV
Changed->420   GOSUB 320:PRINT CHR$(95);
Changed->470   GOSUB 320:PRINT CHR$(95);
Changed->550     GOSUB 320:PRINT CHR$(191);
Changed->640     GOSUB 320:PRINT CHR$(191);
Changed->720   GOSUB 320:PRINT I;
Changed->780   GOSUB 320:PRINT I;
```

Ledger.Bas

Summarize your expenses - neatly

Staff Project. Here's a year-end project that will give you a fast and easy way to account for your expenses, with percentages and both monthly and yearly reports. It also allows you to spread one check over several expense categories.

Ledger.Bas is a program with which you can spread out your expenses into categories. It was designed to make a rather dull job a little faster and easier. For the most part, entries can all be made from the numeric keypad, so you never need to take your hands off the keyboard. The check numbers increment automatically, and if you forget what number a category is, you can simply look at the screen, where they are displayed during the input process. See figure 1.

The program allows you to take one check (a charge card payment, for example) and apply parts of it to different categories. If you enter an amount or category incorrectly, there is an edit feature which will allow you to fix it. If something you change would change the running balance also, there is another feature in the program which allows you to re-balance. This re- balance feature works on any one month or on a series of consecutive months, so if you find that you goofed in April and it is now July, you can give it the correct starting balance for April and tell it to re-balance April, May, June and July. It will.

You don't need to enter an entire month in one sitting; you can do it any time you want to. If you are in the middle of a month and want to update, the program will find the last check number you entered and the current balance and display them for you. When you start a new month, however, you will need to manually enter the starting balance and the first check number.

Any partial or complete month can be printed on your line printer. You must be able to print 132 characters per line to use this program; either in condensed mode on an 8 inch printer or on a regular 14 inch wide, 132-column printer. The printout will not only give column totals, but will figure the percentage of total expenses for each category. This report will also show your total monthly deposits and total expenses, as well as the running balance. The program is set up to handle up to 50 checks and deposits per month. These, plus the heading and totals, will fit comfortably on one standard page. If you find the need for more items per month, the dimensions can easily be changed; in lines 250, 980 and 1420.

One other report can be generated. This is the ledger for accumulated months. It is similar to the monthly ledger, but does not need to show balance or check numbers. With it, you can combine any months in any order you like and see totals and percentages. Of course, you can also get quarterly, semi-annual and annual statements from it.

Now, the bad news. You can only get 14 items across the page at 132 characters per line. One of those items needs to be check number; another needs to be balance and yet another has to be deposits. This leaves you with only 11 categories, which may be a little cramped. You don't need to use the categories we have in the program as listed. Line 370 of the program defines them. If you change these category headings, be sure to keep them to 9 characters or less or you will have trouble printing the report headings. It is important that the first item be check number and that the last two be deposit and balance. Other than that, you can call them what you like. Frankly, we would have liked to have almost twice as many categories. One way around this little dilemma would be to lump all non-taxable household expenses into one category, freeing the others for a better breakdown on the more important items.

Also, we used a little trick to liberate one of the categories. Instead of having a separate category for bank fees, we have it set up so that you enter it as a negative deposit. The bank fee amount is then applied to the Misc. column (line 1060 does that.) If you will notice the monthly report, figure 4,

deposits have no check number (they show as zero check number), and we had the same problem with bank fees. So, by making a negative deposit out of your fees, it gets no check number and is applied to the Misc. category. This, by the way, is the only time during entry when you will need to move your hand from the keypad (unless your keypad also has a minus key.) The whole idea here was to make the entry job quick and easy.

```
2 -Auto          6 -Food          10 -Fun
3 -Rent          7 -Insur         11 -Edu
4 -Util          8 -Tax/Lic       12 -Misc
5 -Medical       9 -Maint         13 -Deposit

   Entering information for the month of: JAN
   Use Category 1 to split expenses, 0 to end input.

Check #:  1015    Balance:  155.7201
Category #: ?
```

Figure 1 - Screen Display

Setting your printer

You can initialize your printer in line 180. If you are using MS-DOS, you will need to set the width here. Whatever system you use, if you are using 8 inch wide paper, you will also need to include whatever code you use to produce condensed print. If you are not using MS-DOS, be sure to set your printer for 132 characters before running the program and add a remark at the start of line 180.

Program details

The block of code starting at line 170 does initialization. Here, we set up the print formatting strings which will be used later with the PRINT USING command. Lines 230 and 240 give you space to insert your own name or other identifying lines that will print on both reports. The arrays are dimensioned here in line 250 and our error trap is set in line 260. The error trap is set to detect "file not found" errors and will cause the code to create a new file if none exists.

The print@/locate subroutine follows at line 280. Line 360 checks to see if your computer automatically keeps the date. If it does we simply continue. Otherwise, we prompt you to enter the date. This date is not terribly important, but is printed on each of the reports.

The ledger headings are in data statements in line 370. As mentioned earlier, be sure that check number is the first item and that deposit and balance are the last two, respectively. If you intend to have a miscellaneous column, put it third from last. Keep all these ledger headings under 9 characters so that they will fit into the report format headings. The little loop at line 400 reads the data into the A$() array so that it will be ready for use when we need it. The block of code from 450 to 600 presents the menu on the screen and directs flow to the proper place when we make a selection.

Start or update a month

The code from lines 620 to 1090 is activated from option 1 of the menu. It allows us to start a new month or add to an already started month. It also prints the categories with their associated numbers on the screen so you can refer to them during input. There is quite a bit of activity going on here, with subroutines calling other subroutines, so let's take a typical case and follow it through.

Saving and loading files takes place so often in this program that these functions have been put into subroutines. So has checking for proper file (month) names. Because of this, it looks a little funny at line 640 where to start or update a month, the first thing we do is go to a subroutine. Line 640 sends us to the subroutine at line 1390. Since you must have a file name before you can get a file, the most logical place to put the prompt for file name is in the subroutine that gets the file.

The first thing that happens at 1390 is that we ask for the month name to be input. Here you can enter upper case, lower case or mixed case; the program will sort it out. You need to enter at least the first three characters of the month. If you don't, line 1400 will go back and repeat the prompt until you do. Assuming that we have at least three characters, line 1400 will send us to another subroutine at line 1120. At line 1120, we will change the characters you input to all upper case, and then in line 1180 we will check to see if what you input corresponds to any of the month abbreviations in that line. If the program cannot find a match, line 1190 sends control back to the main menu. When it finds a match, we return to line 1410. Now, we try to open the file. If the file does not exist on disk, our error trap will spring and we will be sent to line 1220. Here, a file with the name we gave will be created. We already know the name is a valid month abbreviation, because we just checked for that. In lines 1240 through 1260 we open an empty file and go back to 1410, where we read the file we just created. At this point, there is nothing in it. But we have established the file

name as valid, made a file with that name, and in line 1490 we have established that there is nothing in the file because variable N will be zero. We now return to line 660.

The code starting at line 670 takes our category names (they are in the A$ array) and makes three nice columns of four items each at the top of the screen. Variables Y and X are our cursor positioning variables, and we manipulate them in three little loops to do the category display. Right underneath the display of categories, we print a line that tells what month we are working with. The month name and the file name are one and the same. Then we print a line that says that we can use category number 1 to split expenses and category 0 to end input.

At line 830 we check the condition of variable N. If it is zero then the file is empty (i.e., we are starting a new month). If this is the case, we jump down to line 930 and ask for the starting check number and the starting balance. If variable N does not equal zero it means that there is some data in the file, and that we want to continue adding information to it. Since there is information in the file, we can look back and find out what the next check number will be and also get the current balance. This happens in lines 870 to 910. You would think that all we had to do was go and look at the last item in the file and get the check number from it, then increment it by one and go on. It's not that simple. What if the last item in the file was a deposit? Then the check number would be zero. What if the last three items in the file were two deposits and a bank fee? Well, the loop at line 870 reads backwards from the end of the file, and if the check number it finds is zero we just go back another step to see what the next item has. As soon as we find a check number other than zero, line 890 increments that check number by one and we get out of that loop. We know that the last item in the file, regardless, will have the current balance. So in line 910 we set the balance (BL) to equal what is in array A (last line, last item) or A(N,14).

Since we now know the starting check number and the balance, we jump right around the next two lines (930 and 940) and clear out our "dialogue area" (the area on the screen where we will print prompts and take in information.)

Input Data

After all that, we finally get to the point of actually entering data. It happens from lines 980 through 1090. Let's talk about that For...Next loop first. Why do you suppose we introduce variable N1 and then go from N plus 1 to 50? Because it allows us to either start filling an empty file or, more importantly, add to the end of an existing sequential file. If the file is empty N will equal zero and we start filling the file at one. If, say, N is already equal to 14, we start adding to the file at 15.

Inside this loop, we make a couple of checks first to see what was entered. In line 1010 if we had entered zero for the category number we go to a subroutine at line 1290 to save the file and when we come back we RUN 100. The RUN 100 is a way of resetting the error trap in case it had been sprung. If the category number we entered was one it means that we want to split the expenses from one check into more than one category. Line 1020 will send us to a subroutine at line 1540 to do that. Let's save its explanation for a little later. If the category number is other than zero or one, we go on and ask what the amount (variable AM) is. Next, in line 1040, we check to see that the category number is within the proper range. If it is not, we go back to line 1000 and ask all over again. Otherwise, we apply the proper amount to the proper category.

Line 1050 says that if the category was not a deposit, then balance will be equal to the balance less the amount, the first item in the record will equal the check number, the proper category will get the amount, the last item in the record (balance) will equal BL and the check number will be incremented by one (for the next entry.) If you read this and refer to line 1050 you can see how it happens.

Line 1060 treats a bank fee like a negative deposit and assigns it to the Misc. category. You could assign it to any category you want by changing the "12" in line 1060 to another category number. That's the "12" in the part of the line that reads: A(N1,12)=ABS(AM). Line 1070 treats a deposit like a deposit (no check number.) Lines 1050, 1060 and 1070 all lead to line 1080, where we clear the dialogue area and get ready for the next input. If your bank charges a fee for each check you write you can easily incorporate that into line 1050.

How about a void check? When you get to a void check, simply assign it to any valid category and enter zero for the amount.

Split Expenses

We have already covered the following sections of code, and that brings us to the split expenses subroutine. Here you can spread one check amount across as many categories as you want. This one is especially handy when you have paid a credit card

Continues on page 17

Your name goes here
Your address goes here

Monthly Check Ledger for JAN — Date run: 12-13-1987

Ck#	Auto	Rent	Util	Medical	Food	Insur	Tax/Lic	Maint	Fun	Edu	Misc	Deposit	Balance
1000	155.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	344.67
1001	0.00	0.00	89.97	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	254.70
1002	0.00	0.00	0.00	0.00	25.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	229.70
1003	0.00	0.00	0.00	55.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	174.70
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	174.70
1004	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	975.50	1150.20
1005	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.95	0.00	1145.25
1006	0.00	0.00	0.00	0.00	0.00	157.62	0.00	0.00	24.95	0.00	0.00	0.00	1120.30
1007	0.00	0.00	0.00	0.00	0.00	0.00	214.48	0.00	0.00	0.00	0.00	0.00	962.68
1008	0.00	0.00	0.00	0.00	0.00	0.00	0.00	33.36	0.00	0.00	0.00	0.00	748.20
1009	0.00	350.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	714.84
1010	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	13.50	0.00	0.00	0.00	364.84
1011	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	45.00	0.00	0.00	351.34
1012	0.00	0.00	0.00	0.00	33.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	306.34
1013	0.00	0.00	0.00	0.00	0.00	73.08	0.00	0.00	0.00	0.00	0.00	0.00	273.09
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	5.72	0.00	200.01
1014	0.00	0.00	0.00	0.00	0.00	0.00	0.00	38.57	0.00	0.00	0.00	0.00	194.29
													155.72
Tot.	155.33	350.00	89.97	55.00	58.25	230.70	214.48	71.93	38.45	45.00	10.67	975.50	

Total expenses: $ 1319.78
Percentage of total expenses:

11.8	26.5	6.8	4.2	4.4	17.5	16.3	5.5	2.9	3.4	0.8

Figure 4 - The Monthly Report

```
Your name goes here
Your address goes here
                    Accumulated expenses for jan feb                    Date run: 12-13-1987
-------------------------------------------------------------------------------------------
          Auto    Rent    Util  Medical    Food   Insur  Tax/Lic   Maint     Fun     Edu    Misc  Deposit
-------------------------------------------------------------------------------------------
JAN     155.33  350.00   89.97    55.00   58.25  230.70   214.48   71.93   38.45   45.00   10.67   975.50
FEB     150.50  350.00   38.45    23.50   82.23   35.50   157.37   15.63    0.00  187.39   30.30  1520.86
        305.83  700.00  128.42    78.50  140.48  266.20   371.85   87.56   38.45  232.39   40.97  2496.36
Total expenses:  2390.65
Percentage of total expenses:
         12.79   29.28    5.37     3.28    5.88   11.14    15.55    3.66    1.61    9.72    1.71
```

Figure 5 - The Accumulated Months Report

```
Line #              Editing data for month of JAN
 1 - 1000  155.33  0  0  0  0  0  0  0  0  0  0  0  344.67
 2 - 1001  0  0  89.97  0  0  0  0  0  0  0  0  0  254.7
 3 - 1002  0  0  0  0  25  0  0  0  0  0  0  0  229.7
 4 - 1003  0  0  0  55  0  0  0  0  0  0  0  0  174.7
 5 - 0  0  0  0  0  0  0  0  0  0  0  0  975.5  1150.2
 6 - 1004  0  0  0  0  0  0  0  0  0  0  0  4.95  0  1145.25
 7 - 1005  0  0  0  0  0  0  0  0  0  24.95  0  0  0  1120.3
 8 - 1006  0  0  0  0  0  0  157.62  0  0  0  0  0  0  962.6801
 9 - 1007  0  0  0  0  0  0  0  214.48  0  0  0  0  0  748.2001
10 - 1008  0  0  0  0  0  0  0  0  33.36  0  0  0  0  714.8401
11 - 1009  0  350  0  0  0  0  0  0  0  0  0  0  0  364.8401
12 - 1010  0  0  0  0  0  0  0  0  0  13.5  0  0  0  351.3401
13 - 1011  0  0  0  0  0  0  0  0  0  0  45  0  0  306.3401
14 - 1012  0  0  0  0  0  33.25  0  0  0  0  0  0  0  273.0901
15 - 1013  0  0  0  0  0  0  73.08  0  0  0  0  0  0  200.0101
16 - 0  0  0  0  0  0  0  0  0  0  0  0  5.72  0  194.2901
17 - 1014  0  0  0  0  0  0  0  0  38.57  0  0  0  0  155.7201

What check # (or line #) do you want to change ?
```

Figure 2 - The Edit Screen

```
1 -Ck#       1004      8 -Tax/Lic    0
2 -Auto      0         9 -Maint      0
3 -Rent      0         10 -Fun       0
4 -Util      0         11 -Edu       0
5 -Medical   0         12 -Misc      4.95
6 -Food      0         13 -Deposit   0
7 -Insur     0

Enter category number to change or 0 to quit
?
```

Figure 3 - Edit one item

statement with one check. Using the statement, you can assign the proper amounts to the proper categories. This subroutine will remain active until the amount of the check remaining is zero. It first asks you for the total amount of the check, and then, one after another, you tell it how much to each category until the initial amount has been reduced to zero. After each entry, it tells you how much you still have to apply somewhere.

Editing data

Lines 1650 through 1940 are used to edit a file of checks. The first thing we do is read in the appropriate file to edit in line 1670. The double loop in lines 1690 to 1750 then prints the entire file on the screen. See figure 2 for an example of how this looks. In line 1770 we have a choice of how to select the line to edit. It turns out that the check numbers and line numbers are so dissimilar that we can search by either the line number shown on the

screen or for a specific check number. This happens to work out quite well, since if we intend to edit a deposit amount we could only get to the first one in the file because they all start with zero. So, if you want to edit a deposit or a bank fee amount, use the line number associated with that value for the search. The search takes place between lines 1780 and 1800.

When line 1790 has found the proper line item to edit, control drops through to line 1810. Here, we clear the screen and print just that one line item in two columns (see figure 3.) This is accomplished in two loops from line 1820 through 1870. Now you can select the item number to change, and for example, move an amount from one category to another, or fix the value of an amount that had been entered incorrectly. Don't worry about the fact that you cannot edit the balance. If you make editing changes that affect the balance it can be taken care of by a different main menu item that will re-balance the entire month automatically.

When you enter zero to signal that you are done editing, line 1910 will send control to the save file subroutine at line 1290. After saving the file we come back to line 1910 and immediately run the program from the beginning, which will display the main menu.

Print a month routine

This section of code (from line 1960 to 2370) is mostly a housekeeping operation. Its purpose is to print the data in nice columns with decimal points all lined up. However, there are a couple of other things we need to do here; total each column and then provide a total of totals across all the columns.

The first thing we do (again) is to read in the month (file) in question in line 1980. Then we print some identification (lines 1990-2020.) Line 2040 prints the column headings (our categories.) Line 2030 is a "calibration" line, so that you can get the proper number of spaces in the PRINT USING portion of the next line.

The check data is then printed with two loops in lines 2060 through 2120. Next, we get the total of each column in variable T(J). We don't need a total of check numbers, so the loop from 2140 to 2190 only goes from 2 to 13, and accumulates the totals for each column. Variable T(3), for example, will contain the total of all amounts in column 3 (Rent, in our case.)

Now that we have the totals all tucked away, lines 2210 through 2250 will print them for us. While we are doing that we have to look at each of the totals, so why not accumulate them for the total of totals? We do just that in line 2240, where T(14) sums up all the totals in T(J). We can use T(14) (it would normally be Balance) because we never need to get a total on the balance column. There is just one problem though, that being that our total of totals in T(14) also contains the deposits. But we already know that the total deposits are in T(13), so when we print the total expenses in line 2270 we simply subtract T(13) from T(14) and print the difference.

All that is left to do now is to print the percentage of each column compared to the total expenses. Lines 2290 through 2360 handle this. In lines 2320 we figure the percentage and print it under each column. After this, we give the printer a page eject - CHR$(12) - and then go back to the main menu via the RUN 100 command in 2370.

Print Accumulated Months

The accumulated months routine is very similar to that for a single month. There are a few differences. In this option, you can enter more than one month to accumulate, so the code from 2480 to 2530 checks first that all the months you enter are valid (via the GOSUB 1120 in line 2510) and if any are not, the subroutine at 1120 will send control back to the main menu so you can try again. Also, since several months need to be kept track of, the month (file) names are kept in a new array, M$(). Note that the months you accumulate need not be consecutive. You can accumulate FEB, APR and OCT if you want to.

In this section there is no need to indicate check numbers or balance, since they would be meaningless. This gives us a little more space across the page, which is fortunate, because individual amounts may grow when we accumulate them. We take advantage of this extra space to allow for amounts that may get up to five significant figures. Figure 5 shows a typical report of accumulated months.

The Re-balance Option

Like the accumulated months option, this option allows you to enter a series of months or just one. In this case, however, the months you enter should be consecutive since non-consecutive months would make no sense. The code in the first part of this option (from lines 3060 to 3130) is strikingly similar to that in the previous option (from line 2460 to 2530.) It performs the very same function as in the earlier option.

The re-balance option gives you the opportunity to correct the balance (through a whole series of

months, if need be) when you have made an entry error or have edited in a different amount in some month. It is not an uncommon occurrence, so this option does come in handy, especially when it's August already and you find a mistake in February's data.

We haven't used the zero array position in the A(X,X) array. It comes in handy here, where we need to keep track of last month's ending balance and use it to start next month. In line 3140 we input the correct starting balance, variable BL. Inside the loop from 3160 to 3240 we set A(0,14) to equal BL. Then, in an inner Q loop (line 3200) we say that A(Q,14) is equal to the old balance, A(Q-1,14), plus the deposits, if any, less all the expenses. After going through an entire month (i.e., going through the Q loop from 1 to the number of items in that month) we set BL equal to the last balance amount and then go get the next month. At the beginning of the next month, A(0,14) will again be set equal to the balance in BL that was held over from the last month. This is the way we can chain the balance from month to month.

So there it is, just in time to do your year-end totals. It's not a budget program, per se, but we have used it on our own checkbook and found it both easy to use and informative. ■

Main listing for GW-BASIC
Changes for others follow.

```
100 REM * Ledger.Bas * Written for CodeWorks Magazine,
110 REM * 3838 S. Warner St. Tacoma, WA 98409
120 REM * (206)475-2219 voice (206)475-2356 300/1200 modem
130 REM * (c)1987 80-NW Publishing Inc.,& placed in public domain
140 '
150 'CLEAR 5000 ' Use only if you need to clear string space
160 '
170 'do some initialization
180 WIDTH LPRINT 132
190 FA$="\          \"
200 FM$=" #####.##"
210 FP$=" ######.##"
220 FO$="####"
230 D1$="Your name goes here"
240 D2$="Your address goes here"
250 DIM A(50,15),B(14),C(14),A$(14),T(14),M$(13)
260 ON ERROR GOTO 1220
270 '
280 'Universal Locate/Print@ subroutine - pick what works for you.
290 GOTO 360
300 LOCATE X,Y:RETURN
310 'PRINT@((X-1)*64)+(Y-1),;:RETURN 'Tandy Models I/III
320 'PRINT@((X-1),(Y-1)),;:RETURN 'Tandy Model IV
330 'PRINT@(X,Y),;:RETURN 'Some MBASIC machines
340 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN 'CP/M
350 '
360 IF DATE$="" THEN INPUT"Enter today's date ";DATE$
370 DATA Ck#,Auto,Rent,Util,Medical,Food,Insur,Tax/Lic,Maint,Fun,Edu,
    Misc,Deposit,Balance
380 '
```

```
390 '  read in the data items
400 FOR I=1 TO 14
410   READ A$(I)
420 NEXT I
430 '
440 'print the heading and menu
450 CLS
460 PRINT STRING$(22,45)" The CodeWorks "STRING$(23,45)
470 PRINT"          C H E C K   L E D G E R   P R O G R A M
480 PRINT"                 a twelve-column check ledger
490 PRINT STRING$(60,45)
500 PRINT
510 PRINT TAB(5);" 1 - Update (or start a new) month.
520 PRINT TAB(5);" 2 - Print ledger for any month.
530 PRINT TAB(5);" 3 - Print ledger for accumulated months.
540 PRINT TAB(5);" 4 - Edit data in any month.
550 PRINT TAB(5);" 5 - Re-balance month or months.
560 PRINT TAB(5);" 6 - End session.
570 PRINT
580 PRINT TAB(5);:INPUT"The number of your choice ";XX
590 ON XX GOTO 630,1970,2400,1660,3000,3280
600 GOTO 580
610 '
620 'update or start a new month module
630 CLS:PRINT TAB(10);"Update or start a new month"
640 GOSUB 1390 ' to open the file and read it
650 '
660 ' Display the input screen
670 CLS
680 Y=1
690 FOR X=1 TO 4
700   GOSUB 300:PRINT X+1;"-";A$(X+1)
710 NEXT X
720 Y=20
730 FOR X1=5 TO 8
740   X=X1-4:GOSUB 300:PRINT X1+1;"-";A$(X1+1)
750 NEXT X1
760 Y=40
770 FOR X2=9 TO 12
780   X=X2-8:GOSUB 300:PRINT X2+1;"-";A$(X2+1)
790 NEXT X2
800 X=6:Y=5:GOSUB 300:PRINT"Entering information for the month of: ";
    M$
810 X=7:Y=5:GOSUB 300:PRINT"Use Category 1 to split expenses, 0 to
    end input."
820 '
830 ' if n=0 the file is empty
840 IF N=0 THEN 930
850 '
860 'otherwise, find the last check number
870 FOR FB=N TO 1 STEP -1
880   IF A(FB,1)=0 THEN 900
890   CN=A(FB,1)+1:GOTO 910
900 NEXT FB
```

```
910 BL=A(N,14)
920 GOTO 950
930 X=9:Y=1:GOSUB 300:INPUT"What is the starting check number";CN
940 X=10:Y=1:GOSUB 300:INPUT"What is the starting balance";BL
950 Y=1:FOR X=9 TO 14:GOSUB 300:PRINT STRING$(60,32):NEXT X
960 '
970 'input data
980 FOR N1=N+1 TO 50
990   X=9:Y=1:GOSUB 300:PRINT"Check #: ";CN;"   ";"Balance: ";BL
1000   X=10:Y=1:GOSUB 300:INPUT"Category #: ";CA
1010   IF CA=0 THEN GOSUB 1290:RUN 100
1020   IF CA=1 THEN GOSUB 1540:GOTO 1080
1030   X=11:Y=1:GOSUB 300:INPUT"Amount: ";AM
1040   IF CA<2 OR CA>13 THEN 1000
1050   IF CA<=12 THEN BL=BL-AM:A(N1,1)=CN:A(N1,CA)=AM:A(N1,14)=BL:
       CN=CN+1:GOTO 1080
1060   IF CA=13 AND SGN(AM)=-1 THEN BL=BL-ABS(AM):A(N1,12)=ABS(AM):
       A(N1,14)=BL:GOTO 1080
1070   IF CA=13 THEN BL=BL+AM:A(N1,13)=AM:A(N1,14)=BL
1080   Y=1:FOR X=9 TO 14:GOSUB 300:PRINT STRING$(60,32):NEXT X
1090 NEXT N1
1100 '
1110 ' subroutine to change to UC and check for valid month
1120 M$=LEFT$(M$,3)
1130 FOR I=1 TO 3
1140   C$=MID$(M$,I,1)
1150   IF C$=>"a" AND C$<="z" THEN C$=CHR$(ASC(C$)-32)
1160   MID$(M$,I,1)=C$
1170 NEXT I
1180 IF INSTR(".JAN.FEB.MAR.APR.MAY.JUN.JUL.AUG.SEP.OCT.NOV.DEC.","."+
     M$+".") THEN RETURN
1190 GOTO 450
1200 '
1210 ' make a file if none exists.
1220 IF ERR <> 53 THEN ON ERROR GOTO 0
1230 'IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0 'BASICs prior to ver
     5.0
1240 OPEN "O",1,M$
1250 CLOSE
1260 GOTO 1410
1270 '
1280 'save file subroutine
1290 OPEN "O",1,M$
1300 FOR I=1 TO N1-1
1310   FOR J=1 TO 14
1320     PRINT #1,A(I,J);
1330   NEXT J
1340 NEXT I
1350 CLOSE #1
1360 RETURN
1370 '
1380 'open file and read subroutine
1390 INPUT"What month (first three letters)";M$
1400 IF LEN(M$)<3 THEN 1390 ELSE GOSUB 1120
```

```
1410 OPEN "I",1,M$
1420 FOR I=1 TO 51
1430  IF EOF(1) THEN 1490
1440  FOR J=1 TO 14
1450    INPUT #1,A(I,J)
1460 '  PRINT A(I,J);
1470  NEXT J
1480 NEXT I
1490 N=I-1
1500 CLOSE #1
1510 RETURN
1520 '
1530 'split expenses subroutine
1540 X=11:Y=1:GOSUB 300:INPUT"What is the total amount of the check";
     TM
1550 BL=BL-TM:A(N1,1)=CN:CN=CN+1
1560 X=12:Y=1:GOSUB 300:INPUT"Category # ";CA
1570 IF CA<2 OR CA>12 THEN 1560
1580 X=13:Y=1:GOSUB 300:PRINT"Amount to category ";CA;:INPUT AM
1590 A(N1,CA)=A(N1,CA)+AM:TM=TM-AM
1600 IF TM<=0 THEN RETURN
1610 Y=1:FOR X=9 TO 13:GOSUB 300:PRINT STRING$(60,32):NEXT X
1620 X=11:Y=1:GOSUB 300:PRINT"You still have ";TM;" to apply.
1630 GOTO 1560
1640 '
1650 'edit data in any month
1660 CLS
1670 GOSUB 1390 ' to read in the file
1680 CLS:PRINT"Line #";TAB(20);"Editing data for month of ";M$
1690 FOR I=1 TO N
1700   PRINT I;"-";
1710   FOR J=1 TO 14
1720     PRINT A(I,J);
1730   NEXT J
1740   PRINT
1750 NEXT I
1760 PRINT
1770 INPUT"What check # (or line #) do you want to change ";CN
1780 FOR I=1 TO N
1790   IF A(I,1)=CN OR I=CN THEN 1810
1800 NEXT I
1810 CLS
1820 FOR L=1 TO 7
1830   PRINT L;"-";A$(L);TAB(14);A(I,L)
1840 NEXT L
1850 FOR L=8 TO 13
1860   Y=25:X=L-7:GOSUB 300:PRINT L;"-";A$(L);TAB(38);A(I,L)
1870 NEXT L
1880 PRINT:PRINT
1890 PRINT"Enter category number to change or 0 to quit"
1900 INPUT K
1910 IF K=0 THEN N1=N+1:GOSUB 1290:RUN 100
1920 INPUT"Change to ";AM
1930 A(I,K)=AM
```

```
1940 GOTO 1810
1950 '
1960 'print out one month's ledger
1970 CLS:PRINT TAB(10);"Print out ledger for one month."
1980 GOSUB 1390 ' to read in the file
1990 LPRINT D1$
2000 LPRINT D2$
2010 LPRINT TAB(45);"Monthly Check Ledger for ";M$;TAB(110);"Date run:
     ";DATE$
2020 LPRINT STRING$(130,45)
2030 '0123456789012345678901234567890123456789
2040 FOR I=1 TO 14:LPRINT USING "\       \";A$(I);:NEXT I
2050 LPRINT" ":LPRINT" "
2060 FOR I=1 TO N
2070   LPRINT USING FO$;A(I,1);
2080   FOR J=2 TO 14
2090     LPRINT USING FM$;A(I,J);
2100   NEXT J
2110   LPRINT" "
2120 NEXT I
2130 '
2140 FOR J=2 TO 13
2150   FOR I=1 TO N
2160     T(J)=T(J)+A(I,J)
2170   NEXT I
2180 NEXT J
2190 LPRINT" "
2200 '
2210 LPRINT"Tot.";
2220 FOR J=2 TO 13
2230   LPRINT USING FM$;T(J);
2240   T(14)=T(14)+T(J)
2250 NEXT J
2260 LPRINT" "
2270 LPRINT"Total expenses: $";T(14)-T(13)
2280 '
2290 LPRINT"Percentage of total expenses:"
2300 LPRINT"    ";
2310 FOR J=2 TO 12
2320   PC=(T(J)/(T(14)-T(13)))*100:LPRINT USING "     ##.#";PC;
2330 ' 012345678901234567890123456789012345678901234567890123456789012
2340   PC=0
2350 NEXT J
2360 LPRINT CHR$(12) 'give the printer a page eject
2370 RUN 100
2380 '
2390 'print accumulated months option
2400 CLS:PRINT "ACCUMULATED MONTHS OPTION.
2410 PRINT
2420 PRINT"Enter months to accumulate separated by a space."
2430 PRINT"Example: JAN FEB MAR APR<Enter>"
2440 PRINT"Improper entry here returns you to main menu.
2450 PRINT
2460 PRINT"Enter months to accumulate "
```

```
2470 INPUT MO$
2480 K=1
2490 FOR I2=1 TO LEN(MO$) STEP 4
2500   M$(K)=MID$(MO$,I2,3)
2510   M$=M$(K):GOSUB 1120:M$(K)=M$
2520   K=K+1
2530 NEXT I2
2540 LPRINT D1$:LPRINT D2$
2550 LPRINT TAB(20);"Accumulated expenses for ";MO$;TAB(110);"Date
     run: ";DATE$
2560 LPRINT STRING$(130,45)
2570 LPRINT"   ";
2580 FOR I4=2 TO 13
2590   A$(I4)=RIGHT$("       "+A$(I4),10)
2600 ' 012345678901234567890123 45
2610   LPRINT USING FA$;A$(I4);
2620 NEXT I4
2630 LPRINT STRING$(130,45)
2640 LPRINT" "
2650 PRINT "Merging: ";
2660 FOR I3=1 TO K-1
2670   M$=M$(I3):PRINT M$+" ";:GOSUB 1410
2680   FOR P=2 TO 13
2690     FOR Q=1 TO N
2700       B(P)=B(P)+A(Q,P)
2710       C(P)=C(P)+A(Q,P)
2720     NEXT Q
2730   NEXT P
2740   LPRINT M$(I3);
2750   FOR I2=2 TO 13
2760     LPRINT USING FP$;B(I2);
2770     C(14)=C(14)+B(I2)
2780     B(I2)=0
2790   NEXT I2
2800   LPRINT" "
2810 NEXT I3
2820 LPRINT" "
2830 LPRINT"   ";
2840 FOR I2=2 TO 13
2850   LPRINT USING FP$;C(I2);
2860 NEXT I2
2870 LPRINT" "
2880 LPRINT"Total expenses: ";C(14)-C(13)
2890 LPRINT"Percentage of total expenses:"
2900 LPRINT"   ";
2910 FOR I2=2 TO 12
2920   PC=(C(I2)/(C(14)-C(13)))*100
2930   LPRINT USING FP$;PC;
2940   PC=0
2950 NEXT I2
2960 LPRINT CHR$(12)
2970 RUN 100
2980 '
```

```
2990 're-balance month or months
3000 CLS:PRINT"Re-balance month or months option."
3010 PRINT
3020 PRINT"Enter month to re-balance or enter months
3030 PRINT"separated by one space. Example: Feb Mar Apr<Enter>
3040 PRINT"Improper entry here returns you to main menu.
3050 PRINT
3060 PRINT"Enter month or months"
3070 INPUT MO$
3080 K=1
3090 FOR I2=1 TO LEN(MO$) STEP 4
3100   M$(K)=MID$(MO$,I2,3)
3110   M$=M$(K):GOSUB 1120:M$(K)=M$
3120   K=K+1
3130 NEXT I2
3140 INPUT"Enter the correct starting balance";BL
3150 PRINT"Re-balancing: ";
3160 FOR I3=1 TO K-1
3170   M$=M$(I3):PRINT M$+" ";:GOSUB 1410
3180   A(0,14)=BL
3190   FOR Q=1 TO N
3200     A(Q,14)=A(Q-1,14)+A(Q,13)-(A(Q,2)+A(Q,3)+A(Q,4)+A(Q,5)+A(Q,
         6)+A(Q,7)+A(Q,8)+A(Q,9)+A(Q,10)+A(Q,11)+A(Q,12))
3210   NEXT Q
3220   BL=A(N,14)
3230   N1=N+1:GOSUB 1290 ' to save the updated month
3240 NEXT I3
3250 RUN 100
3260 '
3270 'end program
3280 CLS:PRINT"Done"
3290 END' of program
```

Change lines for Tandy II/IV

```
Changed->100 REM * Ledger/Bas * Written for CodeWorks Magazine,
Changed->180 'WIDTH LPRINT 132
Changed->300 'LOCATE X,Y:RETURN
Changed->320 PRINT@((X-1),(Y-1)),;:RETURN 'Tandy Model II/IV
Changed->700   GOSUB 300:PRINT X+1;"-";A$(X+1);
Changed->740   X=X1-4:GOSUB 300:PRINT X1+1;"-";A$(X1+1);
Changed->780   X=X2-8:GOSUB 300:PRINT X2+1;"-";A$(X2+1);
Changed->1860   Y=25:X=L-7:GOSUB 300:PRINT L;"-";A$(L);TAB(38);A(I,L)
;
Added--->2355 LPRINT" "
Added--->2955 LPRINT" "
```

Change lines for Tandy I/III

```
Changed->100 REM * Ledger/Bas * Written for CodeWorks Magazine,
Changed->150 CLEAR 5000 ' Use only if you need to clear string space
Changed->180 'WIDTH LPRINT 132
Changed->190 FA$="%         %"
Changed->300 'LOCATE X,Y:RETURN
Changed->310 PRINT@((X-1)*64)+(Y-1),;:RETURN 'Tandy Models I/III
Changed->360 'IF DATE$="" THEN INPUT"Enter today's date ";DATE$
Changed->700   GOSUB 300:PRINT X+1;"-";A$(X+1);
Changed->740   X=X1-4:GOSUB 300:PRINT X1+1;"-";A$(X1+1);
Changed->780   X=X2-8:GOSUB 300:PRINT X2+1;"-";A$(X2+1);
Changed->1220 'IF ERR <> 53 THEN ON ERROR GOTO 0
Changed->1230 IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0 'BASICs prior t
o ver 5.0
Changed->1860   Y=25:X=L-7:GOSUB 300:PRINT L;"-";A$(L);TAB(38);A(I,L)
;
Changed->2040 FOR I=1 TO 14:LPRINT USING "%         %";A$(I);:NEXT I
Added--->2355 LPRINT" "
Added--->2955 LPRINT" "
```

Computing Notes

The processing speed of machines these days is going up dramatically. The machines we cover run the speed range from about 1 megahertz all the way to 16 and 20 megahertz. This makes things happen a little differently depending on which machine you have. As a case in point, we recently ran Slot.Bas (Issue 14) on a machine with a speed of 8 megahertz. The slot machine handle flew up and down so fast you could hardly see it. Fortunately, it's easier to slow down a machine than it is to speed it up. If things happen too fast you can insert some delay loops. Basically, a delay loop is nothing but a one-liner that looks like this:

FOR I=1 to 5000:NEXT I

You can change the 5000 up or down depending on how much delay you want to have. Where you put the loop depends on where the action is. In Slot.Bas, for example, you would put one at line 1625, where the handle pulls down. There already is a delay loop at line 1710, so all you would need to do there is increase the count. Actually, you wonder how they keep from radiating radio frequencies at 20 megahertz and messing up radio and television. Isn't technology wonderful?

Check the CHR$() commands from CHR$(31) to CHR$(1) for your machine. You can find them listed in your computer manual. They vary depending on which machine you are using, and do some interesting things. Some clear from the current cursor position to the end of the screen, others to the end of the line. Some make reverse video (black on white). CHR$(7) almost universally makes a beep or rings a bell. On some machines there are codes to make the screen change width to half size and make the characters larger. Also note that CTRL-A corresponds to CHR$(1), etc. CTRL-G is the same as CHR$(7), Bell. CTRL-C is the same as CHR$(3), Break, and CTRL-M is the same as the Enter key, CHR$(13). In trying these, you may hang up your computer because in some cases, there may be a CHR$() that wants to print to the printer. If so, turn the printer on and see what happens.

Random Files

Improvements to Randemo5.Bas

Terry R. Dettmann, Associate Editor. The two parts to this installment will make either Randemo6 or Randemo7 out of Randemo5. Both provide needed improvements and make the program operation more convenient.

Part I

Part I of this installment is only for those of you who use MS-DOS, then only for those who are NOT going to compile their program and want records longer than 128 bytes. It will make Randemo6.Bas out of Randemo5.Bas. The rest of you can go ahead to the second part of this installment. The two parts to this installment are NOT mutually exclusive, you can do either or both. TRSDOS users should not use Part I of this article because their record length is automatically limited to 256 bytes.

One of the limitations we have not dealt with until now with the random file programs has been record size. To simplify discussion, we limited the buffer to 128 characters so that anyone could run it without doing anything special. It has served well so far, but now it's time to change things. Many of you have asked how to get larger records; in this installment we will show you how to do it.

The random files series was designed to run on a variety of different machines that all use Microsoft BASIC. What a random file can do on these different machines varies. For example, on the early Tandy (TRSDOS) machines, the maximum record length is generally 256 characters, while on the newer MS-DOS machines the limit can be as high as 32,767 characters. So why did we choose 128 characters as a starting point? Because on MS-DOS machines, that is the default record length. So how can you change the record length? There are two steps for MS-DOS machines, only one step for TRSDOS machines:

1. On MS-DOS machines only, when entering BASIC, if you want a record length longer than 128 bytes, you must specify the S parameter to tell how big the random buffer should be.

BASIC /S:1024

specifies a 1024 character record buffer.

2. On both MS-DOS and TRSDOS machines, you add a record length parameter onto the end of the OPEN statement:

OPEN"R",1,"FILE.DAT",1024

To specify a 1024 character record length. Note: A length greater than 256 is only possible on MS-DOS machines; TRSDOS limits the record length to 256.

On most TRSDOS machines, the record length can only be shortened using the record length parameter (don't knock it, saving 100 characters per record on 10,000 records is *1 megabyte!*) For now, we won't go into setting the length until we get to the changes needed to make the randemo program work better. Before we do that though, let's try some simple experiments with setting up random files to see how they work.

First, for MS-DOS users only, let's see what happens when we try to open a file without setting the /S parameter when entering BASIC. Go to BASIC (without the /S parameter) at the DOS Ready prompt by typing:

BASIC (Enter)

Next, type in the following little program:

```
10 OPEN"R",1,"TEST.DAT",512
20 FIELD 1,128 AS A$,128 AS B$,128 AS C$,128
AS D$
30 CLOSE
```

Not much as programs go, but type and enter RUN. As simple as it is, you get an error: "Illegal function call in 10." The error occurs because we didn't start BASIC with enough buffer space for a 512 byte random file record. Now let's try it again by typing, from the DOS ready prompt:

BASIC /S:512

Now, if you run the same program, it works fine. Several of you have got just about this far in modifying the randemo program without getting it to work. Why?

The problem is *fielding*. If you refer to your Nov/Dec 1987 issue of CodeWorks and turn to the listing of the random files program on page 32, the critical portion of code that fails is the field mapping subroutine at line 5300. Specifically, the field statement at line 5330 fails. But why?

If you are working with interpreted BASIC (and most of us are), strings have a maximum length of 255 characters. As we go through, defining fields, eventually we get to defining a field where NL, the position of the field in the record, is more than 255 characters from the beginning of the record. When NL-1 is 256 or greater, the field statement fails. To see this, we have set up a simple demonstration of it:

```
10 OPEN"R",1,"TEST.DAT",512
20 FOR I=1 TO 511
30 FIELD 1,(I-1) AS X$,1 AS A$
40 NEXT I
50 CLOSE
```

In this demo, we still open the file for 512 character records. In line 30, however, we act as if we're defining a field A$ by gradually increasing its offset one character at a time. Run the program and you will get the "Illegal function call in 30" error message. When you get the error, BASIC will return to the Ok prompt. *Before doing anything else*, type:

PRINT I (Enter)

BASIC will print out the value of the loop variable at the point of failure. At that point, I should be 257. With I at 257, the FIELD statement would actually read:

FIELD 1, 256 AS X$,1 AS A$

In other words, we have exceeded the string length limitation on X$. Note that if you compile this program with either QuickBASIC or Turbo BASIC, it will work as is; strings there are limited to 32,767 characters.

So we have a problem. How do we get around it? The solution is messy and not very general, but it will work for interpreted BASIC. If you intend to compile Randemo5.Bas, don't even bother with this change, but if you don't have a compiler then we fix it by going around the interpreter's end. See figure 1 for how it's done.

Notice that in line 30 of figure 1, we have replaced the field statement with a GOSUB to the subroutine at line 100, which will do the fielding. Line 110 decides *which* field statement to use. The value of INT(I/255) will be zero until I equals 255. Until this point, the ON GOTO statement will drop through and line 120 will be used to field the buffer. From I=255 until I=509, X will be 1 and line 130 will be used. From I=510 until I=511, line 140 will be used.

When you run the program in figure 1, you'll see the offset of the variable A$ as it steps through the buffer, one position at a time. Look especially as it passes through 255 and 510 to see that there is no jump in character position as it switches from one fielding line to the next. As a last check, let's see what the same program looks like for a 1024 character buffer size (see figure 2.)

Same thing. Just longer! As you add lines, notice the progression:

1. The offset strings (X1$, X2$ and so on) are always 255 characters *except* for the last one before X$, which is 254.
2. X$ is always fielded as I minus the *first value of I that reaches this line.*

If you play with these lines, you will have to continue using these rules or develop new ones for yourself. Let's look in on the Randemo5.Bas program now and see how to make it possible to extend the record length to 1024 characters.

To make Randemo5.Bas work with larger record sizes (and buffers), we need to apply the same techniques but with slight modification. Again, if you intend to compile Randemo5.Bas with either QuickBASIC or Turbo BASIC, don't make any changes to it - it will work better without these changes. See figure 3 for the merge lines to make Randemo5.Bas field a record with a length of 1024 characters. Type in the lines in figure 3, save them in ASCII and then merge them with your copy of Randemo5.Bas.

With a new record length, we redefine the function LF (length of file) to use the new variable RL (record length) to determine the length of the

file. Line 51 declares the value of RL to be 1024 and line 130 opens the file at this record length.

The major change occurs when we go to subroutine 5300 to field the data file. Now, instead of line 5330 using a single field statement, it's a GOSUB to line 5360 where we select which field statement to use, based on how far we have to offset into the record.

In line 5365, we have determined which line to use by computing X as INT(NL/255) just like we did in the examples. But now, we have also added Y=(NL MOD 255) which will give us the number of characters to field X$ for. Remember that if we compute NL MOD 255, then Y is zero at NL=255, 510, 765, 1020 and so on.

Extensions beyond 1024 characters are possible, but they get quite messy. If you need that much data in a record, you are better off getting a compiler because you will need the speed.

```
10 OPEN "R",1,"TEST.DAT",512
20 FOR I=1 TO 511
30   GOSUB 100
40 NEXT I
50 CLOSE
99 END
100 REM --- field depending on position
110 X = INT(I/255):ON X GOTO 130,140,150
120 FIELD 1, (I-1) AS X$,1 AS A$:Y=(I-1)+1:GOSUB 200:RETURN
130 FIELD 1, 254 AS X1$, (I-255) AS X$, 1 AS A$:Y=254 + (I-
255) + 1:GOSUB 200:RETURN
140 FIELD 1, 255 AS X1$, 254 AS X2$, (I-510) AS X$, 1 AS
A$:Y = 255 + 254 + (I-510) + 1:GOSUB 200:RETURN
150 FIELD 1, 255 AS X1$, 255 AS X2$, 254 AS X3$, (I-765) AS
X$, 1 AS A$:Y = 255 + 255 + 254 + (I-765) + 1:GOSUB
200:RETURN
200 REM --- Print
210 PRINT Y;:RETURN
```

Figure 1

```
10 OPEN "R",1,"TEST.DAT",1024
20 FOR I=1 TO 1023
30   GOSUB 100
40 NEXT I
50 CLOSE
99 END
100 REM --- field depending on position
110 X = INT(I/255):ON X GOTO 130,140,150,160
120 FIELD 1, (I-1) AS X$,1 AS A$:Y=(I-1)+1:GOSUB 200:RETURN
130 FIELD 1, 254 AS X1$, (I-255) AS X$, 1 AS A$:Y=254 + (I-
255) + 1:GOSUB 200:RETURN
140 FIELD 1, 255 AS X1$, 254 AS X2$, (I-510) AS X$, 1 AS
A$:Y = 255 + 254 + (I-510) + 1:GOSUB 200:RETURN
150 FIELD 1, 255 AS X1$, 255 AS X2$, 254 AS X3$, (I-765) AS
X$, 1 AS A$:Y= 255 + 255 + 254 + (I-765) + 1:GOSUB
200:RETURN
160 FIELD 1, 255 AS X1$, 255 AS X2$, 255 AS X3$, 254 AS X4$,
(I-1020) AS X$, 1 AS A$:Y= 255 + 255 + 255 + 254 + (I-1020)
+ 1:GOSUB 200:RETURN
200 REM --- Print
210 PRINT Y;:RETURN
```

Figure 2

```
10 REM --- RANDOM.BAS - CodeWorks Random Database
41 DEF FNLF(X) = LOF(X)/RL
51 CC=1:CX=1:CL=1:RL=1024
130 OPEN "R",1,FD$,RL:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
5300 REM --- Map Fields
5310 FOR I=1 TO CX
5320    NL = XY(I,3)
5330    GOSUB 5360
5340 NEXT I
5350 RETURN
5360 REM --- Field Records
5365 X = INT(NL/255):Y = NL MOD 255:ON X GOTO
5375,5380,5385,5390
5370 FIELD #1, Y-1 AS X$,XY(I,0) AS FP$(I):RETURN
5375 FIELD #1, 254 AS X1$, Y AS X$, XY(I,0) AS FP$(I):RETURN
5380 FIELD #1, 255 AS X1$, 254 AS X2$, Y AS X$, XY(I,0) AS
FP$(I):RETURN
5385 FIELD #1, 255 AS X1$, 255 AS X2$, 254 AS X3$, Y AS X$,
XY(I,0) AS FP$(I):RETURN
5390 FIELD #1, 255 AS X1$, 255 AS X2$, 255 AS X3$, 254 AS
X4$, Y AS X$, XY(I,0) AS FP$(I):RETURN
```

Figure 3

```
10 REM --- RANDEMO6.BAS - CodeWorks Random Database
41 DEF FNLF(X) = LOF(X)/RL
51 CC=1:CX=1:CL=1:RL=1024
130 OPEN "R",1,FD$,RL:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
5300 REM --- Map Fields
5310 FOR I=1 TO CX
5320    NL = XY(I,3)
5330    GOSUB 5360
5340 NEXT I
5350 RETURN
5360 REM --- Field Records
5365 X = INT(NL/255):Y = NL MOD 255:ON X GOTO 5375,5380,5385,5390
5370 FIELD #1, Y-1 AS X$,XY(I,0) AS FP$(I):RETURN
5375 FIELD #1, 254 AS X1$, Y AS X$, XY(I,0) AS FP$(I):RETURN
5380 FIELD #1, 255 AS X1$, 254 AS X2$, Y AS X$, XY(I,0) AS FP$(I):
     RETURN
5385 FIELD #1, 255 AS X1$, 255 AS X2$, 254 AS X3$, Y AS X$, XY(I,0)
     AS FP$(I):RETURN
5390 FIELD #1, 255 AS X1$, 255 AS X2$, 255 AS X3$, 254 AS X4$, Y AS
     X$, XY(I,0) AS FP$(I):RETURN
```

Merge file to make Randemo6 from Randemo5.

Part II

The changes in this part apply to everyone. The merge file in this section can be applied to Randemo5 and will change it to Randemo7. If you have also made the changes in Part I of this article you can apply this merge file to Randemo6.

Improving the User Interface

This part of the article isn't built around any single change in the system. This time we'll put in a number of individually simple changes which taken together will change the way the program

will work and make it a much more useful system. Here is what we are going to do:

1. Modify the ADD procedure to allow adding one record after another without answering that annoying question.
2. Modify the EDIT and DELETE procedures to allow for repeated operation.
3. Modify the SEARCH procedure to allow getting a designated record by number or all records, one at a time.
4. Modify the PRINT procedure to allow display of record number and a wait between record displays.
5. Modify the Record Display procedure to display the record number along with the record.

We'll also do a little cleanup and generalization in a few places.

Sounds rather ambitious, doesn't it? But you will be amazed at how simple it will be. There are only about 50 lines of changes, and quite a few of them are blank remark lines to cancel lines that are no longer needed.

The merge file with this section of the article will update Randemo5.Bas to Randemo7.Bas. To follow along, you will need a current listing of Randemo5.Bas, which was listed in its entirety in Issue 14. The simplest way to go through the changes is to take them from the top. We'll go through the merge file (see listing) and deal with each change as we come to it.

Lines 41, 50 and 130 - Random file record length.

In Part I of this article we covered some extensive changes to allow the program to work with record sizes other than 128 for MS-DOS users. The general changes from that part of the article are included here with the record length set to 128 characters for normal use. Tandy Model I, II, III and IV users can simply set RL in line 50 to 256. MS-DOS users can set it to any size they want to use. Also, Tandy I/II/III/IV users should delete the divide by RL at the end of line 41. Tandy I and III users should also change the variable "DONE" to "DUN" wherever it comes up in the merge file, and change WD to 64 and LN to 16 in line 50.

Lines 850 and 860 - Time Delay.

A useful routine that hasn't been included because it wasn't needed so far is a simple variable delay routine to allow us to wait for a few seconds. This subroutine will be used by some of the other routines to allow us to display a message and leave it on the screen long enough to be read and then have it automatically cleared.

Lines 54, 1810, 1831, 1835 and 1880 - Entry procedures.

I have created two new editing character variables here for use with our entry subroutine (subroutine 1800.) CM$ (for Complete) is our CTRL-C characters (ASCII code 3.) NX$ (for Next) is the character CTRL-N (ASCII code 14.) Now, when adding records, searching or editing, you use CTRL-N for the next record and CTRL-C to get out of that mode. You should update these prompts in your particular .SCN file.

With the editing characters defined, we can change the data entry routine at line 1800. First, we create two new True/False flag variables (MORE for whether or not to request another record and CHG for whether or not any changes were made since this routine was started.) Line 1810 initializes both these new variables to FALSE before we start.

In line 1831, we replace the CHR$(3) with CM$ for standardization, then we set the flags so that when we hit the CTRL-C, we're saying that we are done with the current record (DONE=TRUE) but we don't want another record (MORE=FALSE.) This will cause whatever is calling the entry routine (ADD or EDIT) to finish what it's doing. In line 1835 we introduce a check for the NEXT character (CTRL-N.) If it is found, we say we are done with the current record (DONE=TRUE) and we do want the next record, (MORE=TRUE.) In order to prevent accidental entry of blank records, line 1880 sets the CHANGED flag (CHG=TRUE) if any change is made to a data field. The ADD procedure checks this when working.

Line 1665 - Record Number Display

I have changed the record display routine here to show the current record number at the upper right hand corner of the screen. This way, you can avoid long searches for records where you already know the record number.

Lines 2105, 2106, 2120, 2170, 2180, 2181, 2185 and 2190 - Search procedure

The last utility routine modification involves the search procedure at line 2100. There are several things we wanted to add here: 1. the ability to select a record directly by record number, 2. the ability to select ALL active records one at a time, and 3. the ability to come back to the routine over and over again, continuing a search where we left off. Each modification is included in these lines.

Line 2105 implements the retrieval by record number feature. When a search string (SF$) starts with the pound sign character (#), then the record number is assumed to come right after the # and control is transferred to line 2170 where the actual record retrieval is done. Now, when you are prompted for a search string, you can type in #5 and get record number 5.

Line 2106 checks to see if the search string is the

single word "ALL." If it is, then the record number is simply advanced by one and line 2170 is asked to get the record.

Line 2120 is a change so subtle it's easy to miss. If you look at the Randemo5.Bas listing, you will notice that the loop went from RN=1. By making the loop start at RN=FR, any routine that calls subroutine 2100 can now start a search at the beginning of the file (set FR=1 before calling) or it can continue a search from the present location by setting FR=RN+1 (one past the current record number.)

Lines 2170 through 2190 retrieve a record with a designated record number, but first makes sure it's an active record. Line 2180 checks to make sure the record number is in the file. If it is not, then FOUND is set to FALSE and we return. If it is, we get the record and set FOUND=TRUE, but we don't return yet.

After we have the record, we check to see if it's deleted in line 2181. If it is deleted and we are asking for all records, we simply skip over it (RN=RN+1) and go for another record. If the record is deleted and we just asked for a particular record number (line 2185 has been reached,) then we simply set FOUND to FALSE because the record we asked for has been deleted.

I know that you are going to ask why we can't use this to get a record back. We can. In fact, the deletion procedure intentionally leaves this as a possibility. But to do it, we'll have to be more sophisticated because we've put that record number on our list of deleted records and we'll have to invent a way to remove it. That's a subject for an article all by itself.

Lines 1060, 1065, 1070, 1080, 1090 and 1100 - Add Procedure

With our changed entry procedure, we can now get rid of the lines which asked the question "Add more?" and simply check our True/False flag (MORE) to see if we want to add more records. By pressing CTRL-N, we save the current record and then ask for another. Lines 1060, 1090 and 1100 get rid of unnecessary lines. Line 1065 checks to make sure that something was entered before it saves, for example, if you have a blank record on the screen and type CTRL-C, it won't save the blank record to the file. Line 1070 checks to see whether CTRL-N was pressed for "Next Record" and if it was not, then line 1080 returns us to the menu.

Lines 2020, 2021, 2025, 2040, 2045, 2050, 2060, 2070, 2080 and 2090 - Searching

With our new search procedure available, we can now expand our searching procedures. With the new data entry procedures, we have to treat searches differently. Line 2020 now tells us that if we don't enter a search string, that we are done (just press Enter or Enter without typing anything else and you will go back to the menu.) Line 2021 checks the search string to see if it's empty and returns if it is.

Line 2025 sets up initial conditions for our first search. The first record to check is record number 1 (FR=1) and if we are using the ALL search criteria, the initial record number is zero so that our first call to the search routine will return record number 1 (remember that line 2106 does RN=RN+1 if SF$="ALL".)

Since we no longer have the "Edit more?" question, the NOT FOUND display won't show because the screen will clear before it can. Line 2040 has been changed to do a time delay (subroutine 850) before going back to the clear screen. Lines 2045 through 2090 get rid of the "Edit more?" display and set up the continuation logic. Lines 2045, 2070, 2080 and 2090 just get rid of useless lines in Randemo5.Bas. Line 2050 checks to see that when we finished editing, that we used the MORE key (NEXT record means CONTINUE the search.) So long as we did not ask for a specific record, it will go back and search for the next one. If we didn't ask for more or we asked for a specific record, then we go back to the start and ask for another search.

Lines 3020, 3021, 3025, 3040, 3045, 3050, 3060, 3070, 3080 and 3090 - Delete

The delete record routine works exactly the same as the edit record routine at this level. In fact, these lines are the same as those we've just discussed, they only call different subroutines. All of the previous discussion for editing applies here.

Lines 4035, 4065 and 4066 - Printing

The last change is to make the print option (print the data to the screen) just a little simpler to use. It will print the record number at the beginning of the record (line 4035) and then wait for you to press the Enter key after you have read the record. You might want to leave lines 4065 and 4066 out and change all the PRINT statements to LPRINT to get a fast listing of the records in the database.

If you include all of the changes we have just gone over, you will find that you have arrived at a point where you now have something more than an interesting random files program, you now have the basis for a useful data management system.

We will be adding still more changes (better sorting, larger file sorts, advanced reporting, etc.) and we'll be building applications you can use with the program. You now have control of your own data. ■

```
10 REM --- RANDEMO7.BAS - Random Files with Screen Control
41 DEF FNLF(X) = LOF(X)/RL
50 WD=80:LN=24:RL=128
54 CM$ = CHR$(3):NX$ = CHR$(14)
130 OPEN "R",1,FD$,RL:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
850 REM --- time delay
860 FOR TZ=1 TO CT:NEXT TZ:RETURN
1060 REM
1065 IF CHG THEN RN=0:GOSUB 1200
1070 IF MORE THEN 1010
1080 RETURN
1090 REM
1100 REM
1655 X=2:Y=WD-15:GOSUB 800:PRINT USING "RECORD: ######";RN;
1710 CC=1:CL=1:CHG=FALSE
1735 IF CF THEN CHG=TRUE
1810 IN$="":DONE=FALSE:MORE=FALSE:CF=FALSE
1831 IF C$=CM$ THEN DONE=TRUE:MORE=FALSE:GOTO 1880
1835 IF C$=NX$ THEN DONE=TRUE:MORE=TRUE:GOTO 1880
1880 IF IN$<>"" THEN LSET FP$(CL) = IN$:CF=TRUE
2020 LINE INPUT"SEARCH FOR (RETURN WHEN DONE): ";SF$
2021 IF SF$="" THEN RETURN
2025 FR = 1:RN = 0
2040 IF FOUND THEN GOSUB 2200 ELSE PRINT"NOT FOUND":CT=1000:GOSUB 850:
     GOTO 2010
2045 REM
2050 IF MORE AND MID$(SF$,1,1)<>"#" THEN FR=RN+1:GOTO 2030
2060 GOTO 2010
2070 REM
2080 REM
2090 REM
2105 IF MID$(SF$,1,1)="#" THEN RN = VAL(MID$(SF$,2)):GOTO 2170
2106 IF SF$="ALL" THEN RN = RN + 1:GOTO 2170
2120 FOR RN=FR TO FNLF(1):GOSUB 1400
2170 REM retrieve the record
2180 IF RN>=1 AND RN<=FNLF(1) THEN FOUND=TRUE:GOSUB 1400 ELSE
     FOUND=FALSE:RETURN
2181 IF LEFT$(FP$(1),7)="DELETED" AND SF$="ALL" THEN RN=RN+1:GOTO
     2170
2185 IF LEFT$(FP$(1),7)="DELETED" THEN FOUND=FALSE
2190 RETURN
3020 LINE INPUT"SEARCH FOR (RETURN WHEN DONE): ";SF$
3021 IF SF$="" THEN RETURN
3025 FR = 1:RN = 0
3040 IF FOUND THEN GOSUB 3200 ELSE PRINT"NOT FOUND":CT=1000:GOSUB 850:
     GOTO 3010
3045 REM
3050 IF MORE AND MID$(SF$,1,1)<>"#" THEN FR=RN+1:GOTO 3030
3060 GOTO 3010
3070 REM
3080 REM
3090 REM
4035         PRINT USING "RECORD: ######";RN
4065         PRINT "Press RETURN to continue"
4066         LINE INPUT XX$
```

Merge file to make Randemo7 from Randemo5.

Repl.Bas

A short search & replace utility

Staff Project. This short utility will search and replace in almost any ASCII file, be it a program or a text file. You can also use it just to see if a given string even exists in a file or not.

Here is a companion utility to Diff.Bas (see Issue 11) which performs a global search and replace function. We'll call it Repl.Bas even though it also does the search.

The program has three modes of operation. First, it allows you to search and display the line containing anything you wish from a program or other ASCII file. We'll see why this is handy a little later on. Next, it allows you to search for a specific string and replace it with a different string and it makes this replacement automatically. Third, it allows you to search for anything and then gives you the option to replace it with the replacement string or not.

The first option finds and displays only. This is handy when you want to see if some variable even exists within your program or ASCII file. Why handy? Because if you already have a B$, for example, and change all your A$'s to B$ in addition, you'll end up with a pretty mess. The other two options create a new file with the same name but a new extension called .NEW to differentiate it from the original program. This way, the original program never gets altered, and in case you goof, you can start over and do it right. It should be mentioned here that you cannot load a program with the .NEW extension and search or replace in it. This is because the program is handled as a sequential file and those files cannot be opened for input and output at the same time. You can, after making changes to a file, rename it and continue to make other changes to it.

This program also works with text files created with your text editor or word processor so long as they are in pure ASCII form. This, of course, means that you can use the program to call up a text file and change all occurrences of a misspelled word, for example.

Repl.Bas also has options to print out the changes you make on your printer. In addition, it gives you a little summary at the end, telling you how many lines were examined, how many lines contained the search string and how many replacements were made. All in all, it does quite a bit for its size - and it's not even too terribly slow. And yes, it can be compiled for greater speed. In fact, since the compiled version would run directly from your DOS ready prompt, its convenience is increased. This is because you can invoke this program, make changes to a file, rename the file and make more changes to it without ever leaving the DOS level.

Program details

After the usual opening lines we encounter line 320 where if you give a file name without extension it appends .NEW to it and if you give one with an extension it is changed to .NEW for the output file (F1$). F$ contains the name of the original file. F1$ is the name of the new output file. The program menu is set up to show you the new name automatically in lines 350 and 360.

The search string (string to look for) is called S$, and if there is a replacement string, it is called R$.

Line 430 is a universal "Yes" finder. You can answer yes almost any way you want and it will turn the printer flag (PR) on, otherwise the printer flag is set to zero. If the printer flag is one then lines 450 and 460 will print a heading on the paper. The difference between lines 450 and 460 is that in one case we are only searching and displaying, while in the other two cases we are making a replacement. If we do not opt for printer output, line 440 sends us right around the next two lines.

The find and display only section of code is from 490 to 570. In it, we first open the file as a sequential file for input (line 500). We next check immediately for end of file, and if so we go to line

850 to do the end processing. Otherwise, we input one line at a time and call it A$ (line 520). In line 530 we search for the occurrence of our search string within A$. Here, variable P will hold the character position number in A$ where our search string was found. In other words, if our search string was found at the 14th character of A$, P will equal 14. If our search string was not found at all in A$, P will equal zero. In any case, we have examined a line, so in line 530 we also increment an accumulating counter (LC) by one. If we made it all the way through A$ without encountering our search string P will be zero and line 540 will send us back to 510 to check for EOF again and then input another line.

If P does not equal zero (which means that it has found our search string), line 550 first prints the entire A$ on the screen. Then, since P is the character position in A$ where our search string was found, we can tab to P on the next line (right under the current A$ on the screen) and print a pretty little up-arrow, pointing right at the beginning of our search string. CHR$(24) is the up-arrow in GW-BASIC. In the Tandy III and IV change lines at the end of the listing you will see that it is changed for them to CHR$(133), which is a thin vertical bar. CP/M machines without graphics characters or arrows may want to change this to the exclamation point or something else that looks vertical and pointy. Also in line 550, since we have found an occurrence of our search string, we increment the accumulating counter (CH) by one. If we had opted for printer output, line 560 will print A$ on the printer, after which we go back and fetch another line to examine. This continues until we reach end of file (EOF) in our input file (F$). You could, if you like, make line 560 similar to line 550, so that the search string would be pointed out on the line printer as well as on the screen. As it stands now, it won't.

The section of code from 590 through 820 is a horse of another color. Here, we service the second and third options of the menu, wherein we must make a replacement and write a new file as well. This means we must write lines to the output file that *don't* contain the search string as well as those that do. Also within this section, we must provide for both automatic replacement and prompt for replacement features. Let's walk through it line by line.

The first thing we do at line 600 is to open the output file (F1$). Note we are using buffer number 2 because we can't have the same buffer open for input and output in sequential files at the same time.

As in the earlier section, lines 610 and 620 open the input file for read and immediately check for end of file. Again, if we encounter EOF we go to do the end processing because we are done.

Line 640 says that T will show the occurrence of our search string in A$. It also says that if we didn't find the search string in A$ (T=0) then print the entire A$ out into our new file, increment the lines examined counter (LC), and go back to 620 to check for EOF and input another line to be examined.

Line 650 simply initializes variable Q to zero. It might contain something from the last time around and we want to be sure it is cleared. Obviously, we are assuming at this point that the search string was found in line 640 and that T contains some value other than zero. Keep in mind that if T was other than zero, the entire remainder of line 640 would not even be looked at by the program.

The code starting at line 660 and ending at line 790 is a loop within a loop. It's just a little bit tricky, so we'll go through it carefully. The first thing that probably strikes you is that in line 680 it says that if P is zero to goto 800, and in 800 it talks about X$, and you are probably wondering where X$ came from all of a sudden. *(Just like I did in trying to explain it -Ed)* Here is the trick: We wouldn't even be here if the search string were found in A$, so P cannot possibly be zero the first time around this loop. The reason for this "funny" code is that it is used to find *more than a single occurrence* of the search string in A$. Let's assume the first time through the loop for discussion purposes, where P cannot be zero yet.

Line 660 sets P equal to the character position in A$ where the search string is found *starting to look for it at the first character position* (Q+1). So now variable P has the position number of our search string within A$.

In line 670 we let Q equal the value of that position plus one. Why? If we didn't, and there were more than one occurrence of the search string in A$, we would keep finding the same one over and over. By advancing one past the last occurrence, we insure that we won't find that particular one again.

Now we run into line 680, where if P is equal to zero we go to line 800. As mentioned earlier, this is for second and subsequent occurrences of the search string in A$. When P is zero, it says there are no more search strings to be found in A$ and line 800 does its stuff and we go back for another input line.

In line 690 we print A$ and then print our little arrow below the occurrence of our search string.

Line 700 is a decision line. If A is less than three (it can only be 2 or 3 at this point) we go to line 760. Let's take that case first. In 760 we let L$ equal A$ up to (but not including) the occurrence of our search string. Then in the same line we increment the "replacement made" counter (LE) by one. In the next line, 770, we make RT$ equal to the entire right hand portion of A$, again not including our search string. We did this by taking the right portion *less the length of our search string* (L). Where did L come from? Way up in line 400 when we input our search string, at the end of the line we established L as the length of that search string.

Now in line 780 we have two pieces of string, one up to the search string and the other from after the search string to the end. Essentially, it's A$ without the search string and it's in two pieces (L$ and RT$). So now in line 780 we can put it back together again, this time inserting our replacement string between the two pieces. We call the new string we just created X$, and it is made up of the left string (L$) plus the replacement string (R$) plus the right string (RT$). Still in line 780, we then set P equal to zero and let A$ equal our new version of A$ called X$.

In line 790 we check to see if Q is less than the length of A$, because if it is, there may be more than one search string in A$, and if it is, we go back to line 660 to find that next occurrence. If Q were equal to or greater than the length of A$ we are done, so line 800 prints the new line (X$) through buffer 2 to the new file, then prints the line on the screen, and then updates LD, which keeps track of lines containing the search string. At this point X$ and A$ are equal, so we could have used either of them. Line 810 checks to see if we want to print this line, and if so it does. We then go back for another input line.

Back in line 700, if A had been equal to 3, (the prompt for replacement feature) we would have printed the prompt to replace or skip on the screen. Then line 720 would loop on itself until we press any key. If the key we press is an R or an r we would go and do what we just did - replace the search string. If it were an S or an s (line 740) we make X$ equal to A$ and jump down to 790 where we check for multiple occurrences and print the line to the file and screen as before. If we press any other key line 750 takes us back to 710 to re-print the prompt on the screen.

The end processing from line 850 on, is set up so that depending on what menu option you choose, unnecessary information is not included. Option 1, for example in line 870, shouldn't be talking about replacements so the goto 910 takes us right around them to the end.

Note that menu option 1 only shows the first occurrence of the search string even if there are more. The other two options, however, will find all such occurrences.

We keep a compiled version of this program on our system disk and use it frequently, many times only to find where certain variables occur in a program. We think you will find it useful too. ■

Listing for GW-BASIC
Changes for others follow.

```
100 REM * REPL.BAS * SEARCH & REPLACE PROGRAM WRITTEN FOR CODEWORKS
110 REM * MAGAZINE 3838 S. WARNER ST. TACOMA, WA 98409
120 REM * (206) 475-2219 VOICE AND (206) 475-2356 300/1200 MODEM
130 REM * PLEASE DO NOT REMOVE THESE CREDIT LINES.
140 'CLEAR 2000 ' Use only if your BASIC needs to clear string space.
150 CLS ' This is a clear screen command, change to suit your BASIC.
160 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
170 PRINT"        S E A R C H  &  R E P L A C E   U T I L I T Y"
180 PRINT"       with options for prompts and line printer output"
190 PRINT STRING$(60,45)
200 PRINT
210 PRINT"This program will search any file that has been saved"
220 PRINT"in ASCII format. This includes word-processor files as"
230 PRINT"well as program files in any language so long as they"
240 PRINT"have a carriage return at the end of their lines and"
```

```
250 PRINT"are in ASCII and the lines are less than 255 characters."
260 PRINT"This program does not destroy your original file,"
270 PRINT"but writes a new one with a '.NEW ' file extension."
280 PRINT
290 INPUT"Press ENTER to continue";XX
300 CLS
310 INPUT"What is the ASCII file name to work with";F$
320 IF INSTR(F$,".") THEN Z=LEN(F$):TP$=MID$(F$,1,Z-4):F1$=TP$+".NEW"
    ELSE F1$=F$+".NEW"
330 PRINT
340 PRINT"1 - Find and display only
350 PRINT"2 - Find and replace automatically  (creates "F1$" )"
360 PRINT"3 - Find and prompt for replacement (creates "F1$" )"
370 PRINT
380 INPUT"Enter the number of your choice";A
390 IF A<1 OR A>3 THEN 380
400 INPUT"Enter string to search for";S$:L=LEN(S$)
410 IF A>1 THEN INPUT"Enter replacement string";R$
420 INPUT"Output to line printer also, (Y/N)";Y$
430 IF INSTR(".Y.y.YES.yes.Yes.","."+Y$+".") THEN PR=1 ELSE PR=0
440 IF PR=0 THEN 470
450 IF A=1 THEN LPRINT"FILE ------> ";F$;" Searching for:
    ";S$:LPRINT" "
460 IF A>1 THEN LPRINT"FILE ------> ";F$;" Searching for: ";S$;"
    Replacing with: ";R$:LPRINT" "
470 ON A GOTO 500,600,600
480 '
490 ' ----- The Find and Display only loop
500 OPEN"I",1,F$
510 IF EOF(1) THEN 850
520   LINE INPUT #1,A$
530   P=INSTR(A$,S$):LC=LC+1
540   IF P=0 THEN 510
550   PRINT A$:PRINT TAB(P);CHR$(24):CH=CH+1
560   IF PR=1 THEN LPRINT A$
570 GOTO 510
580 '
590 ' ----- The Find and Replace (auto and prompt) loop
600 OPEN "O",2,F1$
610 OPEN "I",1,F$
620 IF EOF(1) THEN 850
630   LINE INPUT #1,A$
640   T=INSTR(A$,S$):IF T=0 THEN PRINT #2,A$:LC=LC+1:GOTO 620
650   Q=0
660   P=INSTR(Q+1,A$,S$)
670     Q=P+1
680     IF P=0 THEN 800
690     PRINT A$:PRINT TAB(P);CHR$(24)
700     IF A<3 THEN 760
710     PRINT"(R)eplace or (S)kip"
720       D$=INKEY$:IF D$="" THEN 720
730       IF D$="R" OR D$="r" THEN 760
740       IF D$="S" OR D$="s" THEN X$=A$:GOTO 790
```

```
750      GOTO 710
760      L$=LEFT$(A$,P-1):LE=LE+1
770      RT$=MID$(A$,P+L)
780      X$=L$+R$+RT$:P=0:A$=X$
790    IF Q<LEN(A$) THEN 660
800    PRINT #2,X$:PRINT X$:LD=LD+1
810    IF PR=1 THEN LPRINT X$
820 GOTO 620
830 '
840 ' ----- End of processing routine
850 INPUT"Press ENTER to continue";XX
860 CLS:PRINT LC+LD;" lines were examined in ";F$
870 IF A=1 THEN PRINT CH;" lines contained ";S$:GOTO 910
880 PRINT LD;" lines contained ";S$
890 PRINT LE;" replacements were made."
900 PRINT"The new file: ";F1$;" includes the changes made."
910 CLOSE
920 END
```

Change lines for Tandy I/III

```
Changed->100 REM * REPL/BAS * SEARCH & REPLACE PROGRAM WRITTEN FOR CO
DEWORKS
Changed->140 CLEAR 2000 ' Use only if your BASIC needs to clear strin
g space.
Changed->270 PRINT"but writes a new one with a '/NEW ' file extension
."
Changed->320 IF INSTR(F$,"/") THEN Z=LEN(F$):TP$=MID$(F$,1,Z-4):F1$=T
P$+"/NEW" ELSE F1$=F$+"/NEW"
Changed->550    PRINT A$:PRINT TAB(P);CHR$(133):CH=CH+1
Changed->690      PRINT A$:PRINT TAB(P);CHR$(133)
```

Change lines for Tandy II/IV

```
Changed->100 REM * REPL/BAS * SEARCH & REPLACE PROGRAM WRITTEN FOR CO
DEWORKS
Changed->270 PRINT"but writes a new one with a '/NEW ' file extension
."
Changed->320 IF INSTR(F$,"/") THEN Z=LEN(F$):TP$=MID$(F$,1,Z-4):F1$=T
P$+"/NEW" ELSE F1$=F$+"/NEW"
Changed->550    PRINT A$:PRINT TAB(P);CHR$(133):CH=CH+1
Changed->690      PRINT A$:PRINT TAB(P);CHR$(133)
```

ORDER FORM

Issues

- ☐ Renew Subscription $24.95
- ☐ NEW Subscription (starts with Nov/Dec 1987 issue)..... $24.95
- ☐ All 1st Year CodeWorks issues (Issue 1 through Issue 7) $24.95
- ☐ All 2nd year CodeWorks issues (Issue 8 through Issue 13) ... $24.95

☐ **GIFT Subscription**

Please give both your name and the name and address of the person who will receive the gift.

Clip or photocopy and mail to **CodeWorks,**
3838 S. Warner St.
Tacoma, WA 98409

Computer type:

Comments:

Diskettes

- ☐ 1st year programs on disk (specify type below) $20.00
- ☐ 2nd year programs on disk (specify type below) $20.00

- ☐ PC/MS-DOS 40 track DSDD
- ☐ CP/M 5 1/4 inch (specify format and computer type here _____

_____

- ☐ TRSDOS Model I 35 track SSSD
- ☐ TRSDOS Model III 40 track SSDD
- ☐ TRSDOS Model IV 40 track SSDD

HOW?

- ☐ Check/MO enclosed
- ☐ Bill me later
- ☐ Charge to VISA/MC_____ *Exp* _____

TO: ***(Please print clearly)***

Name _____

Address _____

City _____ *State*_____ *ZIP*_____

Charge card orders may be called in (206) 475-2219 9 am to 4 pm weekdays, Pacific time. 188

Index Update

NFL87.bas, reference, issue 14, page 2
Card.bas, reference, issue 14, page 3
Misc, CLEAR, CLS and CHR$(12), issue 14, page 3
Misc, ESC and CTRL on Tandy machines, issue 14, page 4
Pay.bas, reference, issue 14, page 4
Misc, using FIELD for string variables, issue 14, page 4
Beginning BASIC, on goto and on gosub, issue 14, page 5
Beginning BASIC, restricting input to numbers, issue 14, page 6
Notes, the need for closing quotes in literals, issue 14, page 6
Puzzler, answer to issue 13 puzzler, issue 14, page 6
Notes, print USING, quotes and calibration lines, issue 14, page 6
Misc, global definition of variables, issue 14, page 6
Pay.bas, correction, update, issue 14, page 7
Ecal.bas, main program, issue 14, page 8, an event calendar
Cwindex.dat, additions to this index for 1st 2 years, issue 14, page 16
Slot.bas, main program, issue 14, page 17, a slot machine simulation
Random files, article, recap of year's efforts, issue 14, page 24
Ranindex.bas, update, to work with ranprint.bas, issue 14, page 24
Randemo5.bas, samples of .MAP and .SCN files, issue 14, page 25
Ranprint.bas, sample report formats, issue 14, pages 27, 28, 36
Randemo5.bas, main program, issue 14, page 28
Ranprint.bas, main program, issue 14, page 33, report generator
Random files, notes on Tandy I/III changes, issue 14, page 37
Random files, notes on Tandy IV changes, issue 14, page 38
Download, what's happening there, issue 14, page 40

If you are using Qkey.Bas to keep a running index of CodeWorks articles and notes, these are the changes to bring that index up to date through the last issue.

Happy New Year

from

everyone at

CodeWorks

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
US Postage
PAID
Permit No. 774
Tacoma, WA

CodeWorks

Issue 16 **Mar/Apr 1988**

CONTENTS

CodeWorks

Issue 16 **Mar/Apr 1988**

Editor/Publisher
Irv Schmidt
Associate Editor
Terry R. Dettmann
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Advisor
Al Mashburn

Editor's Notes

It's amazing to see what diverse uses computers are used for. Take the last Super Bowl (please) - where the announcers could identify a player and immediately come up with: "third year draft pick out of Clemson." Or, "he's the youngest player in any Super Bowl to have made two consecutive touchdowns." Meaningful stuff, that. But the one that really got me was (I think) a game between the Bears and the Vikings where the Bears' coach was having telephone problems. After this had happened three times, a nice little box appeared on the TV screen telling everyone:

TELEPHONE PROBLEMS
Vikings 0
Bears 3

Sometimes you learn more than you really wanted to know about things.

All of us, when we first got our computers, dreamed of stuffing all sorts of various and sundry things into it. Sometime later, most of us found out that the stuff wasn't very orderly and that it took an inordinate amount of time to enter more stuff and to find anything after we had entered it. The realization of that seems to be the first step in growing up with your computer. With it, come schemes of organization and ease of input, not to mention quick and accurate searches and ordered displays and printouts.

Basically, a computer should compute. Even when it does compute, there are times (like balancing a checkbook) where using one is like shooting a mouse with an elephant gun. Maybe keeping food recipes in your computer is like that too, although it can be argued that if the recipe is for six people and you want to make it for three the computer can calculate the difference in the amounts of the ingredients. (But if the original recipe calls for one egg, how do you cut it in half? And is there a "half-pinch" of salt?)

Thoughts like these came to me recently when three different program ideas were suggested by real-world people. One of them wanted a recipe file. The second wanted to keep vehicle maintenance records for a few vehicles, and the third wanted the computer to act like a cash register at a local gravel pit.

In spite of the fact that the recipe program doesn't do much, I do like the idea. After all, eating well is the best revenge, as they say. Perhaps we should try one of them in CodeWorks sometime and see how it goes over.

The vehicle maintenance program would be a good thing only if it produced some information you couldn't get too easily manually - like average cost per mile of travel and automatic notice of when different types of maintenance are due. Maybe it could even figure out when a given vehicle has become a maintenance hog and should be dumped for a newer model.

The gravel pit program leaves me in a quandry. The request was for a computer that acted like a cash register that recorded a transaction and printed the bill - all in twenty seconds! It would take the printer longer than that just to print out the bill. My first reaction to this was to compile the program and get a 386 computer to boot, just to get the speed. It's obvious that I must get a lot more information about this one.

All of this tends to point out where some of our program ideas come from. It's not always exactly what was suggested, but what the suggested program leads to. Other ideas are more straightforward. A reader recently made a very simple request: How about a biorhythm program that prints the curves on the printer? It sounded interesting, so we checked out a book on the subject and two days later had a working program. It will probably appear in these pages sometime soon.

And that's the way it goes. Some take a little longer than others, but we do try to get around to most of your requests - it is, after all, why we are here.

Irv

Forum

An Open Forum for Questions & Comments

I recently had need for an ArcSIN function while writing some code involving trigonometric operations. Upon searching my GW BASIC manual I found that ArcSIN, ArcCOS and ArcTAN are not available in my version. ArcSIN, for example, is one form of an expression meaning "the angle whose Sine is...," and is quite useful when working with parts of circles and angles in general.

To solve this problem I devised the bit of code (see Arcsin.Bas listing) which is enclosed, and used it as a subroutine. Upon examination it will be seen to be a cascaded "sieve" sort, a technique which I have found to be most valuable when searching for a single item in a large (numerical) field. Using four sequential For...Next...else loops it reduces the number of brute force tries from a possible 10,000 to no more than 35, reflecting a considerable saving in machine time. The reported angle is to 0.01 degrees sufficiently accurate for most practical purposes, but the program is readily expandable if better data is available.

```
100 REM * Arcsin.Bas * John Omoh
    undro *
110 INPUT"ARCSIN 'Angle' =";X
120 FOR N=0 TO 90 STEP 10
130   A=SIN (N*2*3.1416/360)
140   IF X>A THEN 150 ELSE 160
150 NEXT N
160 FOR M=0 TO 10
170   A=SIN ((N-10+M)*2*3.1416/3
      60)
180   IF X>A THEN 190 ELSE 200
190 NEXT M
200 FOR L=0 TO 1 STEP .1
210   A=SIN ((N-10+M-1+L)*2*3.14
      16/360)
220   IF X>A THEN 230 ELSE 240
230 NEXT L
240 FOR K=0 TO .1 STEP .01
250   A=SIN ((N-10+M-1+L-.1+K)*2
      *3.1416/360)
260   IF X>A THEN 290
270   PRINT "ArcSin "X"="N-10+M-
      1+L-.1+K-.01"degrees"
280   IF X<A THEN 300
290 NEXT K
300 END
```

To modify ArcSIN to ArcCOS, merely change "SIN" to "COS" wherever it appears, and change the direction of each > or <. ArcTAN only requires changing "SIN" to "TAN." No effort has been made to reduce the few numbers to simpler quantities for I feel that this presentation is as good as a remark to anyone wanting to use or change the method for other purposes.

I hope that this little routine will prove useful or that the sieve concept will be valuable.

John Omohundro
Willcox, AZ

Re: Judy Nolting letter, Issue 14, regarding CLS and the H89 Heath computer.

The suggested use of PRINT CHR$(12) will only move the cursor down one blank line on the H89-H/Z19 terminal. For this command to blank the screen use: For I=1 to 24:PRINT CHR$(12):NEXT I. This will blank the screen but will leave the cursor in the lower left corner. A method that works faster and homes the cursor is as follows. At the start of your program type in this line: CL$=CHR$(27)+"E". This sets CL$ to clear the screen and home the cursor. Then substitute all CLS commands with PRINT CL$.

Jon B. Grimes
Pullman, WA

Just a short note of appreciation for CodeWorks and a bit of information on the Poker programs you have published.

I recently acquired Microsoft QuickBASIC Version 4 which I think is an excellent form of BASIC - I'm still learning it - it takes a while even if one has used BASIC previously.

In the Poker programs the variable DO is present. After I loaded the program into QuickBASIC I got an error message at the lines containing DO which stated "Expected - Variable." In GW BASIC that is okay but in QuickBASIC DO is an expression...

Clyde W. Preble
Mill Valley, CA

You are right, DO is a reserved word in QuickBASIC. Change it to DU. You will also find that in using QuickBASIC you will get ND (array not defined) errors when compiling Poker. These errors are not "fatal" errors, only warnings. This is

because regular BASIC allows you to use arrays of less than 10 without explicitly dimensioning them. QuickBASIC, however, sees an array and then immediately checks for the dimensioning statement. If you get the ND error and the array is less than 10 you can safely ignore the warnings, otherwise the array must be properly dimensioned.

...Beginning BASIC on page 6 of Issue 15 demonstrates the use of right-set to obtain the proper sort of numbers in string form. I wish to point out that if you are using a simple sort written within the BASIC program there is another way to do this. For example, in the simple sort routine on page 8 (Issue 15, lines 270-330) if the data were strings then the line 290 and 300 would read as follows:

```
290 IF A$(I)<A$(T) THEN 320
300 Z$=A$(I):A$(I)=A$(T):A$(T)=Z$
```

Numeric strings will not sort correctly. However to correctly sort numeric strings simply change line 290 to read:

```
290 IF VAL(A$(I))<VAL(A$(T)) THEN 320
```

I found this method to be necessary when sorting by percent error. A number in exponential format will not sort properly as a string. The string 5E-2 will appear before 5E-3 although numerically the reverse is true...

Robert A. Hood
Bremerton, WA

I cannot get Ranprint as listed in Issue 14 to work correctly...it appears that something is wrong with the values in line 2620 that causes the items to print incorrectly. I have a PCjr. Can you tell me what is wrong?

Howard W. Clother
North Java, NY

When creating any .PRT file for use with Ranprint.Bas, you must leave space in your report format for the length of each field you want to print plus one. If the field you want to print is, for example, NAME, and that field is 17 characters long, then leave 18 spaces between the pound sign for that field and the pound sign for the field that follows. If you don't leave enough space, Ranprint will skip that field entirely and print the following fields out of order.

In using the Randemo5.Bas program on page 28 of Issue 14, I failed to use the zero option (to quit) because I was rushed to turn off my computer. When I came back, I found the buffer entries had not been closed out. I was not happy about that but this addition to line 1065 takes care of this problem. It is: CLOSE 1:OPEN"R",1,FD$. Thank you for an interesting program.

(And in another letter on the same program.) In debugging my own typed in version of Randemo5 I ran across lines 600 through 740. In particular one line really interests me. That is line 615 which is IF LEN(C$)>1 THEN GOSUB 700. As I understand the INKEY$ function, it allows for one character and no more as line 610 is set up for the C$ variable. Thus, it is not possible for C$ from line 610 to have more than one character in it for line 615 to analyze. Therefore, line 700 is never GOSUBed and lines 700-740 are never used. I feel like I am missing something here but I know not what...

Robert B. Lockhart
Yakima, WA

The program was written so that it would work equally well on many versions of BASIC, without making numerous exceptions to the code. The specific lines you refer to, i.e., 615, 700 and 710 to 740, are there to detect the arrow keys in GW BASIC. In MS-DOS, GW BASIC, the arrow keys do return two characters. INKEY$ does pick up the entire set of digits in that case. Line 610 simply says that if there is a length of more than one, we must be dealing with GW BASIC and to go to the subroutine at 710 to find out which arrow key was pressed. The lines are not accessed for other than GW BASIC.

I am an 80-Micro orphan. When they cut us adrift, they at least gave us your name as a source of early Tandy information...I may be able to make some contribution to the dinosaur cause. I have a complete set of all 80-Micro and many spare copies plus copies of many other early vintage computer magazines. I intend to keep one set, but if someone wants the spares or copies of any articles they may have them for postage costs. If you or your subscribers are interested, let me know and I will send you a complete list...

R. H. Salisbury
5626 Jordan Avenue
El Cerrito, CA 94530

Thanks for the offer. Interested readers may contact Mr. Salisbury directly.

I downloaded (Ecal.Bas) then ran it through LeScript for editing. That was a mistake. You might want to warn others. LeScript removes the

extra spaces in the printout lines! I had to go back and edit them in again with my BASIC editor...

James C. French
Dundee, IL

I have just finished alterations on your program Slot.Bas. I'm one of those who sternly dislike games and have yet to play any on my computer. I find them all quite boring in about a minute or less. However, I thought you did a pretty good job with Slot.Bas. But seeing as how I get bored, I decided to make your game all automatic. I can run it when I'm not busy with other computer stuff and see how the wheels of chance turn. Besides, I love altering or changing programs, probably more than running the refined product after I'm done...

Richard S. Burwell
Gorham, NH

Many CodeWorks readers have stated they don't like games. Personally, I would rather write a game program than play one. But what about simulations? When does a game become a simulation, and what's the difference? I kept thinking about that while writing Broker.Bas in this issue. I would consider it to be a real life simulation, but it has the feel of a game too. Perhaps a simulation is a game that applies to the real world. Is flying a flight simulator a game, or is it an effective way of conditioning a pilot to the real thing? Your comments concerning games versus simulations would be welcome.

I don't like TRSDOS 6.3 - is there any way to get 6.2.1 to accept (the year) 1988?

Susan Hope Dunham
Ridgefield, CT

No, and we can't imagine why it was written with that restriction in the first place. We had thought that once in BASIC you could use RIGHT$,2 to change the year in DATE$, but that only gives a syntax error. This gives a whole new meaning to the phrase: "time bomb."

I see in your reply to Virginia Erickson (Issue 15) that the CodeWorks disk is supposed to work on an IBM computer. We have both a Model IV (Tandy) and a Corona 512K. The disk works well on the Model IV, but I can't get it to load in the Corona...

Eugene W. Gall
Fayetteville, OH

Different computer types use different formatting schemes on their disks. Fortunately, all MS DOS computers use the same, or a compatible, format and so a diskette made for them will work on any MS DOS computer or clone. Model IV has its own unique diskette format, as does the Model III. A diskette made for these computers will not be recognized in an MS DOS machine. If you will notice our order blank on page 39 of each issue, you will see that we offer our programs on several different formats for the various machines we cover. Aside from the different format, we make the few code changes to make the programs function properly on each particular machine.

My delight in and respect for CodeWorks is represented by the fact I am reupping my subscription and requesting a diskette. At age 69 I bought myself a present this past Christmas, a real computer, a gadget which has done more than anything else to stir up the neural synapse junctions in my feeble brain. And your magazine does more to teach than any other publication I have tried... Keep up the good works.

Frank C. Smith
Gig Harbor, WA

Thanks, and you are just one of many of our subscribers who is into retirement. We have heard from many of our readers who got into computing as a retirement hobby and are enjoying it very much.

Almost as soon as I finished reading issue 11 I had a need for Diff.Bas. My sister-in-law and I exchange programs and I made some improvements to a game she had sent. There weren't many changes, just some polishing really. I didn't want to send a disk with that minor of an update, so I used Diff.Bas to show her the lines of code I had changed. I liked the summary that prints to the screen and wanted it to be on the printout, without the clutter you can get from a screen print. I modified Diff.Bas to print the summary. Keep up the good work!

Nade E. Strickland, III
Irmo, SC

That's what we like to hear. We have always hoped that our programs would be a starting point for your own modifications and embellishments.

I would like to suggest that a future article cover the area of printing the new Postal graph representation of the ZIP codes, so that we may add the program to the Addressr/Bas program of issue 11. I have not been able to decode the algorithm for what represents the letters. The printing would be a good exercise in pixel printing.

H. Lawrence Abbott
Wyomissing, PA

We assume you are referring to the little string of bar code under the address? If so, we checked several different letters coming to us from various places and found this: The city, state and ZIP are apparently represented there. There are more spaces for codes unknown to us. The Post Office tells us that this code is put on at the office of entry and represents a routing code for internal Post Office use. Its placement on the piece of mail is also a matter of some importance. We got the impression from them that they don't want us to mess with it.

Football

Well, who would have guessed? The Redskins won it and the Broncos made a third unsuccessful attempt at being Super Bowl winners. And right here, deep in Seahawk country, there was one Redskin fan who was elated. He works in our building and has this smug smile on his face ever since that Sunday. Our Oracle didn't do as well this year as last - it picked slightly over 60 percent, down two or three from last year. Can we just blame it on the strike and let it go at that? Oh well, wait till next year.

Other stuff

The nasty streak of bad weather during January seemed to delay the delivery of the last issue. It had been put into the mail two days before Christmas, but arrived in the middle of January for many of you.

Aside from the weather, that issue was also trimmed so badly that in some cases the page number was cut off. We had to search through the entire press run, looking for acceptable copies to send, and what's left won't even make good sample copies. We hope this issue turns out better, since it is being run with a different printer.

The type wheel we use to make our listings for the magazine finally gave out. The new one, even though it's the same stock number, seems to look slightly different. The old wheel seemed to lean a little more toward the bold, where this one seems a little thinner. We hope it doesn't cause any problems.

Has Spring sprung properly in your neck of the woods? It's almost time to start thinking about outdoor projects again - like maybe the pump program in this issue. I am still amazed and amused by it all.

Do you remember the section in CodeWorks first one or two issues that was labeled "Sources?" Well, that sort of fizzled out quickly - it was mostly press release stuff and it was hard to get something in there that was applicable to you readers. So we quietly dropped it.

Now we are thinking of bringing it back again, but with a difference.

Since 80-Micro dropped support of the early Tandy models (and their advertisers too) we have been hearing a lot from people who say there is no place left to get information about those machines to the people.

There are still some producers of good software for those models out there. Not only that, but they have various amounts of stock left and would like to move it - even possibly at reduced, close-out prices. They may well have new products they would like to move too.

We have always said we don't want to get into the advertising game. There are a whole lot of reasons for that - the biggest reason is that we simply don't have enough circulation to warrant charging anything worthwhile for a page of ads. The printing and shipping of such a page would be more than we could charge for it.

However, there is a way we can serve you, the advertisers and ourselves. How about that, a three-banger solution! Here is how it could work:

We use a couple of pages in the back of the magazine to list sources of software (or even hardware, for that matter). It costs the advertiser nothing to be listed there. All he must do is keep us abreast of what he has, what is sold out and what new items he wants listed.

That's how the advertiser makes out. You get the benefit of finding items of interest to you in those pages, and perhaps you will even want to purchase one or two of them.

How do we make out? Simple. The items listed are all ordered through us. The advertiser agrees to collect the orders from us and we take a small cut of the published price of the item. In other words, the advertiser only pays for his space when an item is sold.

We don't think that's a bad idea. We can certainly use the extra income, since subscriptions and diskette sales are our only other source of income. There are times when the finances are nip and tuck, and this just might make a difference. What do you think?

Thanks for all the fine input, and keep those cards and letters coming.

Irv

Compiled BASIC

What a Compiler can do for you

Staff. Since this was written we have received the Microsoft QuickBASIC 4.0 compiler. It is a major update to previous releases and presents some new and challenging capabilities.

With the complexity of today's computers it is easy to overlook the fact that inside all of them is a CPU (Central Processing Unit) that only responds to different patterns of switches that are either on or off. Most of us are familiar with the name of the CPU - Z80, 8086, 80286, etc., because it usually tells what type of computer we have. Each of these central processing units has its own set of instructions (patterns of ones and zeros) that will make it perform some specific function. It is easy to see that programming such a CPU in its own set of one-zero (switch on-off) bits would be the world's biggest yawn, not to mention boring and error prone.

Because of this, the so-called "higher level" languages were developed: FORTRAN, COBOL, BASIC, and others. These languages allow you to speak to the computer in more or less human terms rather than in ones and zeros.

But the CPU still wants to see ones and zeros, and so there had to be a way to convert the high level language back down to what the CPU could understand. BASIC is an interpreter, which means that it takes one line of code at a time, converts it to CPU-readable code, and then executes it. It follows, then, that in order to run a BASIC program you must have BASIC loaded in memory so that it can do the machine code conversion (interpretation). This process can be a slow one, especially where the interpreter has to re-convert the same code over and over, as for example, in a loop. It also has to build some sort of a list of variables, and then search through that list every time a variable is called for. You don't see this happen when you run a BASIC program, but it, and more flags and pointers, are all there.

A BASIC compiler is a program that takes a BASIC program and converts it to machine readable code ahead of time. It creates a file from the original BASIC program (the "source") which is called the "object" file. The object file is then "linked" with a library of routines, supplied with the compiler software, that handle keyboard scanning, video processing and the like. It then converts the resulting code into machine executable form (the /CMD or .EXE file). This code can then be executed directly from the DOS ready prompt, and BASIC doesn't even need to be near the machine. You can see why we had to link the object file to the library. Where BASIC would have handled all the housekeeping before, it is no longer there and something has to do it.

Where BASIC would have had to search a list of variables to find the right one, the compiled version has a direct memory address for that variable because that was all set up during the compile process. Other processes work in a similar fashion, all resulting in execution speed far exceeding that of BASIC. In fact, speed is one of the main reasons to compile a BASIC program. Depending on the program itself, you can expect from two to 30 times the execution speed with a compiled program. Compiling does not increase the speed of disk input or output, nor does it help any program that has intensive user input through the keyboard.

The executable file created by the compiler cannot be listed or edited. If you look at it with a program that displays what is in memory you will find what looks like a bunch of meaningless numbers. If you are preparing programs for sale and don't want your customers to fool around with your code, this is a good way to do it. There's not much they can do except run it, while you have the original source code. This, of course, brings up another question. How do you modify a compiled program? You can't. All you can do is make the modifications in your original source code and re-compile the whole works again. All of which says you should keep a copy of your source code!

How about space? Well, it used to be that the earlier compilers would compile into a much smaller space than the original BASIC program took. Somehow, the newer compilers manage to make the compiled version larger. As a test, before I sat down to write this article, I took Card1.Bas and ran it through the compile process using the Microsoft QuickBASIC 1.0 compiler. Saved in normal fashion, Card1.Bas took up 5721 bytes on the diskette. Saved in ASCII, it took about 20 percent more - 7195 bytes. After compiling it as a stand-alone executable file it took 41,622 bytes!

But...some nice things happened. It became fast. The sort and searches were lightning quick. And, before I compiled it, I set the dimensions of the arrays so that they would take up 64,000 bytes. If I had tried to run it that way in BASIC it would have given an "out of memory" error. (I tried it and it did.) But after compiling it, it worked just fine because the compiler sets aside 64,000 for data and another 64,000 for the program itself! This may not be true for just any compiler, but is for the QuickBASIC 1.0 compiler from Microsoft. (They are up to version 4.0 as of November 1987 and we are awaiting our update.)

Aside from compiling your own, already-programmed in regular BASIC, programs the new compilers have all sorts of sophisticated capability. They come complete with a new compiler BASIC that excludes some of the features we are all familiar with and add some new features that we are not familiar with. With the QuickBASIC compilers you no longer need to use line numbers. Or, you can if you want to. Or, you can mix line numbers and labels. It's pretty much up to you. You can create lines longer than 254 characters by using the underscore as a continue device. Some of the commands not accepted by QuickBASIC, for example, are: AUTO, BLOAD, BSAVE, CONT, DEF USR, DELETE, EDIT, LIST, LLIST, LOAD, MERGE, NEW, RENUM, SAVE and USR. Other commands may need to be modified in order for them to work, like: CALL, CHAIN, DEF type, DRAW, PLAY, RESUME and RUN.

Some of the new commands and functions are: COMMAND$, which returns the command line used to invoke the program. LBOUND and UBOUND return the lower/upper bound of an array. LOCK and UNLOCK control access by other processes to all or part of an opened file. REDIM changes the space allocated to dynamic arrays. (You can now have static and dynamic arrays.)

Several other functions have been enhanced. One in particular is the DEF FN function, which now supports both single and multiline functions. Which means that you can define some rather wicked functions that cover more than one program line - almost like a little sub-program in itself.

Although this was not intended to be a product review, it's beginning to sound like one. So, back to when you would use a compiler. Any time you have a program that is CPU bound, or any time you don't want to release the original source code. Or when you have a whole system of programs that all must work with each other, using common variables.

How do we use the compiler at CodeWorks? We have a compiled version of Plist.Bas on a system disk here that we use constantly to list the programs that we publish. It's just faster and handier that way. We have another program that takes text files intended to the typesetter and prints them out on paper for proofing (like the typesetter would). It not only tells us that we have embedded the proper typesetting codes in the text file, but also shows the bold, italic and bold-italic where we have inserted them. This saves expensive silver typesetting paper at the other end. The program ran terribly slow before it was compiled and now runs rather respectably. Whenever we have a program that seems a bit sluggish and is intended for publication, we run it through the compiler and note the difference in the article.

When Terry gets the Randemo series to some logical finish point, we fully intend to compile the entire set of programs into one, menu-driven system. That type of program would probably benefit the most from the compile process.

The size of the compilers today would hang out of both ends of the 16K we used to have. Somehow, as the space on diskette and in memory increases, the programs seem to increase in size too. Darn, I remember when an 8K BASIC was advertised in *Byte Magazine* and everyone drooled over it. ■

Pump.Bas

Simulating a Hydraulic Ram Pump

Irv Schmidt, Editor. Springtime, especially around April, is a good time to plan summer projects. Perhaps this one will strike your fancy.

In physics there is a first law of thermodynamics that essentially says, "You don't get something for nothing." The second law states, "What's more, you hardly ever break even."

Nature seems to be that way about a lot of things. So when I first encountered the hydraulic ram pump idea I was, to put it mildly, utterly amazed. I remember it well. It was about 15 years ago or so, and I was in bed with a rather feverish case of the flu. When it got a little better the first thing that struck me was that it was terribly boring, just laying around in bed. So my wife brought a bunch of books and chicken soup upstairs to me and among the books was a Whole Earth Catalog.

You really have to hand it to those guys. They put out one heck of a great catalog. Anyway, among the goodies in it I ran across the hydraulic ram pump. You have to remember these catalogs were put out in the 1970's when energy conservation was a rather big deal. I studied the little write-up they had in the catalog, looking for the trick - there simply had to be one. Then it gave the address for some manufacturers of these pumps, and curiosity getting the best of me, I wrote off for some literature.

Unbelievable is what it was. Here is a water pump that can lift water higher, much higher, than the fall of water into it. And as though that isn't enough, it does it without any external power! No electricity, no steam - not even a big rubber band.

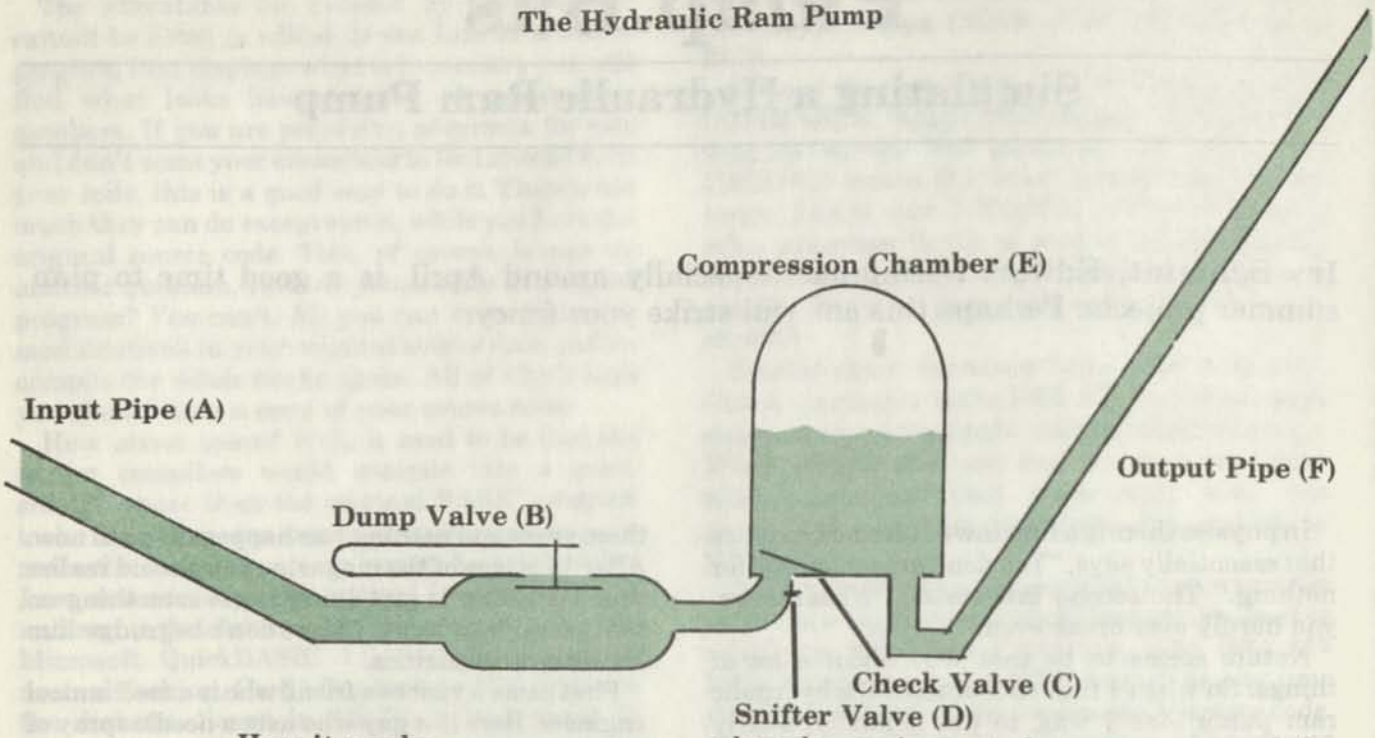
Well, many years pass, and now we are getting ready to create a publication called CodeWorks. On the wall in our office there is a chalk board, and on it is a list of ideas for programs. The first item on that list is a program to simulate the operation of the hydraulic ram pump. It's been there for almost three years and nothing has happened - until now. After 15 issues of the magazine you should realize that the Editor is just dying to get something on that pump into print. Please don't begrudge him his little eccentricities.

First came a visit to a friend who is a mechanical engineer. Here is a guy who uses a needle spray of water to cut through metal - he designed it himself. Yet he takes one look at the ram pump and says, "Gees, Irv, this would be a horrible thing to describe mathematically. It would take all sorts of differential equations just to describe the water falling through the input pipe - and the shock waves, wow! - and I don't know how you would simulate the action of those valves."

Oh well. Next came a visit to a pure mathematician, who said it couldn't work - but it looked interesting - and no, I really needed to see someone who understood the physics of fluid dynamics. Can't you just see the ad in the newspaper: Wanted. Fluid dynamics engineer to assist author in writing an article on hydraulic ram pumps? Even if such a person could be found, can you imagine how much he would charge? We would probably have more of a chance of getting President Reagan to help straighten out the organization and politics of our local computer user group.

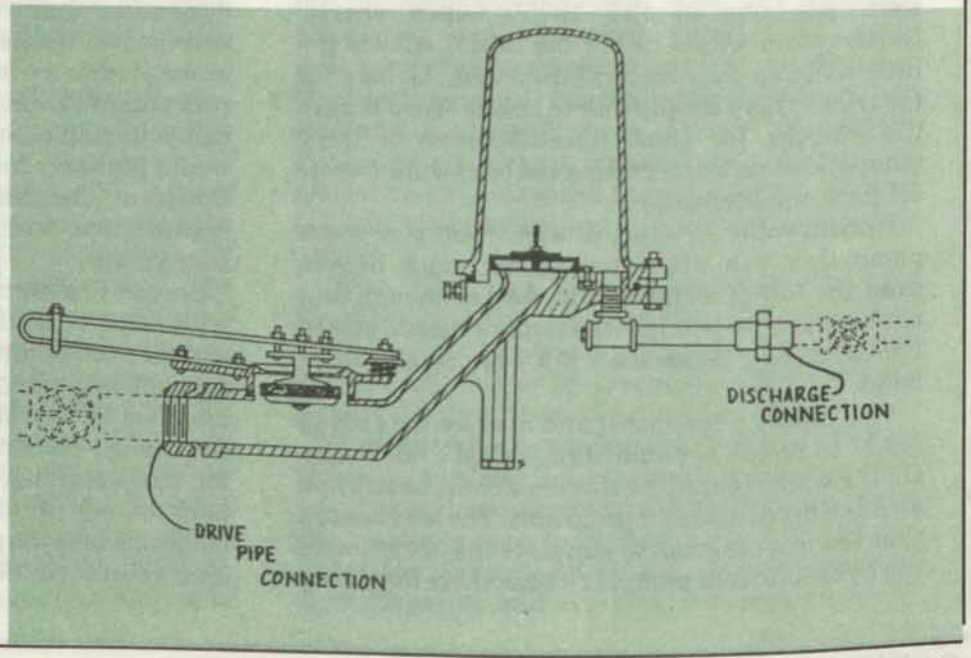
So the Christmas holidays rolled around and with some time off I decided it was time to put together the pump program. Instead, I spent the time off re-reading Asimov's Foundation trilogy, and then the two additional books he had added to the trilogy. The time was not entirely squandered, for while reading, the ram kept pumping in the back of my mind. So now, finally, during the doldrums following the holidays, it has taken form and substance. Somehow, the idea that nature

The Hydraulic Ram Pump



How it works

Water flowing down input pipe (A) flows out of the dump valve (B) until enough pressure is created to close the dump valve. At that point the inertia of the flowing water forces check valve (C) to open and water is forced into the compression chamber (E), compressing the air trapped in the chamber. When the air pressure overcomes the incoming water pressure, check valve (C) closes. The air then forces water up the delivery or output pipe (F). Meanwhile, the water in the input pipe (A) rebounds, causing negative pressure around the dump valve (B), allowing it to open again for the next cycle. This negative pressure also allows a small amount of air to enter the system through the snifter valve (D), which will replace any air lost in the compression chamber on the next cycle. The output pipe typically has a slightly smaller diameter than the input pipe does. The pump moves the maximum amount of water with the lowest number of cycles of the dump valve. The number of cycles is adjusted by varying the spring tension on the dump valve.



gives up something for nothing and lets you pump up to 1,400 gallons of water per day, every day, with no external power still excites me.

How it Works

Obviously, there has to be a source of flowing water. It can either be a stream with a steep drop or a dam can be constructed. Water is then directed down the drive (input) pipe into the ram. At the end of the input pipe there is a dump valve. This valve has enough tension to stay open until the force of the water builds up sufficiently to slam the valve shut. The water that spills through this valve is not wasted, as it simply flows back into the stream. When the spill valve (or dump valve) slams shut the water, which has considerable inertia, has no place to go except through the check valve at the bottom of the compression chamber. As the water enters the chamber, it compresses the air which is trapped in the chamber.

Soon, the compressed air pushes back against the pressure of the incoming water, and the check valve closes. The air then pushes water up the delivery (output) pipe. When the check valve closes, the incoming water "rebounds" back towards the input pipe, creating negative pressure (or suction). This allows the dump valve to open again, allowing water to spill through it into the stream and to build up enough pressure again to close the dump valve and start the cycle all over. Meanwhile, the trapped air is pushing water up the delivery pipe, not in spurts, but in a reasonably steady flow.

One additional valve (the "snifter" valve) is needed to complete the operation of the pump. This snifter valve is located just below the check valve in the compression chamber, and is simply a very small diameter hole. When water is being pushed up into the compression chamber this hole allows a small amount of water to escape through it, keeping it clean. During the "rebound" of the water back towards the input pipe, air is sucked in through this hole and replaces air that is lost in the compression chamber on the next cycle. Without the snifter, the compression chamber would fill up with water and the pump would bang like pipes will from the effect of water hammer. Without air in the chamber there would be nothing to push water up the output pipe.

The dump valve spring is made from mild steel strap, and usually has either a weight attached or a clamp fixed so as to vary the tension on the spring. It is interesting to note that the pump will move the maximum amount of water when the dump valve operates at the lowest number of cycles per minute, but not so slow that the pump stops working.

With a drop in height (head) on the input of only 18 inches, a typical pump with a two inch diameter input pipe will lift water to a height of 10 feet at the rate of about one gallon per minute. Multiply that by 1,440 to get gallons per day. The efficiency of these pumps typically runs at about 60 percent. Generally, adequate amounts of water can be lifted 25 feet for each foot of input head. In the absence of twigs or debris preventing the valves from working, these pumps will work continuously without attention.

According to N. G. Calvert, B.Eng., Ph.D., writing in *The Engineer*, page 597, April 1957, the hydraulic ram in its present form was the subject of a patent by Joseph Montgolfier in 1797. Manufacture of these pumps did not begin until 1820 when the Montgolfier patents were assigned to Josiah Easton. In 1803 Professor Eytelwein of Berlin acquired the Montgolfier ram and built two of his own (see Eytelwein, J. A., 1805, *Bermerkungen uber die Wirkung und Vortheilhafte Anwendung des Stosshebers* (Berlin)).

In this country, Rife Ram and Pump Works (316 W. Poplar St., Norristown, PA 19401) has been manufacturing ram pumps continuously since 1884 and since that time have sold more than 50,000 in every state of the Union and in many foreign countries.

After reading the literature, it is obvious that the ram pump is a "tinkerer's delight," costly if you play with the actual thing but not so with a simulation. Our simulation doesn't get into differential calculus, but works with a formula that defines the amount of water delivered in gallons per minute as: the amount of water supplied to the machine in gallons per minute times the head in feet all divided by the number of feet the water is to be lifted. All of that, of course, is multiplied by an efficiency factor. The efficiency factor is found by taking the output gallons per minute times the height the water is lifted, all divided by the supply flow rate in gallons per minute times the input head in feet.

Our program, Pump.Bas, roughly approximates the operation of the ram. It allows you to vary any of five input factors and see the effect on the output directly on the screen. Pressing upper case A through E will increase the associated value; pressing lower case a through e will decrease them.

Some limits, such as the spring tension, had to be set to keep values within a reasonable range.

Of course, if you have the urge to tinker with the real thing and have a suitable supply of water you want to pump, you can contact one of the following:

Edward Barberie, Box 104, Green Spring, WV 26772

Ce Co Co Chuo Boeki Goshi Kaisha, PO Box 8, Ibaraki City, Osaka, Japan

Davis Foundry & Machine Works, Rome, GA 30161

Rife Ram & Pump Works, 316 W. Poplar St., Norristown, PA 19401

J. W. Jolly, Inc., Holyoke, MA 01040

```
100 REM * Pump.Bas * Simulated Hydraulic Ram pump program *
110 REM * CodeWorks Magazine 3838 S. Warner St. Tacoma WA 98409
120 REM * (206) 475-2219 voice (206) 475-2356 300/1200 modem
130 REM * (c)1988 80-NW Publishing Inc. & placed in public domain.
140 '
150 'Generalized Locate/Print@ subroutine. Unremark as needed.
160 GOTO 230
170 LOCATE X,Y:RETURN ' MS-DOS, GW-BASIC
180 'PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy Models I/III
190 'PRINT@((X-1),(Y-1)),;:RETURN ' Tandy Models II/IV
200 'PRINT@(X,Y),;:RETURN ' Some MBASIC machines.
210 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN ' CP/M
220 '
230 ' Initialization
240 CLS
250 A$=STRING$(20,32)
260 A=1.5  ' input head in feet
270 B=5    ' input pipe length in feet
280 C=2    ' input pipe dia. in inches
290 D=1    ' simulated dump valve tension
300 E=10   ' output pipe vertical lift in feet
310 '
320 REM * set up the screen
330 CLS
340 PRINT TAB(20);"Hydraulic Ram Pump Simulation
350 PRINT
360 PRINT"UPPER case increases, lower case decreases values.
370 PRINT
380 X=5:Y=1:GOSUB 170:PRINT"(A) Input head (feet) ----------";
390 X=6:Y=1:GOSUB 170:PRINT"(B) Input pipe length (feet) ---";
400 X=7:Y=1:GOSUB 170:PRINT"(C) Input pipe diameter (in.) --";
410 X=8:Y=1:GOSUB 170:PRINT"(D) Dump valve tension ---------";
420 X=9:Y=1:GOSUB 170:PRINT"(E) Output pipe lift (feet) ----";
430 X=10:Y=1:GOSUB 170:PRINT"    Strokes per minute ---------";
440 X=11:Y=1:GOSUB 170:PRINT"    Input Gallons per minute ---";
450 X=12:Y=1:GOSUB 170:PRINT"    Efficiency (%) -------------";
460 X=13:Y=1:GOSUB 170:PRINT"    Output Gallons per minute --";
470 GOSUB 660 ' to print initial values on screen
480 '
490 REM * do the calculations
500 '
```

```
510 A1=(B*C)
520 D1=((D*10)/100)+.55
530 E1=((L*E)/(A1*A))*.66:IF E1=>1 THEN E1=1
540 L=((A1*A)/E)*D1
550 IN$=INKEY$:IF IN$="" THEN 550
560 IF IN$="a" THEN A=A-.1 ELSE IF IN$="A" THEN A=A+.1
570 IF IN$="b" THEN B=B-.1 ELSE IF IN$="B" THEN B=B+.1
580 IF IN$="c" THEN C=C-.1 ELSE IF IN$="C" THEN C=C+.1
590 IF IN$="d" THEN D=D-1 ELSE IF IN$="D" THEN D=D+1
600 IF D=<1 THEN D=1 ELSE IF D=>5 THEN D=5
610 IF IN$="e" THEN E=E-1 ELSE IF IN$="E" THEN E=E+1
620 GOSUB 660
630 GOTO 510
640 '
650 REM * update values on the screen
660 REM * first clear old values
670 Y=36
680 FOR X=5 TO 13
690   GOSUB 170:PRINT A$
700 NEXT X
710 REM * now put in new values
720 Y=36
730 X=5:GOSUB 170:PRINT A;
740 X=6:GOSUB 170:PRINT B;
750 X=7:GOSUB 170:PRINT C;
760 X=8:GOSUB 170:PRINT D;
770 X=9:GOSUB 170:PRINT E;
780 X=10:GOSUB 170:PRINT INT(120-(D1*100));
790 X=11:GOSUB 170:PRINT A1*A;
800 X=12:GOSUB 170:PRINT USING ".##";E1;
810 X=13:GOSUB 170:PRINT L;
820 RETURN
830 END ' of program
```

Change lines for Tandy I/III

```
Changed->100 REM * Pump/Bas * Simulated Hydraulic Ram pump program *
Changed->170 'LOCATE X,Y:RETURN ' MS-DOS, GW-BASIC
Changed->180 PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy Models I/III
Changed->690   GOSUB 170:PRINT A$;
```

Change lines for Tandy II/IV

```
Changed->100 REM * Pump/Bas * Simulated Hydraulic Ram pump program *
Changed->170 'LOCATE X,Y:RETURN ' MS-DOS, GW-BASIC
Changed->190 PRINT@((X-1),(Y-1)),;:RETURN ' Tandy Models II/IV
Changed->690   GOSUB 170:PRINT A$;
```

Broker.Bas

An investment simulation program

Staff project. If you have never invested in the markets but want to get the feel of it, this program could be what you are looking for. This simulation is based on actual data covering a ten-year period.

Have you ever wondered what it would be like to be a player in the big financial markets? Broker.Bas is a simulation that allows you to experience what it would be like. It gives you $10,000 and lets you play the markets for a period of ten years. Each calendar quarter you get the opportunity to make as many buy/sell transactions as you wish. You can study the board for as long as you like between quarters to plan your next strategy.

The data for Broker.Bas is based on an actual ten year period from the past. Those of you who have been following the markets for real may even recognize the period, but not knowing it adds to the validity of the simulation. Most of the data is straight out of the record books, but some, like the stock and bond prices, had to be carefully picked and sometimes interpolated. Most of the information was readily available at the local public library, however.

It is up to you to make as much out of your initial $10,000 as possible in 40 quarters while the market goes through typical changes. The program features a rather interesting "moving" quotation board, giving you the current prices or percentages for the Dow Jones Industrial Average, the Consumer Price Index, the Prime Lending Rate, the Money Market rate and the current stock and bond price per share. Another part of the display gives you the change information of prices this quarter as compared to last quarter. Yet another portion of the display shows your individual involvement, as well as what your broker is accumulating due to your transactions.

The command line at the lower portion of the screen is simple and error-trapped. It allows you to buy, sell or continue to the next quarter. Answer with anything but B, S or C, and it will ignore you. You don't even need to worry about upper or lower case. The program takes all of that into account.

When you buy, you are always asked to tell how many dollars worth. Don't use dollar signs or decimal points here. Anything but a number at this prompt will be ignored. When you sell, the program will show you how many shares you own and ask how many to sell. You can sell any number or all of them. If you really need to know how many shares you own during the simulation, you can tell it to sell, and then when it shows you the number of shares, sell zero of them. This prompt too, only allows number entries and will ignore letters or symbols.

The trick part of the program, of course, is the moving reader board. It moves from right to left, just like a real stock streamer and gets updated whenever you change quarters. Obviously, when you make transactions, the moving board stops momentarily because you can't execute code in two different places in the program at once. You can, however, make it *appear* to be in two places at the same time.

Since the program uses some rather interesting code, we will be going into a line-by-line description of many parts of it. We'll do that later in the "Program Details" portion of this article, so that those of you who don't care for that much detail can skip it. But first, let's go into using the program and some hints on staying alive in the markets.

Some helpful hints

It has been said that if you put your investment into one vehicle and leave it there over a period of years you will certainly lose it. You can prove this to yourself by putting all of your money into stocks, for example, and running through the 40 quarters. Dividing your investment into thirds and putting a third each into the money market, stocks and bonds and leaving it there for the ten

year period will give you similar results - not very productive. The financial markets reflect real life to an amazing degree - you must be involved and flexible in order to come out with your shirt on. Which means that you must move your investment from one vehicle to another as the conditions change.

The three vehicles (money market, stocks and bonds) do not necessarily move in the same direction at the same time. They are hardly ever all up at the same time, and we'll all be in trouble if they are all three down at the same time. The three economic indicators (the Dow, the Consumer Price Index and the Prime Rate) are given for each quarter to help you to decide what to do. They may or may not tell you, just as in the real world. Also like the real world, you have all sorts of "good" advice coming at you from your wife, your banker, barber, lawyer and various and sundry others. The program is set up so that you never get advice that doesn't apply, i.e., it won't urge you to buy bonds if you already own at least 100 shares. The advice is sometimes very valid, and sometimes is opposite of what you should be doing. It's up to you to decide when and who to listen to, or if you will listen to any of them at all.

It's hard to believe that you could actually have made over a half-million dollars in ten years from your original $10,000 investment. A half-million ought to be your goal, although we (with considerable inside information) have exceeded that figure by a sizable amount. We may just run a little contest and see who can get the most from the original investment *even though you all have the opportunity to study the data statements to your heart's content.* The period of history we happened to pick turned out to be a tricky one; one in which you will find some un-obvious trends.

If you are interested, you can send us your results, with what you did by year and quarter, and we'll publish the highest ones. Even though this program has the flavor of a game, we think you will find it both educational and intriguing.

Program Details

The program idea was floating around our office for several months - gestating. The idea lacked something. That something turned out to be the reader board. After spending a few nights on getting the information to fit and circulate, the rest of the program naturally evolved around it. The six pieces of information for each quarter are contained in data statements at the end of the program. Let's start at the top of the program and work our way down.

Lines 160-190 do some initialization. T(6) is set up initially to contain our "given" $10,000. In line 180 we establish two print strings for dollar amounts in X$ and X1$ to be used in PRINT USING statements later on. Line 190 declares variables I, J and U as integers. They are loop indices, and making integers of them speeds things up a bit.

Our general purpose locate/print@ subroutine sits between lines 250 and 310. Use the line that fits your computer by un-remarking the appropriate line and remarking all others.

Next, we read the data statements to see how many quarters there are. Variable N in line 370 will contain that number. But we know how many there are - why do this? Because it allows for some flexibility in that you may want to put additional information in later, or even take some out. Because we want to read the data again as the program progresses, we need to RESTORE the data pointer in line 380.

In line 410 we read in the first quarter data. This is so that we have something to compare the second quarter data to for the change information.

Next we clear the screen and put up the usual CodeWorks heading and a description of what the program does. Since there is not enough space on the screen for all the information you may want, the option is given to see a second screenfull in line 580. If you choose to see more, control jumps to line 2780 where another screen is presented and after that one, control comes back to line 620.

At line 620 we clear the screen and initialize the display for the first and only time for the rest of the program. All of the "fixed" data for the screen display is put up in the proper place. From here on, we will only be updating certain spots on the screen since it saves loads of time and makes a more presentable display. Also in this section of code, at line 770, we initialize both the year and quarter to one.

The Main Program Loop

The main program loop is between lines 830 and 1520. Variable J is the index for this loop. Notice that it isn't a For...Next loop, the reason for that being that we are going to jump in and out of it several times and the For...Next scheme simply won't tolerate that. Each time through this large loop will represent one calendar quarter during which we will be doing considerable housekeeping. Also inside this loop we will encounter a smaller loop which controls the reader board. Any time we

are not entering information or changing quarters we will be in the smaller loop so that the reader board will operate. The remainder of the program is made up of subroutines which we will go to at the appropriate times from the main loop or the smaller reader board loop.

Inside the main loop the first thing we do is check to see if we are done at line 830. When J reaches the value of N we go to line 2290 and end the simulation.

Next, at line 860, we read in the next quarter's data. Following this, at lines 890 and 900 we print the current year and quarter numbers on the screen.

Remember that earlier during initialization we read the very first quarter data in line 410? Well, now in the lines from 930 to 980 we can compare the current quarter data to what was read in line 410. We do it with a loop. C(I) represents the current quarter data, P(I) the previous quarter data. In line 940, B(I) will contain the difference between the previous quarter value and the current quarter value for all six items. Then in lines 950-970, CH$(I) will contain either "Unch", "Up" or "Down" depending on what the data actually did. We'll save this information in these arrays for a little later when they will be used.

We use line 1010 to key in on certain values of data to send control to a subroutine to print "questionable" advice to the user. This way, the advice doesn't just come up any old time, but only when certain levels are reached. We'll cover the subroutine when we get to it later.

In line 1040 we clear out the change information spots on the screen in preparation to putting new values there. STRING$(6,32) is six blank spaces, used to do the clearing.

Now that we have cleared the spaces, lines 1070-1090 put the new information on the screen. We put both the change information there and the new values. The rows on the screen where these pieces of information will go are rows five through 10, so we use X as the loop index (it's also the row index for our locate/print@ subroutine), set Y (the column index) inside the loop and go to the subroutine to tell where on the screen to print. But that presents a small problem: Our array containing the change information, CH$(I) and the amount of change B(I) are counted from 1 to 5. We fix that easily by subtracting four from each subscript in line 1080. (Keep in mind that we can create a subscripted array using an I index and read it out later with any other index we like - X in this case.) Note also that we are using the absolute value (ABS) of B(X-4). This is because CH$(X-4) already tells us that the value is unchanged, up or down. B(X-4) only needs to tell us *how much*, and a minus sign (for example) would be out of place here.

Now we need to "condition" our new quarter data (just read in) in preparation for the reader board display. We do this in lines 1130-1160. Here, we make a string out of an integer, remove the sign and the space reserved for the sign (see Puzzler answer, this issue.) Then in line 1150 we left-set this new string number in a field that is six characters long. All current quarter data are now in D$(CS), and each piece of data is in string form and is left-set in a six-character space. Yes, that's important for proper reader board operation.

In line 1190 we pay interest on the amount of money you have in the money market, if any. We do that by taking the current money market interest rate C(4) and multiplying it by the amount you have in money market and then adding it to what you have there already; but we need to divide by four since this is interest for only one quarter.

In lines 1220 and 1230 we update the user's stock and bond values. Variable S1 represents the number of stock shares owned and S2 the number of bond shares. We find the current dollar value owned by multiplying the current stock and bond price per share by the number of shares. If you don't own any shares, multiplying by zero still gives zero. In line 1230 T(9) represents your total dollar value and is made up by adding the money market value T(6), the stock value T(7) and the bond value T(8).

Now that we have all these new values figured out, lines 1260 through 1280 print them at the right spot on the screen. Note again that we are using X as the loop index, Y is fixed inside the loop and X goes from row six to row nine on the screen.

We are now ready to start "building" A$ in preparation for our reader board. A$ will be the string that circulates around the board. In line 1310 we establish A$. Note that it is a concatenated string containing our labels and the string values for the current quarter we set up so carefully earlier in D$(I). If you are typing this program in from the issue, be sure and get this string spaced exactly as shown in the listing. If you are one character off in one direction or the other, you will have pieces of A$ all over the screen. When it's right, it will flow rather smoothly from right to left and appear to go 'round and 'round. In line 1320 we initialize I at one. The reader board loop starts next, at line 1330, where (as usual) we check for a way out first. If I is equal to or greater than 63 we jump to line 1400 where we clear A$

to a null string and then go to line 1310 and read A$ all over again. In case you are wondering, A$ is 63 characters long, so doing what we just did makes the reader board look smooth. If we didn't null A$ and read it again, it would grow to 256 characters and give a "string too long" error. We have another "out" to look for in this section of code. At line 1340 we have an INKEY$ function. If it equals a null we keep right on going inside the loop; the THEN in line 1340 takes us to the very next line of code if IN$ is a null string. This keeps the reader board going as long as we are not doing anything with the keyboard. If we press any key, however, we jump out of this loop to line 1550 (which we will cover later) and the reader board has to stop momentarily while BASIC is in another part of the program. In lines 1350 through 1380 we read A$ one character at a time and let that one character be C$. Then we add C$ to the existing A$. In line 1370 we let B$ equal the right end of A$ (which is all the while building because C$ is being added to it) and then we print B$. The net effect is that B$ starts building up at screen column 63 and moves from there, one character at a time, left to screen column 1 and then repeats while the tail end of the previous iteration is still on the screen. As we mentioned earlier, this code was developed empirically, during a couple of long nights, and we *think* that's how it works.

We now have all the information for this quarter on the screen and in A$, so in lines 1430 to 1450 we take the current quarter values and make them the previous quarter values (getting ready for the next quarter already.) Next, in line 1480 we update the year and quarter numbers. If the quarter is incremented to 5 we increment the year by one and make the quarter equal to 1.

Line 1510 checks for the year number, and if it is equal or greater than 11 we end the simulation. Otherwise, line 1520 increments J by one and we go into another quarter. Checking for 40 quarters in line 830 and again checking for year 11 in line 1510 seems like overkill in retrospect - doesn't it? It looks as though we really want out of this loop, and in fact, we are done with it.

When we jump out of the reader board loop in line 1340 (because we pressed any key) we go to line 1550 which is a GOSUB. That doesn't sound like good form, does it? The problem is that we can't GOSUB directly from the reader board loop because we don't always want to come back to it (for example, when we continue to the next quarter.) So this seemed like the most reasonable way out. Line 1550 is a GOSUB to the "single upper case character" subroutine. It takes the first character entered on the keyboard and if it is not upper case, changes it to upper case. Now the program only has to look for upper case and we don't have to worry about how we enter characters.

When we return from the subroutine we immediately check to see if our character was one of the three permissible characters, B, S or C. If it wasn't, we go back to the reader board loop at line 1340 and keep it moving along while we wait for another (legal) character. It turns out that B and S in this program stand for BUY, SELL, BOND and STOCK, so it may be a little confusing at times. You have to watch *where* in the program they are being used.

If the character we input was C, then it means we want to continue and update to the next quarter. Line 1610 does it. First, we go to the subroutine at 2070 to update the user display, then print the normal prompt on the screen and go to 1430 to finish the current quarter housekeeping and update to the next quarter.

If the character we input at line 1340 was a B then we get to line 1660 where we want to BUY. Here we go to a subroutine at 2230 to clear out the command lines in preparation to asking some new questions there. Then we ask how many dollars worth to buy. Note, in line 1670, that we want to input a number but we make the input a string? Here's the reason for that: It's a very discriminating device. If we want a number and input a string value the VAL of it will be zero, so it lets us detect that and go back and ask the question again. If we input a number here the VAL will be that number and we go on about our way. Now that we know how many dollars worth, we ask if you want to buy stocks or bonds in line 1700. Then we go off to our "single character upper case converter" subroutine again and come back. At this point our answer can only be a B or an S. If it isn't we go back to 1700 and ask the question again. Assuming it is a B or an S, in line 1740 we subtract the dollar amount from the money market amount you own. Then we take three percent of it and add it to the broker's amount (that's what the broker gets every time you buy or sell.) Then we subtract the broker's amount from the amount you started with (that's how much you have left to buy with.) In line 1770, if you decided to buy stock, then S1 (the number of stock shares you own) is increased by the dollar amount you are spending divided by the current stock share price, otherwise, you must be buying bonds, so we do the same thing to your bond shares (S2). Then we go and update the user display to show your new values and then we go to the subroutine at 2230 to clear the

command lines on the screen and when we come back we print the usual prompt there again.

Selling is similar to buying, except that we display the number of shares you own and ask how many of them you want to sell. Anytime you sell, the money goes back into your money market account, and again, the broker gets his three percent. Notice that when you buy or sell, the program flow always gets you back to the reader board loop, but when you continue the quarter loop (J) is executed again. This allows you to do all sorts of business before continuing to the next quarter.

The end of simulation routine starts at 2290 and ends at 2350. It figures what you earned per year for the ten year period and displays it on the screen with appropriate remarks and then ends the program. If you are typing this program in from the magazine, do not include the exclamation marks after the large numbers in lines 2300 to 2320. Let your computer put them there by itself if it wants them.

The last section we need to cover is the advice to the user, from 2380 to 2750. There are five different responses available for five different conditions. The five responses in each section are picked at random by the code in line 2390. This whole section is designed so that the advice comes up on the screen (or doesn't) depending on whether or not you are invested in a certain market and also depending on whether that particular market is above or below a certain price level. The responses were collected from remarks overheard in barber shops, bars, doctor's office, etc., and are supposed to be typical of the kind of advice you would get from people who have nothing to lose by giving it. At first we thought it detracted from the overall feeling of the simulation, but after a few walk-in friends came to our office and tried it and were enthusiastic about it, we left it in.

Simulations of real life are fascinating and fun and especially interesting to program. Call them games if you want - but how do you know that life itself isn't a big game? ■

```
100 REM * Broker.Bas * For CodeWorks Magazine * Aug,1987
110 REM * 3838 S. Warner St. Tacoma, WA 98409 (206) 475-2219
120 REM * (C) 1987 80-NW Publishing Inc.
130 REM * Available to subscribers on download (206) 475-2356
140 '
150 ' do some initialization
160 'CLEAR 5000 ' Use if you need to clear string space.
170 T(6)=10000 ' to start with 10 grand
180 X$="$$###,###":X1$="###.##"
190 DEFINT I,J,U
200 '
210 ' make randomizer if needed
220  RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
230 '
240 'General purpose locate/print@ suboutine. Unremark as needed.
250 GOTO 330
260 LOCATE X,Y:RETURN ' for GW BASIC
270 'PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy I/III for example
280 'PRINT@((X-1),(Y-1)),;:RETURN ' Tandy IV for example
290 'PRINT@(X,Y),;:RETURN 'Some MBASIC machines
300 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN ' CP/M adjust
    to suit
310 '
320 ' establish the number of quarters in N
330 FOR I=1 TO 250
340   READ DU
350   IF DU=-1 THEN 370
360 NEXT I
370 N=INT(I/6)
380 RESTORE
390 '
```

```
400 REM * read 1st set of data for comparison
410 READ P(1),P(2),P(3),P(4),P(5),P(6)
420 CLS
430 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
440 PRINT"          I N V E S T M E N T  S I M U L A T I O N
450 PRINT"     a vicarious study in outrageous fortune or failure
460 PRINT STRING$(60,45)
470 PRINT
480 PRINT"This program simulates an investment plan in which
490 PRINT"you can invest in the Money Market, the Stock or
500 PRINT"the Bond market. Initially, you have $10,000 in the
510 PRINT"Money Market, which is a cash fund and all transactions
520 PRINT"must be made through it. During any trading session you
530 PRINT"can buy and sell as you wish. Continue will always advance
540 PRINT"you one calendar quarter. The simulation covers a 10 year
550 PRINT"period, during which you are to maximize the growth of your
560 PRINT"initial investment. Watch the indicators for clues on how
570 PRINT"and when to buy and sell. Your broker always takes 3%.
580 INPUT"  Do you want more detailed instructions (y/n)";IN$
590 IF IN$="Y" OR IN$="y" THEN 2780
600 '
610 REM * initialize the big board screen
620 CLS:PRINT"Year       Quarter                          THE   B I G   B
    O A R D"
630 X=5:Y=1:GOSUB 260:PRINT"DJIA --------- ";
640 X=6:Y=1:GOSUB 260:PRINT"CP Index ----- ";
650 X=7:Y=1:GOSUB 260:PRINT"Prime Rate % - ";
660 X=8:Y=1:GOSUB 260:PRINT"Money Mkt % -- ";
670 X=9:Y=1:GOSUB 260:PRINT"Stocks ------- ";
680 X=10:Y=1:GOSUB 260:PRINT"Bonds -------- ";
690 X=14:Y=1:GOSUB 260:PRINT"(B)uy, (S)ell or (C)ontinue";
700 X=5:Y=30:GOSUB 260:PRINT"Your current position:";
710 X=6:Y=30:GOSUB 260:PRINT"Money Market value --- ";
720 X=7:Y=30:GOSUB 260:PRINT"Stock value ---------- ";
730 X=8:Y=30:GOSUB 260:PRINT"Bond value ----------- ";
740 X=9:Y=30:GOSUB 260:PRINT"    Total value =====> ";
750 X=10:Y=30:GOSUB 260:PRINT"Total Brokerage fees - ";
760 X=6:Y=54:GOSUB 260:PRINT USING X$;T(6);
770 Y1=1:Q=1
780 X=2:Y=1:GOSUB 260:PRINT STRING$(62,"-");
790 X=4:Y=1:GOSUB 260:PRINT STRING$(62,"-");
800 '
810 REM * start the main program loop here (J loop)
820 J=1
830 IF J=>N THEN 2290
840 '
850 ' read in the next quarter data
860     READ C(1),C(2),C(3),C(4),C(5),C(6)
870 '
880 ' update year and quarter on the screen
890   X=1:Y=6:GOSUB 260:PRINT Y1
900   X=1:Y=20:GOSUB 260:PRINT Q
910 '
920 ' find up/down/unch since last quarter and how much
```

```
930   FOR I=1 TO 6
940     B(I)=C(I)-P(I)
950     IF C(I)=P(I) THEN CH$(I)=" Unch "
960     IF C(I)>P(I) THEN CH$(I)="   Up "
970     IF C(I)<P(I) THEN CH$(I)=" Down "
980   NEXT I
990 '
1000  REM * trigger some questionable advice for the user
1010  IF C(6)>60 OR C(6)<20 OR C(5)<30 OR C(5)>60 OR C(3)>13 THEN
      GOSUB 2380
1020 '
1030 ' clear out up/down/unch space on screen
1040  FOR X=5 TO 10:Y=23:GOSUB 260:PRINT STRING$(6,32);:NEXT X
1050 '
1060 ' write new up/down/unch data to screen
1070  FOR X=5 TO 10
1080    Y=15:GOSUB 260:PRINT CH$(X-4);:PRINT USING X1$;ABS(B(X-4));
1090  NEXT X
1100 '
1110 ' make strings of quarter data, remove left space, and
1120 ' make all fields 6 characters long and left-set the data.
1130  FOR CS=1 TO 6
1140    D$(CS)=RIGHT$(STR$(C(CS)),LEN(STR$(C(CS)))-1)
1150    D$(CS)=D$(CS)+STRING$(6-LEN(D$(CS)),"  ")
1160  NEXT CS
1170 '
1180 ' pay interest on money market amount each quarter
1190  T(6)=T(6)+(T(6)*((C(4)/100)/4))
1200 '
1210 ' update stock and bond values
1220  T(7)=S1*C(5):T(8)=S2*C(6)
1230  T(9)=T(6)+T(7)+T(8)
1240 '
1250 ' print updated values on the screen
1260  FOR X=6 TO 9
1270    Y=54:GOSUB 260:PRINT USING X$;T(X);
1280  NEXT X
1290 '
1300 ' establish A$ to hold titles and variable quarter data in D$( )
1310  A$="DJIA="+D$(1)+"CPI="+D$(2)+"PRIME="+D$(3)+"MM="+D$(4)+
      "STK="+D$(5)+"BND="+D$(6)
1320  I=1
1330  IF I=>63 THEN 1400 ' <-------- start of reader board loop
1340    IN$=INKEY$:IF IN$="" THEN ELSE 1550
1350    C$=MID$(A$,I,1)
1360    A$=A$+C$
1370    B$=RIGHT$(A$,LEN(A$)-I)
1380    X=3:Y=1:GOSUB 260:PRINT B$
1390  I=I+1:GOTO 1330   '  <-------- end of reader board loop
1400  A$="":GOTO 1310
1410 '
1420 ' Make Previous values P() equal to Current values C()
1430  FOR U=1 TO 6
1440    P(U)=C(U)
1450  NEXT U
```

```
1460 '
1470 ' keep the years and quarters straight
1480   Q=Q+1:IF Q=>5 THEN Y1=Y1+1:Q=1
1490 '
1500 ' provide for end of simulation
1510   IF Y1=>11 THEN 2290
1520 J=J+1:GOTO 830
1530 REM * End of main program loop (J loop)
1540 '
1550 GOSUB 2180 ' to change to one upper case character
1560 '
1570 ' make sure it's a character we want (S,B or C)
1580 IF IN$<>"B" AND IN$<>"C" AND IN$<>"S" THEN IN$="":GOTO 1340
1590 '
1600 REM * if user decides to continue (C)
1610 IF IN$="C" THEN GOSUB 2070:X=14:Y=1:GOSUB 260:PRINT"(B)uy,
     (S)ell or (C)ontinue":GOTO 1430
1620 '
1630 IF IN$="S" THEN 1840
1640 '
1650 REM * if user decides to buy (B)
1660 GOSUB 2230
1670 X=14:Y=1:GOSUB 260:PRINT"How many $ worth ";:INPUT HM$
1680 IF VAL(HM$)=0 THEN 1660 ELSE HM=VAL(HM$)
1690 IF HM>T(6) THEN HM=T(6)
1700 X=15:Y=1:GOSUB 260:PRINT"Buy Stocks or Bonds"
1710 IN$=INKEY$:IF IN$="" THEN 1710
1720 GOSUB 2180
1730 IF IN$<>"S" AND IN$<>"B" THEN IN$="":GOTO 1700
1740 T(6)=T(6)-HM
1750 T(10)=T(10)+.03*HM
1760 HM=HM-(.03*HM)
1770 IF IN$="S" THEN S1=S1+HM/C(5) ELSE S2=S2+HM/C(6)
1780 GOSUB 2070
1790 GOSUB 2230
1800 X=14:Y=1:GOSUB 260:PRINT"(B)uy, (S)ell or (C)ontinue
1810 GOTO 1340
1820 '
1830 REM * if user decides to sell (S)
1840 GOSUB 2230
1850 X=14:Y=1:GOSUB 260:PRINT"Sell Stocks or Bonds"
1860 IN$=INKEY$:IF IN$="" THEN 1860
1870 GOSUB 2180
1880 IF IN$<>"S" AND IN$<>"B" THEN IN$="":GOTO 1850
1890 IF IN$="B" THEN 1960
1900 PRINT"You have ";INT(S1)+1;" shares - ";:INPUT"sell how many";
     HM$
1910 IF VAL(HM$)=0 THEN 1840 ELSE HM=VAL(HM$)
1920 IF HM>S1 THEN HM=S1
1930 HD=HM*C(5)
1940 BF=.03*HD:T(10)=T(10)+BF:T(6)=T(6)+(HD-BF):S1=S1-HM:T(7)=S1*C(5)
1950 GOTO 2010
1960 PRINT"You have ";INT(S2)+1;" shares - ";:INPUT"sell how many";
     HM$
1970 IF VAL(HM$)=0 THEN 1840 ELSE HM=VAL(HM$)
1980 IF HM>S2 THEN HM=S2
```

```
1990 HD=HM*C(6)
2000 BF=.03*HD:T(10)=T(10)+BF:T(6)=T(6)+(HD-BF):S2=S2-HM:T(8)=S2*C(6)
2010 GOSUB 2070
2020 GOSUB 2230
2030 X=14:Y=1:GOSUB 260:PRINT"(B)uy, (S)ell or (C)ontinue
2040 GOTO 1340
2050 '
2060 REM * Update the user display
2070 T(7)=S1*C(5):IF T(7)<0 THEN T(7)=0 ' current stock value
2080 T(8)=S2*C(6):IF T(8)<0 THEN T(8)=0 ' current bond value
2090 T(9)=T(6)+T(7)+T(8)                ' current total value
2100 FOR X=6 TO 10
2110   Y=54:GOSUB 260:PRINT STRING$(10,32);
2120   Y=54:GOSUB 260:PRINT USING X$;T(X);
2130 NEXT X
2140 GOSUB 2230
2150 RETURN
2160 '
2170 REM * Upper case converter subroutine
2180 IN$=LEFT$(IN$,1)
2190 IF IN$=>"a" AND IN$<="z" THEN IN$=CHR$(ASC(IN$)-32)
2200 RETURN
2210 '
2220 REM * clear command lines subroutine
2230 FOR X=11 TO 15
2240   Y=1:GOSUB 260:PRINT STRING$(60,32);
2250 NEXT X
2260 RETURN
2270 '
2280 REM * end of simulation routine
2290 X=14:Y=1:GOSUB 260:PRINT"Ten years are up. Your average per year
     was";:PRINT USING X$;(T(9)-10000)/10
2300 IF T(9)>500000! THEN PRINT"Hey Boesky, the S.E.C. is here to see
     you!":END
2310 IF T(9)>250000! THEN PRINT"A true American Capitalist - good
     going!":END
2320 IF T(9)>75000! THEN PRINT"Not bad. Some room for improvement
     though.":END
2330 IF T(9)>25000 THEN PRINT"Not much profit in 10 years, is it?":
     END
2340 IF T(9)>10000 THEN PRINT"You're not even beating inflation.":END
2350 PRINT"Consider burying your money in the backyard.":END
2360 '
2370 REM * advice to the user
2380 X=12:Y=1:GOSUB 260
2390 PR=INT(RND(1)*5)+1
2400 IF C(6)>60 THEN 2450
2410 IF C(6)<20 THEN 2510
2420 IF C(5)<30 THEN 2570
2430 IF C(5)>60 THEN 2630
2440 IF C(3)>13 THEN 2690
2450 IF T(8)<100 THEN RETURN: ELSE ON PR GOTO 2460,2470,2480,2490,
     2500
2460 PRINT"Your broker says bonds can't go higher - SELL!":RETURN
2470 PRINT"Your barber says bonds are good for awhile - hold.":RETURN
2480 PRINT"Your mother-in-law says to SELL your bonds now!":RETURN
```

```
2490 PRINT"Money magazine says to sell your bonds.":RETURN
2500 PRINT"Smith-Barney says EARN your money in the Money Market.":
     RETURN
2510 IF T(8)>0 THEN RETURN: ELSE ON PR GOTO 2520,2530,2540,2550,2560
2520 PRINT"Your mother-in-law says don't buy bonds yet.":RETURN
2530 PRINT"Your neighbor is buying bonds.":RETURN
2540 PRINT"Your barber recommends buying bonds now.":RETURN
2550 PRINT"You are wise to stay out of the bond market.":RETURN
2560 PRINT"Your Doctor is investing in the bond market.":RETURN
2570 IF T(7)>0 THEN RETURN: ELSE ON PR GOTO 2580,2590,2600,2610,2620
2580 PRINT"Rukeyeser's Elves issued a BUY STOCK signal today.":RETURN
2590 PRINT"Wall Street Week says STOCK can't go anywhere but up.":
     RETURN
2600 PRINT"Your broker says wait before you buy Stock.":RETURN
2610 PRINT"Stay out of stocks, they are too volatile.":RETURN
2620 PRINT"Market analysts say stocks are going up - buy now.":RETURN
2630 IF T(7)<100 THEN RETURN: ELSE ON PR GOTO 2640,2650,2660,2670,
     2680
2640 PRINT"Your broker says the STOCK market will fall sharply.":
     RETURN
2650 PRINT"Your boss is selling all of his STOCK.":RETURN
2660 PRINT"Your wife says hold on to your stocks.":RETURN
2670 PRINT"Sell your stock - take your profit now!":RETURN
2680 PRINT"Your lawyer is dumping all his stock.":RETURN
2690 IF T(6)>100 THEN RETURN: ELSE ON PR GOTO 2700,2720,2730,2740,
     2750
2700 PRINT"Your banker says you should be in the Money Market.":
     RETURN
2710 PRINT"Get into the Money Market now.":RETURN
2720 PRINT"Your broker says to stay where you are for now.":RETURN
2730 PRINT"Your wife wants you to get into the Money Market.":RETURN
2740 PRINT"Psssst... You really should be in the Money Market.":
     RETURN
2750 PRINT"Consider the Money Market with such a high prime rate.":
     RETURN
2760 '
2770 REM * more detailed instructions
2780 CLS
2790 PRINT"The display will show a moving reader board which will
2800 PRINT"contain current values for the DJIA (Dow Jones
2810 PRINT"Industrial Average), the CPI (Consumer Price Index),
2820 PRINT"the PRIME (Prime lending rate), Money Market Rate (MM),
2830 PRINT"current stock price per share (STK) and current bond
2840 PRINT"price per share (BND). Also shown is the change
2850 PRINT"information for these. In addition, your holdings in
2860 PRINT"the market are shown, as well as a total brokerage
2870 PRINT"fee. The broker takes 3% on any buy/sell transaction.
2880 PRINT"Use the Money Market to park your cash.
2890 PRINT"  Use the Enter key with dollar amounts, other responses
2900 PRINT"will be activated when you press the first letter. It
2910 PRINT"is possible to parlay your $10,000 into over $500,000
2920 PRINT"in 10 years if you read the indicators correctly.
2930 LINE INPUT"Press Enter to start ";Q$
2940 GOTO 620
2950 '
```

```
2960 REM * Data for 10 years follows:
2970 DATA 847,196,7.5,5,90,40,831,197,8,5.5,88,37
2980 DATA 757,197,8.25,5.5,82,34,818,198,9,6,79,30
2990 DATA 865,205,9.5,6.5,75,27,805,210,11.75,9,72,25
3000 DATA 862,210,11.75,9,70,23,841,212,11.5,9,66,20
3010 DATA 878,222,13.5,11,60,18,838,239,15.25,13.5,40,14
3020 DATA 785,245,19.5,17,20,8,867,256,12,9,18,20
3030 DATA 932,273,13,10,22,30,963,281,21.5,19,18,34
3040 DATA 1003,285,17.5,15,24,43,976,291,20,18,20,62
3050 DATA 849,298,20,18,28,60,875,314,15.75,13.5,34,52
3060 DATA 822,320,16.5,14,38,58,811,327,16,13.5,44,64
3070 DATA 896,333,13.5,11,54,72,1046,337,11.5,9.5,60,80
3080 DATA 1130,338,11,9.5,62,84,1221,339,10.5,9,60,82
3090 DATA 1233,340,10.75,9,58,80,1258,344,11,9.5,52,71
3100 DATA 1164,346,11,9.5,50,67,1132,348,11.5,9.5,42,63
3110 DATA 1206,358,13,10.5,48,57,1211,361,12.75,10,49,58
3120 DATA 1266,368,10.75,8,50,60,1335,370,10,7.5,48,61
3130 DATA 1328,370,9.5,7,56,62,1546,382,9.5,7,62,48
3140 DATA 1818,390,9.5,7,68,37,1892,401,9,6,78,32
3150 DATA 1910,412,8,5.25,84,28,2012,432,7.5,5,82,18
3160 DATA 2034,440,7.5,5,93,14,2230,451,8,5.5,98,16
3170 DATA 2067,442,7.5,5.5,102,13,-1
3180 END 'of program
```

Change lines for Tandy I/III

```
Changed->100 REM * Broker/Bas * For CodeWorks Magazine * Aug,1987
Changed->160  CLEAR 5000 ' Use if you need to clear string space.
Changed->220 'RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->260 'LOCATE X,Y:RETURN ' for GW BASIC
Changed->270 PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy I/III for exampl
e
Changed->2300 IF T(9)>500000 THEN PRINT"Hey Boesky, the S.E.C. is her
e to see you!":END
Changed->2310 IF T(9)>250000 THEN PRINT"A true American Capitalist -
good going!":END
Changed->2320 IF T(9)>75000 THEN PRINT"Not bad. Some room for improve
ment though.":END
Changed->2390 PR=RND(5)
```

Change lines for Tandy II/IV

```
Changed->100 REM * Broker/Bas * For CodeWorks Magazine * Aug,1987
Changed->160  CLEAR 5000 ' Use if you need to clear string space.
Changed->220 'RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->260 'LOCATE X,Y:RETURN ' for GW BASIC
Changed->280 PRINT@((X-1),(Y-1)),;:RETURN ' Tandy IV for example
Changed->2390 PR=RND(5)
```

Beginning Basic

A Lotto Statistics Program

In this issue we have taken code and ideas from previous installments of Beginning BASIC, added a couple of new ideas, and come up with a Lotto statistics program. We hope it will help you make intelligent choices.

In this issue we'll take many of the topics covered in previous installments, add a couple of new ones, and make a program that does statistics on winning lotto numbers. And even though we thought we were out of graphs in the last issue, the program includes one because sometimes that's the best way to view a collection of data. But first, a few comments about the idea behind the program.

Does what happened in the past indicate what will happen in the future? It might, and then it might not. For example, if certain numbers were picked infrequently, would you pick them because they were "about due," or would you stick with the numbers that keep coming up because you think they will continue to come up? Unfortunately for lotto players, if you flip a coin 100 times and it happened to come up heads 100 times, the chance of it coming up heads on the 101st flip is still 50/50. Not much hope, is it?

In spite of all that, it still seems that you can do better than letting the lotto machine pick your numbers for you. After all, would you pick a series of numbers starting in the 20's, and how often do numbers starting like that win? The machine recently picked a couple of numbers for us that had 21 as the low number - fat chance, although it could happen to be a winner.

With that in mind, our program will look at three characteristics of the past winning numbers. It looks at the six numbers by position and gives the average number for the position. It then gives the standard deviation for that position. Standard deviation simply says that roughly 68 percent of all the numbers fall into the amount of standard deviation, plus or minus. If the average is 24 and the standard deviation is 5, then 68 percent of all the numbers were between 19 and 29. The third thing the program does is to graph the number of times any digit came up. The program is set up to look at the last 50 drawings. Some of you who only have 16 screen lines may want to cut that down to the last 25 drawings so that the data will fit on the screen. The data is held in DATA statements at the end of the program, where you can add new data to the end of the list and remove the oldest data from the start of the list (then don't forget to save the program!). This keeps us away from file maintenance and also covers the few of you who still don't have disks.

Once you are familiar with the program, of course, you can easily add your own ideas of what you would like to see. The basic structure and the means to manipulate the data are all there. There are a couple of neat routines in the program that you may find useful in other areas, and we will cover them as we get to them.

The Program

We start with some initialization in lines 160 to 180. Variable H is the number of digits your particular state picks from. Variable S is the number of drawings you carry in the DATA statements at the end of the program. S=50 is a good number for screens that display 24 lines, and also makes a nice number for the averages. Two arrays are dimensioned next. The T(x,x) array is a two dimensional array and will hold the data (50 deep and six wide). The C(x) array is single dimension and will be used to hold the number of times each digit came up.

The cursor positioning subroutine comes next. If you are using GW-BASIC, use it as is; if not, then remark line 220 and un-remark the appropriate line for your machine.

Lines 280 through 390 clear the screen and print the heading and description of the program on the screen. The variable XX in line 390 is a "dummy" variable, used to hold the screen in its current position until you press RETURN or ENTER.

The little nested loop in lines 420 to 460 reads the data statements into the T(x,x) array. Notice that to read into a two-dimensional array you always need two loops - one inside the other. The inside loop (the J loop) will always read from one to six before the outside loop (the I loop) clicks over once, so the T(I,J) array will be filled and look like the data in the DATA statements. Beginning BASIC in issue 8, Nov/Dec 1986 was on arrays of two dimensions if you care to refer back to it.

The code from lines 490 to 650 presents a new idea for Beginning BASIC. This little routine is going to find averages and standard deviation. First, let's get a clear picture of the T(I,J) array. In it, we are counting across (horizontally, or rows) with the J loop and down (vertically, or columns) with the I loop. Again, looking at the data statements in the program helps visualize this. The very first DATA element would be T(1,1) and the last would be T(50,6).

But here, we are going to make the outside loop the J loop and the inner loop will be the I loop. This means that we are going to count up all the digits in the first position, from the first to the 50th, then do it again for the second position (column), etc. The inside I loop from 530 to 550 is going to give us a total of the values in each column of the array and put that total into variable TL, but we are only at the first column right now. When I has finished going from one to S (50) and has totaled the numbers in the first column, the loop ends and line 560 will set variable M equal to that total divided by the number of items in the column. That gives us the average for the column in variable M.

We are still on the first column (the J loop is still at one) and we need to know the average for the next section of code from lines 570 to 600. In these lines, we will find the standard deviation of the values we have just totalled. Just as TL was an accumulator in the previous code, variable V will accumulate values in this section. Variable V is going to accumulate (sum) the square of the difference between each value in column one and the average of all the items in column one. Every value in column one will either be equal to the average of the entire column, above the average or below it. Variable V will sum the square of those differences - the sum of the squares.

When this loop is finished, line 600 will tell us how many units away from the average 68 percent of the values fall. This is the standard deviation (from the average), and is found by taking the square root of the sum of the squares (V) divided by the number of items in column one (S).

Now since we have what we want, both average and standard deviation, we can print them on the screen. Line 610 does that, and because these values may be strung out to umpteen places, we round down to the next whole integer. Variable M is the average and variable SD is the standard deviation. Both these variables were accumulating variables, and now that we are done with them for the first column we have to clear them both out to get ready for the second column, otherwise, second column values would be added to the first column values. The J loop now increments to two and we do the second column, and so on, to column six.

Where does the 68 percent come from? Our statistics book tells us that for normal distributions it turns out that 68.27 of the cases are included between the average less the standard deviation number and the average plus the standard deviation number, i.e., one standard deviation on either side of the average. For further reading on this subject, see Dstat.Bas, in issue 6, Jul/Aug 1986.

Next we need to find out how many times a winning digit appeared in the last 50 drawings. Again, we cruise through the two-dimensional array. This time it doesn't make any difference if we do it by rows or columns, as long as we look at each digit. The code from 680 to 720 looks deceptively simple, but it's interesting. Our C(x) array (single dimensioned array) was initialized at H, or in our case 48, because that is the number of digits the winners are picked from. Right now, that array contains all zeros because when you run a BASIC program all arrays are initially set to zero until something is specifically assigned to them.

Let's digress for just a bit. C(N) denotes the Nth position in the C array. Both the C and the N are *locating* devices, *not the value contained at that location*. So C(9) would be the 9th position in array C, and that position can contain some data value. It follows then, that a position could be designated as C(I+2), or C(J-K).

Now look at line 700. In it, we are defining the location in the C array by T(I,J) or C(T(I,J)), so the *value* contained in T(I,J) becomes the *location* in the C array where we are going to add one to what is already there. In short, as we go through the numbers in the two-dimensional array, every time T(I,J) is equal to 23, for example, then C(23) will get one added to whatever is already in C(23). Obviously, when we read the C array out later, from C(1) to C(48), each of those array positions will hold the number of times that the number of that position came up.

We now get to the graph part of the program.

First, in lines 760 and 770, we create a couple of print strings that contain the numbers that will later go at the bottom of the graph and go from 1 to 48.

Lines 800 to 830 print numbers up the left side of our graph, with zero being at the bottom and the numbers increasing as we go up the screen. Yes, it actually prints up the screen. In line 800 we set Y to 2. Y is the horizontal screen position, so this puts us at column two and leaves us there. The loop then starts at 20 and goes to 5 with a minus step, and prints the value of 20 less X - so it prints the digits zero to 15 up the left side of the graph.

In the next section of code, from lines 860 to 900, we print sort of a grid on the graph space. It's a double loop that controls both X and Y screen positions. Note that the Y position steps 10. We are using the underscore character as a tick mark. Note that the GOSUB in both these loops is to position the cursor at the X and Y locations given.

Lines 930 through 990 put the actual data into the graph. Remember that C(x) contained the number of times any digit came up. Again, we use a double loop, the outside I loop going from one to H or 48 in our case, and the inner X loop going from 20 (the bottom of the graph) up to the value contained in that particular C(x) array position. We started this section of code with Y equal to 6, because that is where the graph starts, but after we have printed the vertical bar representing the value for the digit 1, we need to move over one to the next position. Line 980 does it by incrementing Y by one. The CHR$(179) in the program is a very nice vertical bar that is centered in the character space and goes from the top of the character space to the bottom. This makes it look continuous when they are printed on successive lines. The change lines for other machines at the end of the listing will have other CHR$ characters for this, and may not look as neat. The one shown in the main listing is for GW-BASIC and MS DOS.

Actually, it makes a pretty nice looking graph, and you can tell at a glance which numbers are coming up and which ones aren't. It won't change much from drawing to drawing, but over a period of time it will. The only thing left to do is to put our numbers at the bottom of the graph, and lines 1020 and 1030 do that for us.

If you have any neat ideas on things to add to this program, you can put a "Press enter to continue" line at 1040 and start your code there. If the DATA statements get into your way, you can renumber them selectively with the NEWLINE, STARTLINE, INCREMENT scheme. For example to move them to 5001, you would say, RENUM 5001,2001,1. We purposely started the DATA lines at 2001 instead of 2000 so that you can tell exactly how many there are by looking at the last digits of the line number.

The DATA lines shown in the program are fake. We generated them with the random number generator, using the Lpick program from the last issue. There are several lotto publications around that give the numbers for your state. Also some newspapers carry them from time to time. Failing that, you might write to your local state lottery commission and ask them for the winning numbers - who knows, they might give them to you. You can use the program with less than 50 numbers by simply changing the 50 in line 170 to whatever number of DATA lines you have. Which is why we made a variable out of that number; so you wouldn't have to change it in many places in the program. While you are at it, don't forget to set H in line 160 to the number of digits your state picks from.

Given all that, will it work? We have an unsubstantiated feeling that you might do better with this than letting the machine pick your numbers. Also, we tend to think that numbers that don't come up that often will continue to fail to come up - mostly because there might be something about how they blow those ping-pong balls around. What if some of them are just a little bit heavier? We're sure they check for things like that, but who knows?

If you should luck out and hit it big using this program - please don't forget who your friends are! If you don't win, you have the consolation that at least you learned something about BASIC programming, and if knowledge is power, who needs handouts? ■

```
100 REM * Lstat.Bas * does stats on winning lotto numbers
110 REM * CodeWorks Magazine 3838 S. Warner St. Tacoma WA 98409
120 REM * (206) 475-2219 voice (206) 475-2356 300/1200 modem
130 REM * (c)1988 80-NW Publishing Inc. & placed in public domain.
140 '
150 'Initialization
160 H=48 ' pick six from H numbers.
170 S=50 ' number of weeks you have data for.
180 DIM T(50,6),C(H)
```

```
190 '
200 'Generalized Locate/Print@ subroutine. Unremark as needed.
210 GOTO 280
220 LOCATE X,Y:RETURN ' MS-DOS, GW-BASIC
230 'PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy Models I/III
240 'PRINT@((X-1),(Y-1)),;:RETURN ' Tandy Models II/IV
250 'PRINT@(X,Y),;:RETURN ' Some MBASIC machines.
260 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN ' CP/M
270 '
280 CLS
290 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
300 PRINT"        L O T T O   S T A T I S T I C S   P R O G R A M
310 PRINT"          shows statistics on past winning numbers
320 PRINT STRING$(60,45)
330 PRINT
340 PRINT"This program analyzes the past fifty winning lotto numbers.
350 PRINT"It gives you the average for each of the six positions, the
360 PRINT"standard deviation of each, and prints a graph showing how
370 PRINT"many times each number was picked as a winner.
380 PRINT
390 INPUT"Press enter to continue";XX
400 '
410 'read the data
420 FOR I=1 TO S
430   FOR J= 1 TO 6
440     READ T(I,J)
450   NEXT J
460 NEXT I
470 '
480 'compute average and standard deviation
490 CLS
500 PRINT"Position","Average","Standard Deviation"
510 PRINT
520 FOR J=1 TO 6
530   FOR I=1 TO S
540     TL=TL+T(I,J)
550   NEXT I
560   M=TL/S
570   FOR K=1 TO S
580     V=V+(T(K,J)-M)^2
590   NEXT K
600   SD=SQR(V/S)
610   PRINT J,INT(M-.5),INT(SD-.5)
620   TL=0:V=0
630 NEXT J
640 PRINT
650 INPUT"Press enter to see the graph";XX
660 '
670 'find frequency for each number
680 FOR I=1 TO S
690   FOR J=1 TO 6
700     C(T(I,J))=C(T(I,J))+1
710   NEXT J
720 NEXT I
```

```
730 '
740 CLS
750 PRINT TAB(10) "How many times each number was picked"
760 F1$="                1111111111222222222233333333334444444444
770 F2$="     12345678901234567890123456789012345678901234567890123456789012345678
780 '
790 'print the numbers to the left of the graph
800 Y=2
810 FOR X=20 TO 5 STEP -1
820   GOSUB 220:PRINT 20-X
830 NEXT X
840 '
850 ' print the tick marks every ten spaces across
860 FOR Y=5 TO 55 STEP 10
870   FOR X=20 TO 5 STEP -1
880     GOSUB 220:PRINT " "
890   NEXT X
900 NEXT Y
910 '
920 'now print the data on the graph
930 Y=6
940 FOR I=1 TO H
950   FOR X=20 TO 20-C(I)+1 STEP -1
960     GOSUB 220:PRINT CHR$(179)
970   NEXT X
980   Y=Y+1
990 NEXT I
1000 '
1010 ' put the numbers under graph
1020 X=21:Y=1:GOSUB 220:PRINT F1$
1030 X=22:Y=1:GOSUB 220:PRINT F2$
1040 END
1050 '
1060 'data lines start here - there should be 50 lines.
1070 'Each week, remove the first and add the new one at the end.
1080 '
2001 DATA 6,23,28,34,38,47
2002 DATA 11,17,18,33,37,44
2003 DATA 5,11,22,28,42,43
2004 DATA 3,9,13,23,40,46
2005 DATA 4,8,26,34,40,46
2006 DATA 8,14,16,25,27,42
2007 DATA 2,22,23,32,38,45
2008 DATA 2,23,25,26,28,45
2009 DATA 9,19,27,31,42,44
2010 DATA 2,13,17,31,39,46
2011 DATA 4,6,22,30,41,43
2012 DATA 3,6,9,31,37,38
2013 DATA 4,15,19,26,38,39
2014 DATA 17,20,21,25,34,42
2015 DATA 4,12,22,26,32,34
2016 DATA 14,26,28,32,36,37
2017 DATA 21,27,31,32,41,44
2018 DATA 6,26,29,34,38,45
```

```
2019 DATA 3,13,25,42,45,47
2020 DATA 4,5,6,24,27,30
2021 DATA 4,18,27,30,39,46
2022 DATA 15,23,29,35,37,39
2023 DATA 6,11,14,27,40,47
2024 DATA 4,5,40,44,22,39
2025 DATA 2,23,29,35,38,48
2026 DATA 5,8,9,12,22,28
2027 DATA 3,18,26,27,32,37
2028 DATA 6,9,25,33,35,47
2029 DATA 2,10,20,33,38,48
2030 DATA 9,16,18,36,46,47
2031 DATA 10,11,17,18,24,44
2032 DATA 1,5,18,38,42,48
2033 DATA 9,19,22,26,33,45
2034 DATA 1,16,28,29,42,48
2035 DATA 4,9,17,32,35,45
2036 DATA 3,10,28,29,46,48
2037 DATA 25,29,34,37,38,41
2038 DATA 23,25,32,33,46,48
2039 DATA 12,17,21,22,23,46
2040 DATA 3,4,15,22,27,39
2041 DATA 9,18,19,29,39,46
2042 DATA 5,25,32,39,40,45
2043 DATA 4,11,20,22,38,43
2044 DATA 9,25,43,44,46,47
2045 DATA 15,36,41,42,43,45
2046 DATA 14,16,27,32,35,47
2047 DATA 1,10,22,23,27,48
2048 DATA 1,4,12,22,26,45
2049 DATA 1,5,7,14,19,42
2050 DATA 4,13,32,33,38,46
```

"Do you have one that can keep a secret?"

Change lines for Tandy I/III

```
Changed->100 REM * Lstat/Bas * does stats on winning lotto numbers
Changed->140 CLEAR 2000
Changed->220 'LOCATE X,Y:RETURN ' MS-DOS, GW-BASIC
Changed->230 PRINT@((X-1)*64)+(Y-1),;:RETURN ' Tandy Models I/III
Changed->580      V=V+(T(K,J)-M)[2
Changed->810 FOR X=13 TO 5 STEP -1
Changed->820    GOSUB 220:PRINT 13-X;
Changed->870    FOR X=13 TO 5 STEP -1
Changed->880      GOSUB 220:PRINT " ";
Changed->950    FOR X=13 TO 13-C(I)+1 STEP -1
Changed->960      GOSUB 220:PRINT CHR$(149);
Changed->1020 X=14:Y=1:GOSUB 220:PRINT F1$
Changed->1030 X=15:Y=1:GOSUB 220:PRINT F2$
```

Change lines for Tandy II/IV

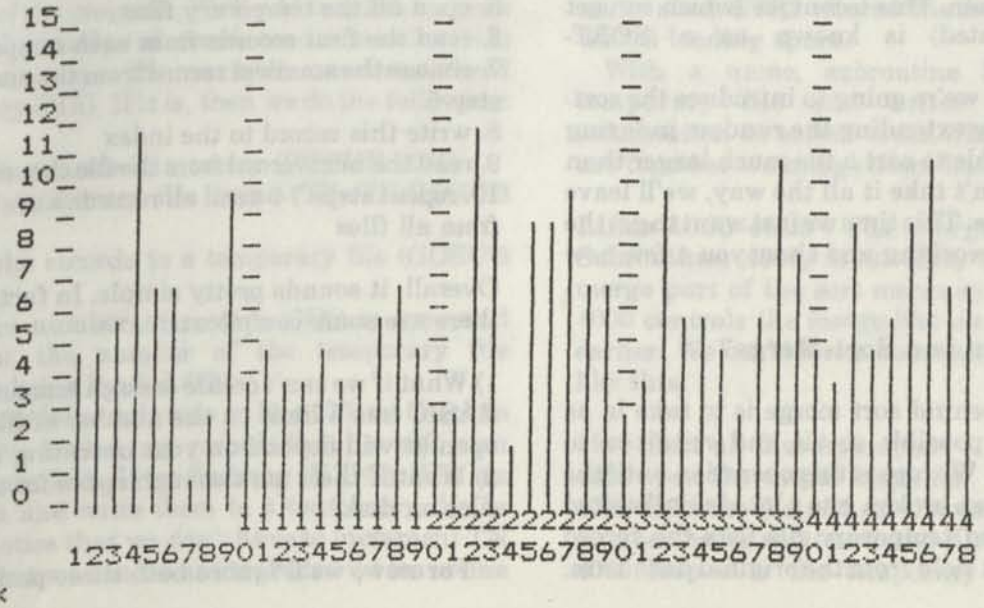
```
Changed->100 REM * Lstat/Bas * does stats on winning lotto numbers
Changed->220 'LOCATE X,Y:RETURN ' MS-DOS, GW-BASIC
Changed->240 PRINT@((X-1),(Y-1)),;:RETURN ' Tandy Models II/IV
Changed->820   GOSUB 220:PRINT 20-X;
Changed->880     GOSUB 220:PRINT "_";
Changed->960     GOSUB 220:PRINT CHR$(149);
```

```
 Position        Average        Standard Deviation

    1              6                 5
    2             14                 7
    3             22                 7
    4             29                 6
    5             35                 6
    6             43                 3

Press enter to see the graph?
```

Sample RUN of Lstat.Bas



Random Files

Improved Sorting

Terry R. Dettmann, Associate Editor. In this installment Terry shows how to sort files that are larger than can be contained in memory. It is a very useful and necessary addition to the Randemo series.

Random Indexing, Part 2

Several issues ago (Issue 13, Sep/Oct 87) when we introduced indexing for sorting files, we had a pretty bad limit on it, we could only sort as many records as we could fit into memory. The program Ranindex.bas set this limit to 400. With MS DOS machines, it is practical to set that limit up to 1000 or even higher in some cases. Even at that, we still will run out of memory. To get beyond the bounds of memory, we have to go a little further. That's the subject of this and the next article in this series.

Extending a sort beyond the available memory is a problem that programmers on large installations have had to fight for years. The first system I worked on (an old IBM 7040 system with five tape drives and no disk) had only 32K of memory. In order to sort items, we had to write temporary files out to tape and then merge them back together again. This technique (which can get pretty complicated) is known as a 'SORT-MERGE'.

In this article, we're going to introduce the sort-merge concept by extending the random indexing program to be able to sort a file much larger than memory. We won't take it all the way, we'll leave that for next time. This time we just want to get the basic functions working and show you a few new techniques.

What is a Sort-Merge?

The concept behind sort merge is to take in as much of a file as possible, sort it, and write it out to a temporary file. We repeat this operation until the whole file has been broken into a lot of small sorted files. Each sorted temporary file lists the record number and sort field from the original data base.

When the whole file has been covered, then we take all of the temporary, sorted files and read in the first record of each. Since ALL of the files are in sorted order, we know that the first record in each file is the smallest record in that file. If we simply take the smallest of all the first records, that *must* be the smallest record in the whole data base, so we write its record number to the index. Then we read the next record from the file we just chose the smallest record from and repeat the whole thing again.

The steps look like this:
1. load n records from the data base
2. sort the n records
3. write the records to a data file
4. if there are more records in the data base, then repeat steps 1-3 until the whole data base has been sorted
5. open all the temporary files
6. read the first records from each temporary file
7. choose the smallest record from the ones read in step 6
8. write this record to the index
9. read the next record from the file chosen in step 7
10. repeat steps 7-9 until all records have been read from all files

Overall, it sounds pretty simple. In fact, it is! But there are some complications:

1) What if we can't create enough temporary files? BASIC has a limit on the number of files we can open (it will depend on your operating system).
2) What if there isn't enough space for temporary files on disk?

For now, we'll ignore both these problems and

come back to them next issue. Each is a whole article all in itself.

To implement our procedure above, I've taken a copy of RANINDEX.BAS and converted it to the new program RANIDX.BAS. to get the proper starting point, I took the RANINDEX program from Issue 13 and applied the changes made in Issue 14 (page 24 ... allows selecting records for printing) and then I added the changes for the sort merge operation. Let's go through the changes to see what they cover.

Lines 10-60 - Program Initialization
We've added line 25 to set the maximum number of data records the program is allowed to sort in memory at one time (I've set it here for 500. Some people can set it higher. Experiment for yourself). In line 31, I've changed the arrays which hold the data to be sized by the MX variable created in line 25. This way, you only need to change line 25 to change how much you can hold in memory.

Line 51 has changed to create the variables TN=1 (The current temporary file number) and TX=10 (The maximum number of temporary files). For the present, if you have more than TX*MX = 5000 records in your data base, you won't be able to sort it until next issue. You could try increasing TX and MX to see how much more you can get sorted. I'd be interested in how high someone can go.

Lines 100-170 - Random File Initialization
Nothing changes here. The random file won't be changed by this program at all.

Lines 200-290 - Program Main Loop
The main loop of the program gets shorter in this version of the program. We'll add line 265 to check to see if the number of records read in to index (NX) is at least equal to the maximum number allowed in memory (MX). If it is, then we do the following:

1. Sort the records in memory (GOSUB 2000)
2. Get a temporary file name (TF=TN:GOSUB 3200)
3. Save the records to a temporary file (GOSUB 3300)
4. Set the number of records (NX) to zero and increment the number of the temporary file currently being loaded (TN)

When our loop ends now at line 270, we check to see if we've loaded any records that haven't been sorted (lines 280). If we find any (NX>0), then we sort them and write them to a temporary file as before. Notice that we *don't* have to increment TN here. In fact we don't want to. When we leave line 280, TN will be the number of temporary files that were written to the disk. We'll use that in merging them together.

When everything has been put into a temporary file, we can then merge it back together by calling the subroutine at 4000. When that's done, the program's done! Simple isn't it? (By the way, if you REALLY believe that, I have this bridge in Brooklyn I've been trying to sell. I'd make you a REAL good deal!).

Lines 300-510 - The end of the program
If you compare this version of the program to RANINDEX.BAS published originally, you'll notice that lines 300-450 have been deleted. Since we now have a print program that uses an index (RANPRINT.BAS) there isn't much need for a print program here. Lazy as I am, I decided not to rework the routine and trusted using RANPRINT for file printouts.

Lines 3000-3999 - The temporary file operations
Our next changes don't occur until we get into the 3000's. I've deleted lines 3000- 3100 (I didn't reuse them for clarity only. We'll reuse the lines later). The subroutine at 3200 assigns a temporary file name for use with the program.

For this program, I've chosen to assign all temporary files the name SRT plus a unique file number (taken from the TN variable originally) and finally a .TMP extension. In this way, I could have up to 99999 files with unique names (you probably don't have a disk that could hold that much data). The MID$ function called in the name building is to prevent a space from getting into the name since STR$ returns the number as a string with a leading space.

With a name, subroutine 3300 opens that temporary file and writes out the sorted information in sorted order.When all of the files are together we merge them back.

Lines 4000-4399 - The merge file operation
Subroutines 4000, 4100, 4200, and 4300 are the merge part of the sort merge system. Subroutine 4000 controls the merge like our basic procedure earlier. We could write subroutine 4000 in words like this:

4010 close all files now open
4020 open the output index file
4030 field the file and set the variable NR (record number being written to the file)
4040 loop over the temporary files (TN is the

number of temporary files created, TX is the maximum to work with at one time)
4050 open a group of temporary files
4060 get the smallest record from each of the open temporary files, if there are no more records to get, we're done
4070 write the smallest record to the index and repeat from line 4060
4080 repeat the loop from 4040
4090 close all files and return, we're done

Writing the lines out this way, their meaning is clear. This is the heart of the merge operation and the steps are the same as those we set up before.

Subroutine 4100 opens a group of temporary files. As each file is opened, it reads the first entry in the file and saves it in the arrays IX (index number) and DA$ (data). Since we already have these arrays built in the program, we save space by reusing them for this. Now, the index for the arrays refers to file number instead of a sort index.

Subroutine 4200 gets the lowest data record from those read in already. We look through the K files read into memory. The smallest (variable LW for LOW points to it) is chosen to write to the file. To indicate that we've reached the end of the file, we put three tilde characters in DA$(I) to indicate that file I has no more records to select. If the lowest value of DA$ is now three tildes, then ALL of the files are at an end. At the end of the loop, we check for the lowest DA$ being three tildes and if it is, we return FOUND = FALSE which signals line 4060 that we're done.

If DA$(LW) wasn't three tildes, then we get ready to return the values in IX(LW) (the record number for the index) and the data value (the sorting data field). Before we're done, we have to get the NEXT record from the data file into memory. If the file is at end of file, then the data item becomes three tildes, otherwise, it's the data from the next record.

I know you're wondering, "WHY THREE TILDES?" It's simple really. The tilde character (ASCII code 126) was the highest printable character available (ASCII code 127 is the DELETE character). By selecting three tildes, when we did our check for the next lowest record, *anything* that we found would be lower EXCEPT another set of three tildes (it's also unlikely to be REAL data in a data base). When the lowest available data IS three tildes, then NOTHING IS LEFT and we're done.

The subroutine at 4300 simply writes the record to the index file (remember, it writes ONLY the record number of the next desired record in sequence).

With these changes made, you can now sort files of up to 5000 records (10 files times 500 records) or larger. But the merge is incomplete. To get larger files still, we have to allow for more than one level of merging. That will be our next installment. After that, we'll look into how to do a sort of the data 'IN PLACE' without writing temporary files. It will be slower, but you could sort a file that completely filled a disk if you wanted to.

```
10 REM --- RANIDX.BAS - Random File Indexing - VERSION 2.0
20 REM --- Terry R. Dettmann for Codeworks Magazine
25 MX=500
30 DIM FP$(20), SC$(24), XY(20,3)
31 DIM DA$(MX), IX(MX), IR(MX)
40 DEF FNCTR$(X$)=STRING$((WD-LEN(X$))/2," ")+X$
41 DEF FNLF(X) = LOF(X)/128
50 WD=80:LN=24
51 NX=0:TN=1:TX=10
60 FALSE=0:TRUE = NOT FALSE
100 REM --- file setup
110 CLS:PRINT FNCTR$("RANDOM FILE INDEXING"):PRINT:PRINT
120 LINE INPUT"FILENAME: ";FF$
125 FD$=FF$+".dat":FS$=FF$+".stk"
130 OPEN "R",1,FD$:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
135 IF LOF(2)=0 THEN LSET SK$=MKI$(1):PUT 2,1
140 FM$=FF$+".MAP":FX$=FF$+".SCN"
150 GOSUB 5000: REM Read Map
170 GOSUB 5300: REM Setup Fielding
```

```
200 REM --- main menu
210 CLS:PRINT FNCTR$("RANDOM FILE INDEXING"):PRINT:PRINT
215 LINE INPUT"Name of the index: ";FI$:FI$=FI$+".idx"
220 INPUT"Sort on what field number";FX
230 IF FX<1 OR FX>CX THEN PRINT"OOPS - no such field":GOTO 220
231 INPUT"Select field number (enter 0 for none)";SX
232 IF SX=0 THEN 240
233 IF SX<1 OR SX>CX THEN PRINT"No such field number";GOTO 231
234 LINE INPUT"Select Criteria: ";SX$
240 FOR RN=1 TO FNLF(1):GOSUB 1400
250    IF FP$(1)="DELETED" THEN 270
255    IF SX>0 THEN IF INSTR(FP$(SX),SX$)=0 THEN 270
260    GOSUB 1000
265    IF NX>=MX THEN GOSUB 2000:TF=TN:GOSUB 3200:GOSUB 3300:NX=0:
       TN=TN+1
270 NEXT RN
280 IF NX>0 THEN GOSUB 2000:TF=TN:GOSUB 3200:GOSUB 3300
290 GOSUB 4000
500 REM --- End of Program
510 CLS:PRINT"All Done":CLOSE:END
550 REM --- Save the program
560 SAVE "ranidx.bas"
570 RETURN
600 REM --- input a character
610 C$=INKEY$:IF C$="" THEN 610
615 IF LEN(C$)>1 THEN GOSUB 700
620 RETURN
700 REM --- look for arrows
710 C = ASC(MID$(C$,2,1))
720 IF C=72 THEN C$=UP$ ELSE IF C=77 THEN C$=RT$
730 IF C=80 THEN C$=DN$ ELSE IF C=75 THEN C$=LF$
740 RETURN
800 REM --- GOTO XY ROUTINE
810 LOCATE X,Y:RETURN
900 REM --- break line
910 FOR K=1 TO 10:BL$(K)="":NEXT K
920 JN$=IN$:NB=1
930 K = INSTR(JN$,":"):IF K=0 THEN BL$(NB)=JN$:RETURN
940    BL$(NB) = MID$(JN$,1,K-1)
950    NB = NB + 1
960    JN$ = MID$(JN$,K+1)
970 GOTO 930
1000 REM --- Add the record to the index
1010 NX = NX + 1
1020 DA$(NX) = FP$(FX)
1030 IX(NX) = NX:IR(NX)=RN
1040 RETURN
1400 REM --- get record from data base
1410 IF RN<1 OR RN>FNLF(1) THEN RETURN
1420 GET 1,RN
1430 RETURN
2000 REM --- Sort the index
2010 DF = NX:PRINT "SORTING ..."
```

```
2020 IF DF = 1 THEN RETURN
2030    DF = INT(DF/2)
2040    SWP = FALSE
2050    FOR I=1 TO NX-DF
2060       IF DA$(IX(I))>DA$(IX(I+DF)) THEN GOSUB 2100:SWP = TRUE
2070    NEXT I
2080    IF SWP THEN 2040 ELSE 2020
2100 REM --- swap the data fields
2110 T = IX(I):IX(I) = IX(I+DF):IX(I+DF) = T
2120 RETURN
3200 REM --- Select Temporary File Name
3210 FT$="SRT"+MID$(STR$(TF),2)+".TMP"
3220 RETURN
3300 REM --- Save the Sorted data to a Temporary File
3310 PRINT "Saving Temporary File ";FT$
3320 OPEN "O",3,FT$
3330 FOR I=1 TO NX
3340    PRINT #3,IR(IX(I));",";DA$(IX(I))
3350 NEXT I
3360 CLOSE #3
3370 RETURN
4000 REM --- Merge Data from Temporary Files to Index
4010 CLOSE
4020 OPEN "R",1,FI$,2
4030 FIELD 1, 2 AS XX$:NR = 1
4040 FOR I=1 TO TN STEP TX
4050    GOSUB 4100
4060    GOSUB 4200:IF NOT FOUND THEN 4080
4070    GOSUB 4300:GOTO 4060
4080 NEXT I
4090 CLOSE:RETURN
4100 REM --- open temporary files
4110 IF I+TX > TN THEN JX=TN ELSE JX=I+TX
4115 K=0
4120 FOR J=I TO JX:K=K+1
4130    TF=J:GOSUB 3200
4140    OPEN "I",K+1,FT$:INPUT#K+1,IX(K),DA$(K)
4141    PRINT"FILE: ";K;" ENTRY=";IX(K);DA$(K)
4150 NEXT J
4160 RETURN
4200 REM --- get lowest entry
4210 LW=1:FOUND = TRUE
4220 FOR J=2 TO K
4230    IF DA$(J)<DA$(LW) THEN LW=J
4240 NEXT J
4245 IF DA$(LW)="~~~~" THEN FOUND=FALSE:RETURN
4250 IX = IDX(LW):IX$ = DA$(LW)
4255 IF EOF(LW+1) THEN DA$(LW)="~~~~":RETURN
4260 INPUT#LW+1,IDX(LW),DA$(LW)
4265 PRINT"FILE: ";LW;" ENTRY=";IX(LW);DA$(LW)
4270 RETURN
4300 REM --- save to index
4310 PRINT "ITEM (";NR;") = ";IX$
```

```
4320 LSET XX$ = MKI$(IX):PUT #1,NR:NR=NR+1
4330 RETURN
5000 REM --- read data map
5001 CX = 0
5005 OPEN"I",3,FM$
5010 IF EOF(3) THEN 5035
5015    LINE INPUT#3,IN$
5020    GOSUB 900
5025    GOSUB 5100
5030 GOTO 5010
5035 CLOSE#3
5040 RETURN
5100 REM --- decode map line
5110 IF BL$(1)="FIELD" THEN GOSUB 5200:RETURN
5120 RETURN
5200 REM --- define a field
5210 NF = VAL(BL$(2)):FL = VAL(BL$(4)):FP = VAL(BL$(5))
5220 XY(NF,0)=FL:XY(NF,3)=FP
5225 CX = CX + 1
5230 RETURN
5300 REM --- Map Fields
5310 FOR I=1 TO CX
5320    NL = XY(I,3)
5330    FIELD #1, NL-1 AS X$,XY(I,0) AS FP$(I)
5340 NEXT I
5350 RETURN
```

Computing Notes

If you have MS DOS and a color monitor here is a little program to allow you to set the color balance correctly. It puts out 15 bars of the different colors and puts the number at the end of each bar. As you adjust the brightness and contrast controls on the monitor, color 8 (slate gray) will disappear first. Adjust so that color 8 is, indeed, slate gray when compared to color 7 right above it (which is all white). When you have that, all the other colors will fall into place.

```
100 COLOR 7,0
110 CLS
120 FOR I=0 TO 15
130   COLOR I,0:PRINT STRING$(I*5
      ,220);:PRINT I
140 NEXT I
```

When you write a program with DATA statements in it, it is customary to put them at the end of the program. During the writing of the program, however, they constantly get in the way as you add to the end of the program. So put them at the start of the program during the writing and move them later. This, of course, is because DATA statements will be read by BASIC no matter where they occur in the program.

Our calculator said that the sine of 45 degrees was .707 but the computer gave a totally different number when we asked for SIN(45). Checking the manuals we found that most of the BASICs we cover require that when finding the sine of a number, the number must be expressed in radians. So asking the computer to print SIN(45/57.2958) gave the proper answer: .707106. The number 57.2958 is what you get by dividing 180 degrees by Pi, which is what one radian is in degrees.

The TAB function in BASIC does the same rounding as PRINT USING. If you TAB(20.4) it will tab to position 20, but if you TAB(20.5) it will

go to position 21. Taking the INTeger of the number before you tab to it in both cases would get you to position 20, because INTeger wipes out everything after the decimal.

In GW BASIC and BASIC after version 5.0 if you want to get rid of the question mark on an INPUT statement without using LINE INPUT, do this: INPUT";",X This leaves the cursor at the input position without a question mark. This example works only for integers, entering a letter here would get you a "redo from start" prompt. It's interesting to note that if you say: INPUT";",X$ you get the question mark again. If you say: INPUT"Press enter",X you don't get the question mark but if you enter a number it bumps right up against the prompt, so you can put a space between the word enter and the closing quote mark and it still works.

As you probably already know, our download system is a UNIX system. While browsing around in it the other day I noted that all the files have a month, day and year stamp on them. However, anything you do in a given day has only a time stamp. At midnight each day, the system automatically changes the time stamp to a date stamp. UNIX is full of goodies like that. While marveling over this little feature it occurred to me that if you have TIME$ in your computer you could append the hour and minute to your file name automatically before you save a program you are working on. Then, as you updated and modified your program you could save several versions of it, each with a slightly different file name and each telling you (by the name itself) how old that version was. That way, if something you are trying to do doesn't work out, you can go back and load a previous, unspoiled copy. You would probably need to keep the working name of the program short, so that you could append (from DATE$ or TIME$) the day, hour and minutes. An example would be: AB231523.BAS, which would be the 23rd of the month, at 3:23 in the afternoon. Naturally, after you have perfected your masterpiece, you would want to kill off all the files that led up to it, and change the finished version's name to something more intelligible.

How long does it take to thoroughly check out a program? We have been using Plist.Bas ever since Issue 5 and it has always performed flawlessly. You would think that it had, by this time, really been subjected to just about every possibility. But noooo - in the last issue (14), in Ecal.Bas it dropped the apostrophe (remark) in lines 1160 and 1180. These lines were "calibration" lines so that you could get the spacing of the line above or below just right. We are sure that most of you figured out that these were supposed to be remark lines, but we did get a couple of calls about syntax errors in those lines.

The other day we tried to load up a program on one of our work disks and it told us that the file was not found. OK, so we tried to do a directory on the disk and it told us again that the file was not found! Heaven's to Betzy! What do you do now? There were all sorts of goodies on that disk - new programs under development, all sorts of goodies that made life easier and a couple of biggies that were in the finishing stages. Well, we hauled out PC Tools and took a look. The file allocation table was mismash - couldn't read a thing. So we tried to fix it and sure enough, we could get a directory. But when we loaded one of the files we found that there was a totally different bunch of code half-way through the program. We also loaded program A and got program B and parts of C instead! All in all, there were only two programs out of about 57 that we could recover.

Fortunately, we found some recent backup copies of most of this disk on our hard disk; and so we did recover over 90 percent of what we lost.

We tell this little tale for one reason only: ALWAYS keep current backup copies of important data! That's number one. Number two is to not keep using the same diskette year after year. Copy it over to a fresh diskette every few months if it is something you use daily. We found that the disk with the problem had been created in March 1986 - two years ago. It was worn rather thin, and it was rather dumb of us to ignore such a simple thing. It's also a good idea to write-protect your backup diskettes. That way, you won't inadvertently write all over them. Take the write-protect off only when you need to update the diskette and put it back on when you're done.

ORDER FORM

Issues

- ☐ Renew Subscription $24.95
- ☐ NEW Subscription (starts with Nov/Dec 1987 issue)..... $24.95
- ☐ All 1st Year CodeWorks issues (Issue 1 through Issue 7) $24.95
- ☐ All 2nd year CodeWorks issues (Issue 8 through Issue 13) ... $24.95

☐ **GIFT Subscription**

Please give both your name and the name and address of the person who will receive the gift.

Clip or photocopy and mail to **CodeWorks,**
3838 S. Warner St.
Tacoma, WA 98409

Computer type:

Comments:

Diskettes

- ☐ 1st year programs on disk (specify type below) $20.00
- ☐ 2nd year programs on disk (specify type below) $20.00

- ☐ PC/MS-DOS 40 track DSDD
- ☐ CP/M 5 1/4 inch (specify format and computer type here _____

_____

- ☐ TRSDOS Model I 35 track SSSD
- ☐ TRSDOS Model III 40 track SSDD
- ☐ TRSDOS Model IV 40 track SSDD

388

HOW?

- ☐ Check/MO enclosed
- ☐ Bill me later
- ☐ Charge to VISA/MC_____ *Exp* _____

TO: ***(Please print clearly)***

Name _____

Address _____

City _____ *State*_____ *ZIP*_____

Charge card orders may be called in (206) 475-2219 9 am to 4 pm weekdays, Pacific time.

Index Update

Misc, setting graphics in MS DOS machines, issue 15, page 4
Misc, more on ASCII files and identification, issue 15, page 4
Misc, STR$ and VAL explained, issue 15, page 6
Misc, program, Rset.bas, issue 15, page 8, on right set values
Misc, program, Lc.bas, issue 15, page 8, a page line counter
Misc, program, Lpick.bas, issue 15, page 8, select lotto numbers
Randist.bas, main program, issue 15, page 9, shows random distribution
Ledger.bas, main program, issue 15, page 13, a check ledger program
Notes, processing speed and delay loops, issue 15, page 26
Notes, the CHR$ commands below number 31, issue 15, page 26
Random files, improvements to Randemo5.bas, issue 15, page 27
Repl.bas, main program, issue 15, page 34, a search and replace utility

If you are using Qkey.Bas to keep a running index of CodeWorks articles and notes, these are the changes to bring that index up to date through the last issue.

Download

What's happening on the download

January and July seem to be the favorite months of the year for our download to crash. Sure enough, it happened again in January this year. For reasons unknown to us, the 68000 processor in the download system crashed and burned. The repair of it took a little longer than it usually does, and so we were down for over a week around the middle of the month. Fortunately, when we got the computer back up and running, we found that the integrity of the files was still there. That in itself saved a couple of days in restoring the file system. The files are on a 15 megabyte, outboard hard disk, and apparently the crash didn't have any effect on it.

We apologize for the inconvenience to any of you who tried to get on the system during that time and found it not answering. At least, by not answering, it didn't cost you for the phone call. The system appears to be functioning properly now.

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
US Postage
PAID
Permit No. 774
Tacoma, WA

CodeWorks

Issue 17 **May/June 1988**

CONTENTS

CodeWorks

Issue 17 May/Jun 88

Editor/Publisher
Irv Schmidt
Associate Editor
Terry R. Dettmann
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Advisor
Al Mashburn

Subscription price: $24.95 per year (six issues.) A subscription year runs from Nov Dec through Sep Oct. Anyone subscribing during the current subscription year receives all issues for that year. Not available outside the United States Zip codes. **VISA and Master Card orders are accepted by mail or phone (206) 475-2219. Charge card orders may also be left via our on-line download system (206) 475-2356.**

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in the United States of America. Bulk rate postage is paid at Tacoma, Washington.

Sample Copies: If you have a friend who would like to see a copy of CodeWorks, just send the name and address and we will send a sample copy at no cost.

Editor's Notes

We have been hearing from more and more of you who are moving into MS DOS, both from CP/M and the earlier Tandy models. And looking around at what is available these days and at what price, it's no wonder. There are some rather neat bargains out there. In spite of that, and according to our best method of accounting for such things, about 40 percent of you are non-MS DOS.

All of which incited a spirited debate in our office. With so many people turning to MS DOS, we reasoned, we should do a column on helping them get started in this (to them) new operating system. After all, it is extremely versatile and has features beyond most other operating systems. But what about the 40 percent who don't have it yet and don't want it? Why rob space from them to cover an operating system they couldn't care less about?

When we plugged this question into our overall situation we finally came up with the answer. Why not publish a one-time "How to get started with MS DOS" booklet and sell it to those of you who need it. That would get the information directly to those who need it and at the same time help out our ever-present financial resource situation.

With that in mind, I thought back to when I first encountered the system, and then started writing an outline. In all fairness though, I'll have to bounce it off of someone else, because prior to my getting into MS DOS, I had considerable experience with UNIX and CP/M from which MS DOS borrowed quite a few conventions. The booklet will probably be slanted heavily towards the TRSDOS user who is moving into the new system. Aside from that, it should sell for less than $10, and get you started in MS DOS with a minimum of fuss and bother. We'll keep you posted on how it progresses.

About three issues ago I mentioned that I was exploring the possibilities of desktop publishing. Well, since then I have rather thoroughly checked out both Xerox Ventura and the Aldus PageMaker programs. Although both have their fine points, I finally settled on PageMaker as the one I would use. Since then, I have created a few full-page advertisements and a couple of newsletters, and it works pretty slick. This is one area where computers can really make a contribution. For old-time typesetters and paste-up people there is an initial hump to get over. It's a bit unnerving to have the whole page in front of you and be able to simply add more text where it's needed and then rearrange the whole page with a few keystrokes. And the output on a laser printer doesn't look that bad. If the cost of a laser printer scares you, there are places that will run off the pages you bring to them on diskette for about a dollar per page (which is what I did.)

And speaking of laser printers, there are a ton of them around, but not too many that will truly give you what a typesetting machine will. For that, you need a postscript compatible laser with bunches of memory and lots of downloadable fonts. Needless to say, those printers that will do that cost several thousand bucks. The other thing I found was that most laser printers will zip out a page of text in seconds, but if you have graphics on the page it could take hours, even assuming that the printer has enough memory to hold an entire page of graphics.

Like just about everything else these days, it's what they don't tell you that gets you. Like the so-called "simplified tax forms" we just went through. But that's another story I'd rather not even get started on. And I'll bet you don't either.

Irv

Forum

An Open Forum for Questions & Comments

Issue 16 came just in time! I was beginning to have withdrawal symptoms, or something similar. For quality control purposes, my copy started coming apart already due to insufficient gluing.

Suggestion: A genealogy database. Given the name of any person, it would look up (1) ancestors, (2) descendants, and (3) siblings. It should also print out genealogy charts and family data sheets on an 80-column printer.

Sam Laswell
South Haven, MI

Hmmmmm... see the next letter and the answer to it.

...I purchased a TRS-80 Model III with the idea of taking up computer programming as a hobby during my retirement years, and I found CodeWorks to be just what I needed to get started in that direction. But I obtained a genealogy program and became engrossed in research of the history and genealogical record of my family. I still enjoy learning how to work in BASIC, but have little time for it as my genealogical research began to require up to six hours daily. The software I was using proved inadequate to accommodate the growing database, so I purchased COMSOFT ROOTS II, a much more sophisticated genealogy program. The problem was that it runs on MS DOS and requires a minimum of 384K of RAM, so my Model III could not handle it, and I switched to a Tandy 1000 SX with 640K of RAM and two disk drives.

I still would like to become proficient in BASIC programming, and I have seen nothing that compares with CodeWorks from that point of view, but for the foreseeable future my genealogical research gets priority. I am printing out family group sheets, pedigree charts, etc., for relatives scattered across the country, plus publishing a newsletter to keep everyone posted on my progress in compiling data for a family history... I echo the feelings expressed by so many of your readers in the past: Keep up the good work!

Russell Bond
Buffalo, NY

A good genealogy program is really a system of programs. It is no small undertaking, by any means. There is a very good program of this type called: Genealogy on Display (Melvin O. Duke, PO Box 20836, San Jose, CA 95160 - Shareware - $35.00 donation requested - contains more than 25 BASIC programs all connected via a master menu.) The one problem I see with this program is that it only has the capacity to hold the names of 500 persons. This, in spite of the fact that a person is only in the database once and links are provided to other names that apply - which is the way it should be. I recently modified Card.Bas for one genealogical enthusiast with which he kept a list of immigrants (ship passenger lists). We found out rather quickly that it was woefully inadequate; simply not enough space in memory unless it was coded down to obscure, one-letter codes. Our random file series would be better suited to the task. If we can persuade Terry to establish a linking system in it, we could use the program "as is" for file maintenance and then create a special report generating program that uses the links. No absolute promises yet, but it is something I think deserves attention.

...everyone is selling hard disk drives at good prices. That is not a problem. When I call and ask if their disk drive will work with my Tandy III or IV no one seems to know. Can you tell me? Can I use an AT or XT compatible with my TRS-80, can I buy one type of drive and match it with a different controller and get it to work - or what? You can see why I need help. How about a good article about what to do or what can be done to get into hard disking?

Bill Sharpe
Atlanta, GA

Get hold of AEROCOMP, 2544 West Commerce Street, Dallas, TX 75212 (214) 638-8886. They have drives and operating system patches for the Models III and IV. And yes, this is one we can aim at our buddy and resident technical expert, Al Mashburn.

...your magazine has helped me very much with the chore I have to change my TRSDOS BASIC programs over to MS DOS BASIC...I am having trouble (with a TRSDOS BASIC program) that uses CHR$(27) to keep the cursor at the same position when you use default information. Now, is there a routine that will accomplish this in MS DOS, if so point me in that direction. PEEK and POKE are another problem...

Ray Jasek
Spooner, WI

The CHR$(27) problem you mention may be fixed by changing it to a semicolon instead. We have never seen that convention used in TRSDOS before, however, the semicolon will keep the cursor at the position it was when you enter something. As for the PEEKs and POKEs and other miscellaneous differences between MS DOS and TRSDOS, see Robert Hood's fine article on that subject in this issue.

When I run Ledger.Bas I get errors in line 300. I think the problem is somewhere between lines 300 and 350. I used CP/M line 340 alone and it did not help. I sure could use this program - can you help?

Samuel Levens
Evergreen Park, IL

(We assumed that Dr. Levens was using CP/M and a Heath 89) For our cursor locating line to work you should remark line 300 and un-remark line 340. Also see our article on direct cursor positioning in Beginning BASIC, this issue. We also went through our correspondence and found one Heath 89 owner who said that our CP/M line worked for him when he changed the "31" in that line to "33" in both places. You might try that.

Last year I got this swell Kaypro II with a lot of software on a closeout. That is when the fun started. First, my wife laughed at me as I am 55 years old. Then I found I could not type. Well, after a lot of time and fun, I got a sample copy of CodeWorks and thought I would learn BASIC. The magazine is great, but I have a small problem with some of the programs I don't know how to fix... I don't know what to do with LOCATE X,Y. The Kaypro uses Microsoft BASIC-80 Rev. 5.2.

Dale Sweet
Grand Island, NE

The line you un-remark for your Kaypro is the one for CP/M. However, it still won't work as it is because our line is for the Heath H19 terminal, which happens to be a very popular one. A little research brought up this for the Kaypro II:

PRINT CHR$(27) CHR$(61) CHR$(31+X) CHR$(31+Y);:RETURN

Notice that there isn't the usual comma between the CHR$ groups, only a space. The semi-colon at the end is there to keep the cursor at that location on the screen. There are well over 200 different terminal types for CP/M, so you can see why we can't print all of them in our LOCATE/PRINT@ subroutine. And, if your wife laughs at you at 55, what is she going to do when you turn 60?

First off, I want to commend you on the excellent programs that you have published thus far. Keep up the good work. I recall on at least two occasions, your reply, which was most prompt, helped me make the necessary corrections to my problems. Having nothing better to do during the cold New England months, I took the opportunity to go through all back issues, page by page. You would be surprised how many suggested corrections to various programs I found. I recall one incident where you published my letter regarding PEEKs and POKEs and Mr. Walker's suggestion (issue 7) of obtaining a copy of BASIC Better and Faster might be what I was looking for. By using the eight patches suggested in that book, it allowed me to enter almost any program that uses either PEEK or POKE.

Regarding some of the programs you have published to date, I have used SEARCH on many programs. Your program REPL.Bas, in my estimation, goes one step further. By utilizing this program, I find that by searching and replacing LPRINT to PRINT, I have, for example, Wood/Prt and Wood/Vid, which gives me the opportunity to print or view on the screen...

Howard M Cruff, Jr.
Attleboro, MA

Thank you for the kind words. (Readers should note that the book referred to is for TRS-80 models, and was a "must read" back in those days.)

I enjoyed reading your ram (pump) piece because I have admired them for a good sixty years. The ram is an elegant machine that uses the momentum of falling water to overcome the inertia of standing water and move the standing water to a higher level. When it does so, some of the rising slug of water is discharged at the higher level.

The air compressed by the falling water is only a shock absorbor; its expansion is small compared to the volume of the water discharged in one pulse. If you could build the plumbing strong enough, or controlled the closing of the dump valve (to prevent excessive pressure), you wouldn't need the

air chamber at all. The air chamber absorbs the "water hammer." ...I appreciate your efforts. By all means please continue.

George Mueden
New York, NY

Your letter characterizes the replies received concerning the pump program. I am still batting zero in pulling off a decent April Fool joke. You were all supposed to tell me this article and program were a hoax - and then I would have come back and said it was real (which it is). But you all bought it for what it really was, so I suppose the joke is on me after all. Next year I'll describe the antigravity machine I'm working on.

Re: John Omohundro's letter on the ArcSIN function in Issue 16. I thought that the program was a very good example of the technique for searching for an answer, especially the sieve concept which narrowed down the search. It is very similar to some of the techniques I discussed in my book, Artificial Intelligence Programming Techniques In BASIC (Wordware Publishing). I do have two comments on the program, however.

First, almost all computers do have an ArcTAN function. It is ATN(X). You can derive the ArcSIN and ArcCOS from the ArcTAN with the following equations:

ArcSIN(X)=ATN(X/SQR(1- X*X))
ArcCOS(X)=1.570796-ATN(X/SQR(1-X*X))

All angles are in radians. To convert to degrees, multiply by 180/3.1416.

Second, sending the computer to the next line with an IF...THEN statement seems rather silly since the computer would normally go there anyway. I am referring to lines in the program such as:

140 IF X>A THEN 150 ELSE 160

This also makes the program unusable on computers that do not have the ELSE statement. I suggest reversing the condition expression. For example, line 140 becomes:

140 IF X<=A THEN 160.

David Leithauser
New Smyrna Beach, FL

Thanks for the good suggestions. (Mr. Leithauser also submitted a program called Mediator.Bas, which should appear in the next issue of CodeWorks.)

Regarding the letter of Susan Hope Dunham which appeared in issue 16 reference her inability to use TRSDOS 6.2 with a date past 12/31/87, please advise Ms. Dunham as follows:

TRSDOS 6.2 as well as previous versions, stores the last two digits of DATE$ at memory location 51. If you POKE 88 into that memory location you will have the proper year. To have this done automatically each time you boot up, enter and save the following BASIC program called DATE88/BAS:

```
0 CLS:POKE 51,88:SYSTEM "DATE":
SYSTEM
```

Then set the AUTO function from TRSDOS Ready by keying in:

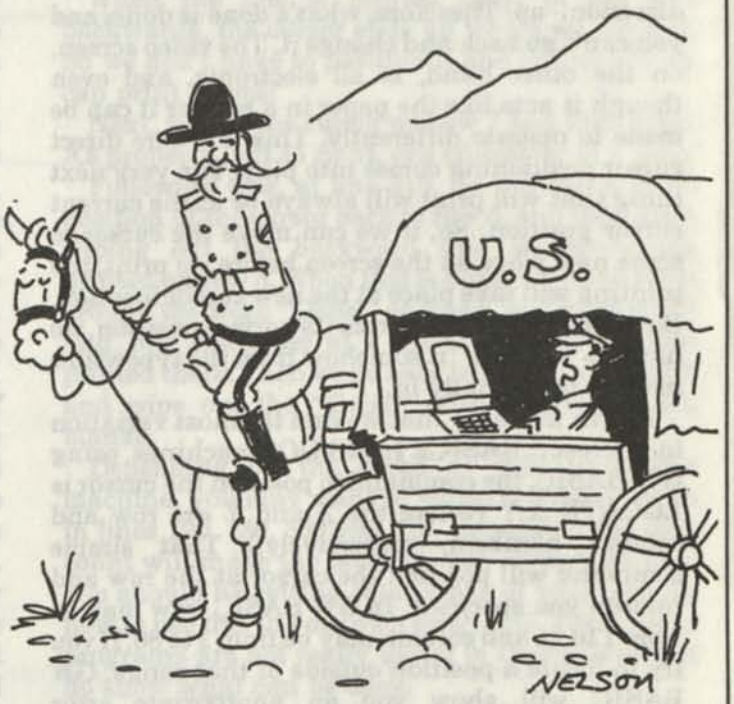
AUTO BASIC DATE88/BAS (ENTER)

Each new boot up after that will automatically go into BASIC and set the year to 1988.

Stanley J. Goldstein
Westwood, NJ

Thanks, Stanley, we tried this and it worked just fine on our Model IV.

That's it for this time. Enjoy the late springtime, and we'll see you again in early July. - Irv



"General, the computer says there never were that many Indians under one chief before."

Beginning Basic

Direct Cursor Positioning

In this installment we will look at direct cursor positioning and present three different sample programs to illustrate how it works. Before we start, however, let's find out exactly what we mean when we talk about direct cursor positions.

Most of the time when you use your computer it is in what may be called "typewriter mode." Which means that the cursor starts at the upper left of the screen, fills the line it is on, drops down one line and fills that one, and so on - very much like a typewriter would. When the screen fills up, the new line simply is added to the bottom of the screen and the line at the very top is pushed off of the screen. This is commonly called a "scrolling display." So far, the screen is acting very much like a typewriter or teletype machine.

In most printers the paper only moves in one direction - up. Therefore, what's done is done, and you can't go back and change it. The video screen, on the other hand, is all electronic, and even though it acts like the paper in a printer it can be made to operate differently. This is where direct cursor positioning comes into play. The very next thing that will print will always be at the current cursor position. So, if we can move the cursor to some new place on the screen before we print, the printing will take place at the new cursor location. But to move the cursor from its normal position, we have to "unhook" it somehow from the typewriter mode it is normally in.

It is in this area that we find the most variation in Microsoft BASICs. In MS-DOS machines, using GW BASIC, the command to position the cursor is LOCATE X,Y (where the X and Y are row and column numbers, respectively.) That simple command will position the cursor at the row and column you specified. In GW BASIC, row may be from 1 to 24 and column may be from 1 to 80. If you try to locate a position outside of that range, GW BASIC will show you an appropriate error message.

Other machines, notably the Tandy models prior to their MS DOS models, use a PRINT@ command to position the cursor. The numbering scheme on these machines is quite different than that of MS DOS. Here, they don't use rows and columns, but start counting at zero at the upper left of the screen and then count every position, just like you would read a book, right down to the lower right of the screen. It's not that difficult to get used to, but it is harder to visualize PRINT@ 832 than it is to think of row 11, column 32. On these machines, if you try to position the cursor outside the legal range, you will get an Illegal Function Call error.

Other machines, especially those running MBASIC under CP/M, use an escape sequence to unhook the cursor. There is a lot of variation in this particular sequence from one MBASIC machine to another. One of the most common, however, is the one for the H-19 terminal. Escape (ESC) is CHR$(27), and so the sequence goes like this: PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y). In this case, X and Y are again row and column numbers, respectively. When you supply the X and Y numbers and print the sequence just given, the cursor will position at the X and Y location.

So how do we make CodeWorks programs work for all these different cursor locating schemes? We have created a "Universal PRINT@/LOCATE" subroutine. You will find it in all of our programs that require cursor positioning. Take a look at the listing for DMAKER.Bas (in this issue), lines 190 through 260. Note that only *one* of the lines from 210 to 250 should be un-remarked. This way, no matter which computer you are using, if you un-remark the proper line and remark all the others, the program will automatically be right for you and should work.

Since this subroutine is used so much, it is placed near the beginning of the program (so it won't take so long for BASIC to find it), and the GOTO in line 200 simply jumps program flow around the subroutine when you RUN the program. Note too, that each positioning line is terminated with a RETURN, since each line is a one-line subroutine.

With this suboutine in the program, all we need to do now when we want to position the cursor (for any machine) is to set X equal to the row number, Y equal to the column number, and GOSUB 210. Then we print whatever it is we want printed at that X and Y location.

With direct cursor positioning we can go anywhere on the screen, print something there

without disrupting what is already on the screen and without scrolling, and then go somewhere else on the screen and print something else there. We can just as well erase something that is already on the screen by going there and printing a suitable number of spaces. What you can do with this that a printer cannot do is to go back up the screen and print something. In fact, you can jump around, anywhere on the screen, and print or erase at will. Our three demo programs with this installment show various ways in which cursor positioning works. The most interesting one of the three is Cursor3.Bas because it not only shows how direct cursor positioning works, but in addition, gives a very graphic demonstration of how a bubble sort works. You can see it happen right there on the screen. Try them all though, they are short and easy to type in, so you will get the feel for what it's all about.

Cursor1.Bas

Cursor1.Bas is a simple demo that prints a list of names at the upper left of the screen. It then goes through the list, and if a name starts with the letter J, it clears out the space and prints the name to the right of where it was. It does this without disturbing any of the other names.

The names from the DATA line are read in with the loop at line 160. The same loop also prints the names on the screen. Line 200 is a short delay loop, so that you can see what the list looks like before anything happens to it. In line 220 we set Y (the column number for the positioning subroutine) to 1. It will stay there for the rest of the program. Notice that we can use the row number (X) as a loop counter and at the same time as the row number. And why not? We are going to go down the list of names (one row at a time) to see if the name starts with the letter J. If it does, then we are at Y=1 and X=(the name that starts with J) and we go to the subroutine to position the cursor and then print a string of blanks and then the name that was there. The string of blanks, of course, is determined by the number of spaces between the quotes in line 240. In line 250 we again put in a little delay so that you can keep up with what is going on. If we ended and didn't do anything about the cursor first, it would probably wipe out the last name or two. So in line 270, we position the cursor out of the way of the names and then issue a print nothing. That way, the names remain intact and the cursor ends up a couple of lines below them.

Cursor2.Bas

In the first program we left the Y position fixed and varied the X position. In this one we will fix X and vary Y for some interesting variations. Line 150 says that we are going to start printing on line 5. But Y is the loop counter this time, and it's going to start printing at the length of A$ (from line 140). What's more, it is going to put the first character of A$ way out there at the end of the line and print the line backwards! A$ is 45 characters long. X is at row 5, and Y starts at column 45 and prints the first character of A$ there (it's a T). Then, coming back to the left, it will print the remainder of the line. In line 170 both X and Y have been defined already, so all we have to do is go to the positioning subroutine to get the cursor there. Then line 180 prints the character at that Y location on the screen. To get it to print backwards, we had to introduce the variable C and increment it each time through the loop. Note that in line 180 we are printing the character defined by MID$ of A$ at position C for one character. The delay in line 190 is there to slow things down so you can see them happen. For some variation, change the C in line 180 to Y and see what happens.

When the line is printed right to left and backwards, the cursor ends up where we want it. So we don't have to position it and in line 220 we can print "Oops!"

Next, we put the cursor on row 7 and then print A$ properly, left to right, one character at a time.

Following this, at line 310, we put the X (row) position of the cursor back to row 5, and then start wiping out the backward line we previously printed on row 5. When this is done, we jump right around the word "Oops!" on row 6, and go to row 7 and, one character at a time, wipe out the line printed there. Then we go back to row 6 (in line 440) and wipe out the "Oops!" with a string of six blanks.

Depending on the speed of your particular machine, you may want to adjust the delay loops in lines 190, 280, 350 and 420. Increasing the loop count will make the program run slower. By now you should be getting some ideas of your own on how to use this in your own programs. These demo programs are not really useful in themselves, but do show what can be done.

Cursor3.Bas

This should be the most interesting of the three programs. In it, we go through a standard bubble sort on ten names and sort them into alphabetical order. But, each time through the sorting loop that a swap has to be made, we jump out of the loop to a

subroutine where we will make the swap of names shown on the screen as well. When a name with a larger ASCII value comes ahead of one with a smaller ASCII value, that name will jump out to the right. Then the name below will move up to where the first name was, and then the name at the right will jump back in to where the moved up name was. This way, you can actually see a bubble sort at work. The bubble sort is so named because the lighter (lower ASCII value names) will "bubble" to the top of the list. (Or, you could just as well say that the "heavier" names "sink" to the bottom of the list.)

In this program we have set the delay as DL in line 120. By changing the value there, you can make the demo run as fast or slow as you want. The loop at line 170 reads the DATA items and prints them on the screen. The loop from lines 220 to 300 is the bubble sort. If a swap is made in line 260, then the GOSUB in line 270 will be activated and the subroutine at 340 will do the switching on the screen. The subroutine first clears the space where a name was. Then it prints that same name to the right of where it was. It then moves the next name on the list up one to the empty space. Following this it clears the space just made by having moved up the name. Then it moves the name at the right down one row and over to the space just created. In the first pass through the loop, you can see Wendell move from the very top of the list to the very bottom. Whenever a swap has to be made, variable F in line 280 is set to one. In line 300, if F is one it means that a swap has been made and we need to go through the loop again. This continues until F is equal to zero, and we are done and the list is in sorted order.

Notice how much manipulation we have to do in the subroutine to clear spaces and move names around. We do it mostly by adding or subtracting one from X (the row position).

If you go through these three demo programs carefully, you will certainly gain a new appreciation for direct cursor positioning. Aside from that, it's sort of fun to play around with it. ■

Cursor1.Bas

```
100 REM * Cursor1.Bas * Direct cursor positioning demo *
110 CLS
120 GOTO 160
130 LOCATE X,Y:RETURN
140 DATA Ann,Jerry,Henry,John,Roger,Patty,James,Oscar,Oliver,Wendell
150 '
160 FOR I=1 TO 10
170   READ A$(I)
180   PRINT A$(I)
190 NEXT I
200 FOR Q=1 TO 200:NEXT Q
210 '
220 Y=1
230 FOR X=1 TO 10
240   IF LEFT$(A$(X),1)="J" THEN GOSUB 130:PRINT"      ";A$(X);
250 FOR Q=1 TO 200:NEXT Q
260 NEXT X
270 X=12:Y=1:GOSUB 130:PRINT
280 END

Changes in cursor1.tr3 compared to cursor1.bas

Changed->130 PRINT@((X-1)*64)+(Y-1),;:RETURN
```

```
Changes in cursor1.tr4 compared to cursor1.bas

Changed->130 PRINT@((X-1),(Y-1)),;:RETURN
```

Cursor2.Bas

```
100 REM * Cursor2.Bas * demo of direct cursor positioning
110 CLS
120 GOTO 140
130 LOCATE X,Y:RETURN
140 A$="The quick brown fox jumped over the lazy dog."
150 X=5:C=1
160 FOR Y=LEN(A$) TO 1 STEP -1
170  GOSUB 130
180  PRINT MID$(A$,C,1)
190 FOR Q=1 TO 50:NEXT Q
200 C=C+1
210 NEXT Y
220 PRINT"Oops!"
230 '
240 X=7
250 FOR Y=1 TO LEN(A$)
260   GOSUB 130
270   PRINT MID$(A$,Y,1)
280   FOR Q=1 TO 50:NEXT Q
290 NEXT Y
300 '
310 X=5
320 FOR Y=LEN(A$) TO 1 STEP -1
330   GOSUB 130
340   PRINT " ";
350   FOR Q=1 TO 50:NEXT Q
360 NEXT Y
370 '
380 X=7
390 FOR Y=1 TO LEN(A$)
400   GOSUB 130
410   PRINT" "
420   FOR Q=1 TO 50:NEXT Q
430 NEXT Y
440 X=6:Y=1:GOSUB 130:PRINT STRING$(6,32)
```

```
Changes in cursor2.tr3 compared to cursor2.bas

Changed->130 PRINT@((X-1)*64)+(Y-1),;:RETURN
```

Tandy I/III changes

```
Changes in cursor2.tr4 compared to cursor2.bas

Changed->130 PRINT@((X-1),(Y-1)),;:RETURN
```

Tandy II/IV changes

Cursor3.Bas

```
100 REM * Cursor3.Bas * Direct cursor positioning demo *
110 CLS
120 DL=300
130 GOTO 150
140 LOCATE X,Y:RETURN
150 DATA Wendell,Betty,Oscar,Roger,Patty,James,Ann,Ruth,Carol,John
160 '
170 FOR I=1 TO 10
180   READ A$(I)
190   PRINT A$(I)
200 NEXT I
210 '
220 Y=1
230 FOR X=1 TO 9
240   L=X+1
250   IF A$(X)<=A$(L) THEN 290
260   T$=A$(X):A$(X)=A$(L):A$(L)=T$
270   GOSUB 340
280   F=1
290 NEXT X
300 IF F=1 THEN F=0:GOTO 230
310 X=13:Y=1:GOSUB 140:PRINT
320 '
330 END
340 GOSUB 140:PRINT STRING$(10,32);A$(L);
350 FOR Q=1 TO DL:NEXT Q
360 GOSUB 140:PRINT A$(X);
370 FOR Q=1 TO DL:NEXT Q
380 X=X+1:GOSUB 140:PRINT STRING$(30,32);
390 GOSUB 140:PRINT A$(X);
400 X=X-1:Y=10:GOSUB 140:PRINT STRING$(20,32);
410 FOR Q=1 TO DL:NEXT Q
420 Y=1
430 RETURN
```

Changes in cursor3.tr3 compared to cursor3.bas

Tandy I/III changes

```
Changed->120 DL=200
Changed->140 PRINT@((X-1)*64)+(Y-1),;:RETURN
```

Changes in cursor3.tr4 compared to cursor3.bas

Tandy II/IV changes

```
Changed->120 DL=500
Changed->140 PRINT@((X-1),(Y-1)),;:RETURN
```

Random Files

Now Sort Really BIG Files

Terry R. Dettmann, Associate Editor. This is the finish to what Terry started in the last issue. Ranidx.Bas now replaces Ranindex.Bas, and allows you to sort really big files.

Sort/Merge - The Final Chapter

In the last issue, we introduced the sort/merge concept by extending the random indexing program to handle larger files in small steps. As configured, we could handle up to 5000 records (more, if more could be held in memory). This issue, we'll go for the whole file no matter how large by allowing our program to sort the file section by section AND recover space on the disk by deleting files as we go.

We'll only have to replace one existing subroutine (and then only some lines of it) to allow this and we'll add a few new subroutines for specific purposes. Let's jump right into the procedure to see how it works.

What we've done so far

The random indexing program as it exists so far has been written to sort sections of the file in memory and write them to disk. Then we merge them back together again to give us a single file. The procedure falls apart if we need to sort a file too large for the number of records we can hold in memory, times the number of files we can have open at a time. We arbitrarily set the limit at 5000 records last issue to be sure everyone could do it. Now, we'll make the limit so large that only a large hard disk will have greater capacity than our file. Files which get that large will need other techniques (or a lot of time).

As it exists now, the procedure works like this:

1. Open the final output file
2. For i=1 to # of files in steps of 10
3. open the temporary output files
4. get the lowest record from the temp. files
5. if there is no lowest record, then 8.
6. write record to output file
7. go to 4.
8. Next i
9. Close all files

You can look at the subroutine at line 4000 in last issue's indexing program, Ranidx, to see how this basic procedure is actually implemented in BASIC.

How we change the procedure

To make this same procedure work for a larger data base, we can modify it like this:

1. set the output file number (n) to zero
2. For i=1 to # of files in steps of 10
2a. n = n + 1:open output file n
3. open the temporary output files
4. get the lowest record from the temp. files
5. if there is no lowest record, then 7a.
6. write record to output file n
7. go to 4.
7a. close all files
7b. delete temporary files
7c. rename output file n to nth temporary file
8. Next i
9. Close all files
10. if there has been more than 1 output file (n>1), then goto 1.
11. rename last output file to final output file name

We've included a replacement for subroutine 4000 with this article which implements the procedure. To follow it along, we start by setting N (the output file number to zero) and then entering the loop as before. Line 4045 calls subroutine 4400 to assign the name to the temporary output file and open it. Lines 4050 through 4070 remain the same (except that line 4060 branches to 4075 instead of 4080, the end of the loop). In line 4075, we close all the files, delete the temporaries to recover the disk space and then rename the output file to set it up as

a temporary for the next cycle.

When we've successfully merged all temporary files into a smaller number of output files, we leave the loop and check in line 4085 to see if there is more than one output file. If there is, we start over at the beginning and remerge to a new set of output files.

If there is only ONE output file, then we rename it to the desired output file name (subroutine 4700) and then return, we're done. The implementation is simple once we understand what we want to do.

To make all this work, we've added several short subroutines, each to handle a specific task. Subroutine 4400 assigns temporary file names and opens the file for output files. Given the output file number, N, it creates an output file named TMP#.XXX where '#' is the number N (TMP1.XXX, TMP2.XXX, and so forth).

Subroutine 4500 copies the procedure in subroutine 4100 for cycling through the temporary file names. Instead of opening them though, each one is deleted by the call to subroutine 4570 (you'll have to modify this subroutine since its exact statement form is dependent on your particular computer type).

Subroutines 4600 and 4700 perform file renames (once again using the BASIC 'SHELL' command). Again, you'll have to supply a specific format for the command which depends on your computer type. 4600 renames the output file to the name of a temporary file so we can merge it in the next round of merges. Subroutine 4700 does the final rename of the last output file to the final output file name desired.

As a procedure, it works (as we used to say in the Navy, 'works fine, lasts a long time, and drains to the bilge'). Each step is small enough that there should be no question about how the magic is accomplished. Taken in small steps like this, nearly all programming magic becomes simple. Have fun with it. ■

Program Notes

Lines 4580, 4620 and 4720 are ok for MS DOS users. For Tandy IV, use:

```
4580 SYSTEM "Remove "+FT$
4620 SYSTEM "NAME "+FX$+" "+FT$
4720 SYSTEM "NAME "+FT$+" "+FI$
```

Would anyone knowing how to run a DOS command from BASIC (and then return to BASIC) in Tandy III let us know about it?

We are including the complete listing of Ranidx.Bas here as well as the change lines to update the Ranidx.Bas from last issue to this version. This is intended to be a final version, and replaces Ranindex.Bas from previous issues.

```
10 REM --- RANIDX.BAS - Random File Indexing - VERSION 2.0
20 REM --- Terry R. Dettmann for Codeworks Magazine
25 MX=500
30 DIM FP$(20), SC$(24), XY(20,3)
31 DIM DA$(MX), IX(MX), IR(MX)
40 DEF FNCTR$(X$)=STRING$((WD-LEN(X$))/2," ")+X$
41 DEF FNLF(X) = LOF(X)/128
50 WD=80:LN=24
51 NX=0:TN=1:TX=10
60 FALSE=0:TRUE = NOT FALSE
100 REM --- file setup
110 CLS:PRINT FNCTR$("RANDOM FILE INDEXING"):PRINT:PRINT
120 LINE INPUT"FILENAME: ";FF$
125 FD$=FF$+".dat":FS$=FF$+".stk"
130 OPEN "R",1,FD$:OPEN"R",2,FS$,4:FIELD 2, 4 AS SK$
135 IF LOF(2)=0 THEN LSET SK$=MKI$(1):PUT 2,1
140 FM$=FF$+".MAP":FX$=FF$+".SCN"
150 GOSUB 5000: REM Read Map
170 GOSUB 5300: REM Setup Fielding
200 REM --- main menu
210 CLS:PRINT FNCTR$("RANDOM FILE INDEXING"):PRINT:PRINT
215 LINE INPUT"Name of the index: ";FI$:FI$=FI$+".idx"
```

```
220 INPUT"Sort on what field number";FX
230 IF FX<1 OR FX>CX THEN PRINT"OOPS - no such field":GOTO 220
231 INPUT"Select field number (enter 0 for none)";SX
232 IF SX=0 THEN 240
233 IF SX<1 OR SX>CX THEN PRINT"No such field number";GOTO 231
234 LINE INPUT"Select Criteria: ";SX$
240 FOR RN=1 TO FNLF(1):GOSUB 1400
250    IF FP$(1)="DELETED" THEN 270
255    IF SX>0 THEN IF INSTR(FP$(SX),SX$)=0 THEN 270
260    GOSUB 1000
265    IF NX>=MX THEN GOSUB 2000:TF=TN:GOSUB 3200:GOSUB 3300:NX=0:
       TN=TN+1
270 NEXT RN
280 IF NX>0 THEN GOSUB 2000:TF=TN:GOSUB 3200:GOSUB 3300
290 GOSUB 4000
500 REM --- End of Program
510 CLS:PRINT"All Done":CLOSE:END
550 REM --- Save the program
560 SAVE "ranidx.bas"
570 RETURN
600 REM --- input a character
610 C$=INKEY$:IF C$="" THEN 610
615 IF LEN(C$)>1 THEN GOSUB 700
620 RETURN
700 REM --- look for arrows
710 C = ASC(MID$(C$,2,1))
720 IF C=72 THEN C$=UP$ ELSE IF C=77 THEN C$=RT$
730 IF C=80 THEN C$=DN$ ELSE IF C=75 THEN C$=LF$
740 RETURN
800 REM --- GOTO XY ROUTINE
810 LOCATE X,Y:RETURN
900 REM --- break line
910 FOR K=1 TO 10:BL$(K)="":NEXT K
920 JN$=IN$:NB=1
930 K = INSTR(JN$,":"):IF K=0 THEN BL$(NB)=JN$:RETURN
940    BL$(NB) = MID$(JN$,1,K-1)
950    NB = NB + 1
960    JN$ = MID$(JN$,K+1)
970 GOTO 930
1000 REM --- Add the record to the index
1010 NX = NX + 1
1020 DA$(NX) = FP$(FX)
1030 IX(NX) = NX:IR(NX)=RN
1040 RETURN
1400 REM --- get record from data base
1410 IF RN<1 OR RN>FNLF(1) THEN RETURN
1420 GET 1,RN
1430 RETURN
2000 REM --- Sort the index
2010 DF = NX:PRINT "SORTING ..."
2020 IF DF = 1 THEN RETURN
2030    DF = INT(DF/2)
2040    SWP = FALSE
```

```
2050     FOR I=1 TO NX-DF
2060        IF DA$(IX(I))>DA$(IX(I+DF)) THEN GOSUB 2100:SWP = TRUE
2070     NEXT I
2080     IF SWP THEN 2040 ELSE 2020
2100 REM --- swap the data fields
2110 T = IX(I):IX(I) = IX(I+DF):IX(I+DF) = T
2120 RETURN
3200 REM --- Select Temporary File Name
3210 FT$="SRT"+MID$(STR$(TF),2)+".TMP"
3220 RETURN
3300 REM --- Save the Sorted data to a Temporary File
3310 PRINT "Saving Temporary File ";FT$
3320 OPEN "O",3,FT$
3330 FOR I=1 TO NX
3340     PRINT #3,IR(IX(I));",";DA$(IX(I))
3350 NEXT I
3360 CLOSE #3
3370 RETURN
4000 REM --- Merge Data from Temporary Files to Index
4010 CLOSE
4020 N=0
4030 NR=1
4040 FOR I=1 TO TN STEP TX
4045     N = N + 1:GOSUB 4400
4050     GOSUB 4100
4060     GOSUB 4200:IF NOT FOUND THEN 4075
4070     GOSUB 4300:GOTO 4060
4075 CLOSE:GOSUB 4500:GOSUB 4600
4080 NEXT I
4085 IF N>1 THEN TN=N:GOTO 4000
4086 GOSUB 4700
4090 CLOSE:RETURN
4100 REM --- open temporary files
4110 IF I+TX > TN THEN JX=TN ELSE JX=I+TX
4115 K=0
4120 FOR J=I TO JX:K=K+1
4130     TF=J:GOSUB 3200
4140     OPEN "I",K+1,FT$:INPUT#K+1,IX(K),DA$(K)
4141     PRINT"FILE: ";K;" ENTRY=";IX(K);DA$(K)
4150 NEXT J
4160 RETURN
4200 REM --- get lowest entry
4210 LW=1:FOUND = TRUE
4220 FOR J=2 TO K
4230     IF DA$(J)<DA$(LW) THEN LW=J
4240 NEXT J
4245 IF DA$(LW)="~~~" THEN FOUND=FALSE:RETURN
4250 IX = IDX(LW):IX$ = DA$(LW)
4255 IF EOF(LW+1) THEN DA$(LW)="~~~":RETURN
4260 INPUT#LW+1,IDX(LW),DA$(LW)
4265 PRINT"FILE: ";LW;" ENTRY=";IX(LW);DA$(LW)
4270 RETURN
4300 REM --- save to index
```

```
4310 PRINT "ITEM (";NR;") = ";IX$
4320 LSET XX$ = MKI$(IX):PUT #1,NR:NR=NR+1
4330 RETURN
4400 REM --- open intermediate file n
4410 FX$ = "TMP"+MID$(STR$(N),2)+".XXX"
4420 OPEN "R",1,FX$,2
4430 FIELD 1, 2 AS XX$
4440 RETURN
4500 REM --- delete temporary files
4510 IF I+TX > TN THEN JX=TN ELSE JX=I+TX
4515 K=0
4520 FOR J=I TO JX:K=K+1
4530    TF=J:GOSUB 3200
4540    GOSUB 4570
4550 NEXT J
4560 RETURN
4570 REM --- delete the named file
4580 SHELL "erase "+FT$
4590 RETURN
4600 REM --- rename output file
4610 TF = N:GOSUB 3200
4620 SHELL "ren "+FX$+" "+FT$
4630 RETURN
4700 REM --- rename last output file to final name
4720 SHELL "ren "+FT$+" "+FI$
4730 RETURN
5000 REM --- read data map
5001 CX = 0
5005 OPEN"I",3,FM$
5010 IF EOF(3) THEN 5035
5015    LINE INPUT#3,IN$
5020    GOSUB 900
5025    GOSUB 5100
5030 GOTO 5010
5035 CLOSE#3
5040 RETURN
5100 REM --- decode map line
5110 IF BL$(1)="FIELD" THEN GOSUB 5200:RETURN
5120 RETURN
5200 REM --- define a field
5210 NF = VAL(BL$(2)):FL = VAL(BL$(4)):FP = VAL(BL$(5))
5220 XY(NF,0)=FL:XY(NF,3)=FP
5225 CX = CX + 1
5230 RETURN
5300 REM --- Map Fields
5310 FOR I=1 TO CX
5320    NL = XY(I,3)
5330    FIELD #1, NL-1 AS X$,XY(I,0) AS FP$(I)
5340 NEXT I
5350 RETURN
```

If you have already entered Ranidx.Bas from last issue, the following change lines should be applied to bring it up to the version of Ranidx.Bas listed in this issue.

```
Changed->4020 N=0
Changed->4030 NR=1
Added--->4045     N = N + 1:GOSUB 4400
Changed->4060     GOSUB 4200:IF NOT FOUND THEN 4075
Added--->4075 CLOSE:GOSUB 4500:GOSUB 4600
Added--->4085 IF N>1 THEN TN=N:GOTO 4000
Added--->4086 GOSUB 4700
Added--->4400 REM --- open intermediate file n
Added--->4410 FX$ = "TMP"+MID$(STR$(N),2)+".XXX"
Added--->4420 OPEN "R",1,FX$,2
Added--->4430 FIELD 1, 2 AS XX$
Added--->4440 RETURN
Added--->4500 REM --- delete temporary files
Added--->4510 IF I+TX > TN THEN JX=TN ELSE JX=I+TX
Added--->4515 K=0
Added--->4520 FOR J=I TO JX:K=K+1
Added--->4530     TF=J:GOSUB 3200
Added--->4540     GOSUB 4570
Added--->4550 NEXT J
Added--->4560 RETURN
Added--->4570 REM --- delete the named file
Added--->4580 SHELL "erase "+FT$
Added--->4590 RETURN
Added--->4600 REM --- rename output file
Added--->4610 TF = N:GOSUB 3200
Added--->4620 SHELL "ren "+FX$+" "+FT$
Added--->4630 RETURN
Added--->4700 REM --- rename last output file to final name
Added--->4720 SHELL "ren "+FT$+" "+FI$
Added--->4730 RETURN
```

EASYDATE - by Dexter Walker, Birmingham, Alabama

In my custom programming work, I have an opportunity to experiment with different routines, discarding many but keeping a few for use in many of the programs I write. EASYDATE is a routine I use any time a program calls for a date entry, because it is quick, and because I find it actually cuts down on entry errors.

In use, any time we call for a date entry, the date is entered as D9$ and then run through this subroutine. With this routine in the program, the user can enter the date just about any way he or she wants to, with or without the slash which most programs require. Entry of 06/12/88, for instance, can be done with 06/12, 0612, 06.12.88, 06-12, or 06-12-88, all of which will return 06/12/88.

Line 15010 and 15020 convert either the dash or period to the conventional slash if there are any of either in the entry. Lines 15030 to 15070 then check each of the combinations, inserting slashes where needed, and providing the year from the system if it is missing. Each DOS type will have a different variable holding the date, and early in the program you will want to define the year in some way to make it available for this routine.

This short routine is worth looking at even if you don't use dates much. It starts off using the INSTR command, which is a very powerful tool, and in the first two lines also uses the MID$ command. The rest of the routine uses LEFT$, MID$ and RIGHT$ and patches parts of the date together with the plus sign (concatenation), then finally, if you entered something completely out of reason, it simply asks you to start over. A most useful routine, which you can use with the MERGE command, and delivers every date in a file to you in a uniform manner.

If you are not using MS DOS, you may need to change the MID$(DATE$,9,2) in two places (lines 15060 and 15070) to suit your operating system. What you want to do here is to pick off the last two digits of the year.

```
10 INPUT"enter a date with periods, dashes or slashes ";D9$
20 GOSUB 15010
30 PRINT D9$
40 END
14990 '
15000 ' date entry subroutine
15010 X=INSTR(D9$,"."):IF X>0 THEN MID$(D9$,X,1)="/":GOTO 15010 ELSE
      15020
15020 X=INSTR(D9$,"-"):IF X>0 THEN MID$(D9$,X,1)="/":GOTO 15020 ELSE
      15030
15030 L9=LEN(D9$):I1=INSTR(D9$,"/"):I2=INSTR(I1+1,D9$,"/")
15040 IF L9=8 AND I1=3 AND I2=6 THEN RETURN
15050 IF L9=6 AND I1=0 AND I2=0 THEN D9$=LEFT$(D9$,2)+"/"+MID$(D9$(3,
      2)+"/"+RIGHT$(D9$,2):RETURN
15060 IF L9=5 AND I1=3 AND I2=0 THEN D9$=D9$+"/"+MID$(DATE$,9,2):
      RETURN
15070 IF L9=4 AND I1=0 AND I2=0 THEN D9$=LEFT$(D9$,2)+"/"+RIGHT$(D9$,
      2)+"/"+MID$(DATE$,9,2):RETURN
15080 PRINT:INPUT"Invalid Date Entry - please re-enter (MM/DD/YY) ";
      D9$:GOTO 15020
```

Dmaker.Bas

An aid to Decision Making

Staff Project. Here is an aid in making decisions. It is especially helpful when you have many things to consider and each of them has several attributes that also need to be accounted for.

When you can only take one course of action the decision of what to do is easy. You may not like the choice, but if there is only one you have nothing against which to weigh your decision. Even faced with two or three choices, the decision may not necessarily be a difficult one to make.

But what if you have a half- dozen or more choices, each with another dozen attributes, and you need to pick the best choice from the lot? You may encounter such problems when you decide to buy a new automobile and are faced with five different makes or models. Ditto on buying a new computer. Or, you may even have such a decision to make when evaluating applicants for employment. Given such decision problems, do you analyze the possibilities with objectivity? How confident are you in the final course you have chosen?

Your computer can help you make such decisions. Notice the word is "help" because the computer does not actually make the decision - you do. The solution to most big problems is usually found by breaking the main problem down into smaller parts and making easy decisions on them, then combining the answers for a solution to the entire problem. The computer can help in this regard by guiding you to look at each part of the problem (without forgetting any of them along the way) and leading you to the most logical conclusion.

But we're talking about subjective judgments here; how is the computer going to determine what you mean by "good" or "better" attributes? How can such subjective evaluations be quantified?

Several years ago, the French developed a Multicriteria Decision-Aid program called *Electre*. A BASIC version of that program can be found in the book *Executive Planning with BASIC*, by X. T. Bui (Sybex, 1982). Electre works with a matrix denoting concordance, another denoting discordance and another matrix which is called the "outranking" matrix. Like most programs of this type it considers a set of alternatives against another set of criteria or attributes.

Our program, Dmaker.Bas, also considers alternatives against attributes, but does it in a way unlike that of Electre. In our case, the alternatives are given as the things among which you need to make a choice, i.e., Car A, Car B, etc. The attributes (or criteria) are then given as, for example, price, service, comfort, etc. Each alternative is rated against each of the attributes on some numerical scale. Then, the attributes themselves are rated *against each other* to find a numerical value. In other words, we are rating the ratings. While we are rating the attributes against each other we can, at the same time, check for consistency in our ratings and let that become a factor in the final answer. In fact, the consistency factor can become a percentage of assurance that we have chosen the best alternative.

Because the program is rather complex we will include a sample run using three cars with three attributes so that you can check for proper operation of the program. Keep in mind, however, that you may have as many as 20 alternatives with 20 attributes for each.

The Program

Two double-subscripted arrays, the A(x,x) and the S(x,x) arrays, are dimensioned to 20,20 in line 160. The A(x,x) array will contain the attributes and the S(x,x) array will contain the rating of the

alternatives. From here until line 440 is all housekeeping and heading information.

In lines 440 to 510 we establish both the number of alternatives to consider and their names. Variable N1 tells how many, and if the number is less than two, we don't need a program to do it, so line 470 sends us back to 460 to ask the question again. The names of the alternatives are held in array N$(I).

The number of attributes and their names are input in lines 530 to 620. The number of attributes is held in variable N, while the names are held in C$(I). But what is the A(I,I)=1 doing here? It is a convenient place to initialize the A(x,x) array. This array will contain the attributes, and if there are three, for example, then a three by three array will be created. The A(I,I)=1 will then put a one in the upper left space, the center space and the lower right space. The other positions of the array will be filled in later.

The next section of code from 640 to 760 rates the alternatives and fills the S(x,x) array. Each alternative is rated from one to nine against each of the attributes. The one to nine rating is used here for consistency only. Actually, these numbers should be between zero and one, so in line 730 we divide your input by 10.

The rating of the attributes against each other takes place from line 790 to 990. Here is where the remainder of the A(x,x) array is filled in. Notice that in lines 960 and 970, not only the value, but its reciprocal as well, is inserted into the array.

We have now arrived at the crux of the program. It sits between lines 1010 and 1340. Let's start with a dictionary definition: *Characteristic Root*, 1. a scalar for which there exists a nonzero vector such that the scalar times the vector equals the value of the vector under a given linear transformation on a vector space. 2. a root of the characteristic equation of a given matrix. Also called *eigenvalue*.

Ouch!

Shall we just say that it works, and go on? Naw - let's give it a shot. Basically, the code here refines its calculations repeatedly, looking for the dominant or characteristic value. In the end, this characteristic root will tell us which of our options is the most likely to be the best one.

The code from 1020 to 1050 is executed only once. It initializes arrays E(x) and B(x), putting the reciprocal of the number of attributes in each of the array spaces. Both arrays at this point contain the same values, because one of them will be modified while the other will be retained for future reference. In our sample run, for example, with three attributes, each of these array spaces will contain .33333 since we have three attributes to consider.

The code from 1060 to 1120 now multiplies the B(x) array by the vertical columns of the A(x,x) (attribute) array. The R(x) array will contain the sum of the products obtained. Now, in lines 1140 to 1160, variable EM becomes the sum of the R(x) array. Lines 1170 to 1190 put new values into the E(x) array. These values are the result of dividing each R(x) by the sum of all the R(x) values. Lines 1200 to 1230 create a new array called C(x). It contains the absolute (without sign) value of B(x) less E(x). If C(x) less .001 is less than zero the process ends here, otherwise the refinement process repeats by going back to line 1060 until C(x) is less than zero. If it is necessary to repeat the loop, lines 1250 to 1270 set new values into the B(x) array first. The B(x) array is the one that contains the changing values each time through the loop (evaluated in line 1090). It is interesting to note that the values in both the A(x,x) and S(x,x) arrays never change, their contents are only used as input to the other arrays used in this section of code.

The code from lines 1300 to 1340 now calculates the degree of consistency of your rating of the attributes. Q should be a number between zero and one. If you were perfectly consistent, Q will be zero, otherwise, if you were totally inconsistent, Q will be one or greater than one. In lines 1390, if Q is greater than one, we assume you cannot trust the results and print a statement to that effect.

Variable N1 is the number of alternatives, and in lines 1400 to 1460 we apply the values from the analysis routine to the alternative array. Array DC(x) will hold the values associated with each alternative. The best choice of alternatives will have the highest value, and in lines 1490 to 1560 we use a bubble sort to sort the alternatives into descending order.

Lines 1590 to 1610 print the sorted list of alternatives on the screen. Lines 1640 to 1660 tell us what the best choice is and also the degree of confidence we can have in that choice. The degree of confidence is a rather loose interpretation of the result but should suffice as a relative indicator in any case.

In running the program we found that our own prejudice for or against one alternative or another won out when the number of alternatives was small. It does better exactly where it should, when the number of alternatives and attributes exceed about five or six.

Use the sample run to test the correct operation of the program. ■

```
100 REM * Dmaker.Bas * A Decision making program * CodeWorks
110 REM * Magazine 3838 S Warner St. Tacoma, WA 98409
120 REM * (206) 475-2219 voice (206) 475-2356 300/1200 modem
130 REM * (c)1988 80-NW Publishing Inc & placed in public domain
140 '
150 'CLEAR 2000 'use if your BASIC is prior to ver. 5.0
160 DIM A(20,20),S(20,20),B(20),C(20),D(20),E(20),R(20)
170 DIM DC(20),T(20),N$(20),T$(20),C$(20)
180 '
190 'Universal print@/locate subroutine, un-remark as needed.
200 GOTO 260
210 LOCATE X,Y:RETURN 'MS-DOS, GW-BASIC
220 'PRINT@((X-1)*64)+(Y-1),;:RETURN 'Tandy models I/III
230 'PRINT@((X-1),(Y-1)),;:RETURN 'Tandy model IV
240 'PRINT@(X,Y),;:RETURN 'some MBASIC machines
250 'PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN 'CP/M, adjust
    to suit.
260 '
270 CLS ' clear the screen and home the cursor. Adjust to your needs.
280 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
290 PRINT"        D E C I S I O N   M A K I N G   P R O G R A M
300 PRINT"                an aid to making tough decisions
310 PRINT STRING$(60,45)
320 PRINT
330 PRINT"  This program will allow you to enter up to 20
    alternatives
340 PRINT"and 20 attributes pertaining to those alternatives. It will
350 PRINT"then ask you to evaluate the alternatives and the
    attributes.
360 PRINT"It will check the consistency of your answers and give you
370 PRINT"a reasonable best choice of alternatives.
380 PRINT"  The program contains no magic, but guides you in making
390 PRINT"rational choices.
400 PRINT
410 LINE INPUT"Press Enter to begin ";XX$
420 '
430 ' input the alternatives
440 CLS:PRINT TAB(10);"Establish the Alternatives"
450 PRINT
460 INPUT"How many alternatives will you consider ";N1
470 IF N1<2 THEN 460
480 X=5:Y=1:GOSUB 210
490 FOR I=1 TO N1
500   PRINT"What is the name of alternative";I;"? ";:LINE INPUT N$(I)
510 NEXT I
520 '
530 'input the attributes to be considered
540 CLS:PRINT TAB(10);"Now, we need the attributes to be considered."
550 PRINT
560 PRINT"How many attributes will you consider for
570 INPUT"each alternative ";N
580 IF N<2 THEN 560
590 FOR I=1 TO N
600   A(I,I)=1
```

```
610   PRINT"What is attribute number ";I;"? ";:LINE INPUT C$(I)
620 NEXT I
630 '
640 'rate the alternatives
650 CLS:PRINT TAB(10);"Next, we will rate the alternatives."
660 PRINT
670 PRINT"For each case, enter a value between 1 and 9
680 PRINT"where a higher number always means BETTER."
690 PRINT
700 FOR I=1 TO N
710  FOR J=1 TO N1
720    X=6:Y=1:GOSUB 210:PRINT"Enter ";C$(I);" rating for ";N$(J);
730    INPUT S(I,J):S(I,J)=S(I,J)/10
740    X=6:Y=1:GOSUB 210:PRINT STRING$(60,32)
750  NEXT J
760 NEXT I
770 '
780 ' rating the attributes
790 CLS:PRINT"Rating of the attributes."
800 PRINT
810 PRINT"Next, we will find the relative importance to you
820 PRINT"of the attributes when compared to each other.
830 PRINT
840 PRINT"For each pair of attributes enter 1 or 2 to indicate which
    is"
850 PRINT"more important, then enter a value from 1 to 9 to indicate"
860 PRINT"how much more important, where 1 indicates equal importance"

870 PRINT"and 9 indicates extreme importance of one over the other."
880 PRINT
890 FOR I=2 TO N
900   FOR J=1 TO I-1
910     X=11:Y=1:GOSUB 210:PRINT"1 - ";C$(J);TAB(20);"2 - ";C$(I)
920     PRINT
930     INPUT"Which is more important";F
940     INPUT"How much more important is it";G
950     FOR X=11 TO 14:Y=1:GOSUB 210:PRINT STRING$(60,32):NEXT X
960     IF (F=2) THEN A(I,J)=G:A(J,I)=1/G
970     IF (F=1) THEN A(J,I)=G:A(I,J)=1/G
980   NEXT J
990 NEXT I
1000 '
1010 REM * analysis routine
1020 FOR I=1 TO N
1030   E(I)=1/N
1040   B(I)=E(I)
1050 NEXT I
1060   FOR I=1 TO N
1070     TS=0
1080     FOR J=1 TO N
1090       TS=TS+B(J)*A(I,J)
1100     NEXT J
1110     R(I)=TS
1120   NEXT I
```

```
1130    EM=0
1140    FOR I=1 TO N
1150      EM=EM+R(I)
1160    NEXT I
1170    FOR I=1 TO N
1180      E(I)=R(I)/EM
1190    NEXT I
1200    FOR I=1 TO N
1210      C(I)=ABS(B(I)-E(I))
1220      IF (C(I)-.001)>0 THEN 1250
1230    NEXT I
1240    GOTO 1300
1250    FOR I=1 TO N
1260      B(I)=E(I)
1270    NEXT I
1280 GOTO 1060
1290 '
1300 FOR I=1 TO N
1310   D(I)=E(I)*N
1320 NEXT I
1330 EU=(EM-N)/(N-1)
1340 Q=SQR(EU/2)
1350 '
1360 'print the conclusions
1370 CLS:PRINT TAB(10);"Here are the results:"
1380 PRINT
1390 IF Q>1 THEN PRINT"Your answers were inconsistent. Don't trust
     the results."
1400 FOR J=1 TO N1
1410   DC(J)=1E+09
1420   FOR I=1 TO N
1430     S(I,J)=S(I,J)^D(I)
1440     IF S(I,J)<DC(J) THEN DC(J)=S(I,J)
1450   NEXT I
1460 NEXT J
1470 '
1480 'sort the alternatives into descending order
1490 FOR I=1 TO N1-1
1500   L=I+1
1510   IF DC(I)=>DC(L) THEN 1550
1520   T=DC(I):DC(I)=DC(L):DC(L)=T
1530   T$=N$(I):N$(I)=N$(L):N$(L)=T$
1540   FL=1
1550 NEXT I
1560 IF FL=1 THEN FL=0:GOTO 1490
1570 '
1580 PRINT"This is the ranking of your alternatives:"
1590 FOR I=1 TO N1
1600   PRINT I;"- ";N$(I)
1610 NEXT I
1620 PRINT
1630 IF Q>1 THEN 1660
```

```
1640 PRINT"Based on your input there is a";100-INT(Q*100);"percent
1650 PRINT"probability that ";N$(1);" is your best choice.
1660 PRINT:PRINT TAB(15);"End of analysis."
1670 END 'of program.
```

Change lines for Tandy I/III

```
Changes in dmaker.tr3 compared to dmaker.bas

Changed->100 REM * Dmaker/Bas * A Decision making program * CodeWorks
Changed->150 CLEAR 2000 'use if your BASIC is prior to ver. 5.0
Changed->210 'LOCATE X,Y:RETURN 'MS-DOS, GW-BASIC
Changed->220 PRINT@((X-1)*64)+(Y-1),;:RETURN 'Tandy models I/III
Changed->1430     S(I,J)=S(I,J)[D(I)
```

Change lines for Tandy II/IV

```
Changes in dmaker.tr4 compared to dmaker.bas

Changed->100 REM * Dmaker/Bas * A Decision making program * CodeWorks
Changed->210 'LOCATE X,Y:RETURN 'MS-DOS, GW-BASIC
Changed->230 PRINT@((X-1),(Y-1)),;:RETURN 'Tandy model IV
```

Sample Run

```
        Establish the Alternatives

How many alternatives will you consider ? 3
What is the name of alternative 1 ? Car A
What is the name of alternative 2 ? Car B
What is the name of alternative 3 ? Car C
        Now, we need the attributes to be considered.

How many attributes will you consider for
each alternative ? 3
What is attribute number  1 ? price
What is attribute number  2 ? service
What is attribute number  3 ? comfort
        Next, we will rate the alternatives.

For each case, enter a value between 1 and 9
where a higher number always means BETTER.

Enter price rating for Car A? 8

Enter price rating for Car B? 5
```

```
Enter price rating for Car C? 2

Enter service rating for Car A? 3

Enter service rating for Car B? 6

Enter service rating for Car C? 9

Enter comfort rating for Car A? 1

Enter comfort rating for Car B? 5

Enter comfort rating for Car C? 7

Rating of the attributes.

Next, we will find the relative importance to you
of the attributes when compared to each other.

For each pair of attributes enter 1 or 2 to indicate which is
more important, then enter a value from 1 to 9 to indicate
how much more important, where 1 indicates equal importance
and 9 indicates extreme importance of one over the other.

1 - price          2 - service

Which is more important? 1
How much more important is it? 7



1 - price          2 - comfort

Which is more important? 1
How much more important is it? 4



1 - service        2 - comfort

Which is more important? 2
How much more important is it? 5
          Here are the results:

This is the ranking of your alternatives:
 1 - Car B
 2 - Car A
 3 - Car C

Based on your input there is a 83 percent
probability that Car B is your best choice.

               End of analysis.
```

Etax88.Bas

Estimating Quarterly Taxes

Col (Ret) John J. Betz, Jr., Gig Harbor, Washington If, like many self-employed or retired persons, you must file quarterly estimated income taxes, here is a program that will assist you in calculating how much.

If you have sweated through Form 1040 this year and can't face the thought of wading through your Form W-4 to figure what your withholding should be for next year, then this program is for you.

The program ETAX88.Bas incorporates and applies all the IRS changes for 1988 taxes and covers most tax situations up to an income of $250,000. A knowledge of tax preparation is not required. In addition, error traps are provided after each section of entries so that if you do make an error, it will be relatively easy to correct.

The Program

The first part of the program (lines 160-290) display the type of financial data needed to use the program and outlines the sequence in which the program will consider the data.

The second part of the program (lines 320-480) requires the input of data pertaining to your status and number of dependents.

The third section (lines 500-590) computes your standard deduction based on your status previously input into the program.

The next section (lines 610-700) requires input of income from social security, capital gains and all other sources. The reason for this breakdown is that social security is only partially taxed, based on your status and income and capital gains may be taxed at a different rate from the rest of your income.

If it is applicable, the amount of social security subject to tax is computed by a sub-program at lines 1430-1600, which applies the pertinent parameters.

Lines 750-870 display items which are directly deductible from income and require the input of the total of these deductions. Lines 880-890 then compile and compute total income (TT) and adjusted gross income (AG).

You are next given the option (lines 910-1000) of using the standard deduction or to itemize deductions. If you elect to itemize, the program computes and displays the limitations that apply to medical, casualty losses and miscellaneous deductions (lines 1060-1080). You must then input the total of your itemized deductions (TJ) at line 1110.

The program next computes the amount of income subject to tax. This is done at line 1160, which selects the greater of the standard deduction (SD) and the total itemized deductions (TJ) and deducts it from adjusted gross income (AG). Line 1170 then computes the allowance for dependents (AD) and deducts it from the previous total.

At line 1240 the program deletes capital gains (if any), as they may be taxed at a different rate.

The actual computation of the tax is done in a sub- program beginning at line 1620. Although there are five different filing categories, two of them (Joint Return, where FS=2, and Qualifying Widower, where FS=5) share the same criteria. Line 1630 combines the two.

Although wide publicity has been given to the statement that there are only two brackets (15 and 28%), there are actually four. The program arranges the four into three groups. They are identified in the sub-program as the Low, High and Super Brackets. In the Low Bracket there are two levels of taxation (15 and 28%) that deal with three different levels of income criteria. In the tax instructions, these combinations require eight separate formulas to compute the tax. To simplify the program, one formula (line 1710) covers all the variations and the variables for each category are contained in the DATA statements at lines 1780-1850. One other item (the tax bracket, BR) is added as the last element of the DATA statements, as it may be required later in the main program to compute the capital gains tax, and this is accomplished in line 1720. The final step is to syphon off the incomes that qualify for the High Bracket and this is done in lines 1730- 1760.

There is only one variable required for each filing status in the High Bracket and these are provided in line 1880. The formula at line 1890 uses the tax previously determined in the Low Bracket and adds an additional tax level for this bracket. As was done previously, incomes that qualify for the Super Bracket are syphoned off by lines 1910-1950. (Note: If you wish to change the range of incomes that the program can handle, you can change the 250000, which is the third element in the data statements, lines 1790, 1810, 1830 and 1850, to the desired level.)

The same procedure that was used in the High Bracket is used in the Super Bracket with the pertinent variables in line 1980. The Super Tax is based on the lower of two formulas. The first is based on income (line 1990) and the second (line 2000) is based on the number of dependents. Line 2010 selects the lower of the two.

After the tax is computed, the program returns to the main program where at lines 1230-1260 the capital gains tax is computed, if applicable.

All the pertinent data is then displayed on the screen by lines 1280-1400. ■

```
100 REM ETAX88.BAS -COMPUTE ESTIMATED INCOME TAX FOR 1988
110 REM written by John J. Betz, Jr.
120 Q$="Press any key when ready to proceed."
130 Z$="Check the entries. Are they correct (Y/N)."
140 'WIDTH 80
150 '
160 REM INITIAL INSTRUCTIONS
170 CLS:PRINT TAB(30)"Estimated Tax Program"
180 PRINT "1. To use this program you will need the following:"
190 PRINT TAB(10)"a. Estimated income from capital gains(if any)."
200 PRINT TAB(10)"b. Income from social security(if any)."
210 PRINT TAB(10)"c. Income from all other sources."
220 PRINT TAB(10)"d. Adjustments to income."
230 PRINT TAB(10)"e. Total of your itemized deductions if any."
240 PRINT "2. The program will consider the following in the order
    shown."
250 PRINT TAB(10)"a. Filing status."
260 PRINT TAB(10)"b. Number of dependents."
270 PRINT TAB(10)"c. Income."
280 PRINT TAB(10)"d. Ajustments to income."
290 PRINT TAB(10)"e. Itemized deductions."
300 PRINT Q$;:INPUT X$
310 '
320 CLS:PRINT TAB(34)"FILING STATUS"
330 PRINT :PRINT "1-Single."
340 PRINT "2-Married Filing Jointly."
350 PRINT "3-Married Filing Seperately."
360 PRINT "4-Head of Household."
370 PRINT "5-Qualified widow(er)."
380 PRINT :INPUT "Enter the number of your status.";FS
390 PRINT :PRINT "Answer yes or no (Y/N) to the following."
400 PRINT :INPUT "Is the filer 65 or older.";FA$
410 INPUT "Is the filer blind.";FB$
420 IF FS<>2 THEN GOTO 450
430 PRINT :INPUT "Is your spouse 65 or older.";SA$
440 INPUT "Is your spouse blind.";SB$
450 PRINT :INPUT "Enter your total number of dependents including
    yourself.";ND
460 PRINT :PRINT Z$;:INPUT X$
470 IF X$="N" OR X$="n" THEN GOTO 320
480 IF X$<>"Y" AND X$<>"y" THEN GOTO 460
```

```
490 '
500 REM COMPUTE STANDARD DEDUCTION - SD
510 IF FA$="Y" OR FA$="y" THEN FA=1
520 IF FB$="Y" OR FB$="y" THEN FB=1
530 IF SA$="Y" OR SA$="y" THEN SA=1
540 IF SB$="Y" OR SB$="y" THEN SB=1
550 TA=FA+FB+SA+SB
560 IF FS=1 THEN SD=3000+(TA*750)
570 IF FS=2 OR FS=5 THEN SD=5000+(TA*600)
580 IF FS=3 THEN SD=2500+(TA*600)
590 IF FS=4 THEN SD=4400+(TA*750)
600 '
610 REM SOURCES OF INCOME
620 CLS:PRINT TAB(31)"SOURCES OF INCOME"
630 PRINT :PRINT "Enter the amounts of the following. Enter 0 if
    appropriate."
640 PRINT :INPUT "Social Security.";SS
650 PRINT :INPUT "Capital Gains.";CG
660 PRINT :INPUT "All other income.";AI
670 PRINT :PRINT Z$;:INPUT X$
680 IF X$="N" OR X$="n" THEN GOTO 620
690 IF X$<>"Y" AND X$<>"y" THEN GOTO 670
700 IF SS=0 THEN GOTO 750
710 '
720 REM COMPUTATION OF TAXABLE SOCIAL SECURITY
730 GOSUB 1430
740 '
750 REM ADJUSTMENTS TO INCOME
760 CLS:PRINT TAB(30)"ADJUSTMENTS TO INCOME"
770 PRINT :PRINT "ADJUSTMENTS TO INCOME INCLUDE THE FOLLOWING:"
780 PRINT TAB(10)"1. Reimbursed employee expenses."
790 PRINT TAB(10)"2. IRA deductions."
800 PRINT TAB(10)"3. Self employed health insurance."
810 PRINT TAB(10)"3. Keogh Plan or other SEP plans."
820 PRINT TAB(10)"4. Penalty on early savings withdrawal."
830 PRINT TAB(10)"5. Alimony paid."
840 PRINT :INPUT "Enter the total of the above if any.";TD
850 PRINT Z$;:INPUT X$
860 IF X$="N" OR X$="n" THEN GOTO 760
870 IF X$<>"Y" AND X$<>"y" THEN GOTO 850
880 TT=TS+CG+AI
890 AG=TT-TD
900 '
910 REM STANDARD DEDUCTION OPTION
920 CLS:PRINT TAB(31)"STANDARD DEDUCTION OPTION"
930 PRINT :PRINT "Your standard deduction amounts to:";SD
940 PRINT :IF FS<>3 THEN GOTO 960 ELSE INPUT "Will your spouse
    itemize deductions? Enter (Y/N).";SI$
950 IF SI$="N" OR SI$="n" THEN GOTO 960 ELSE PRINT "You cannot use
    the standard deduction and must itemize.":GOTO 1020
960 PRINT :PRINT "In order to be of benefit your itemized deductions "

970 PRINT"must exceed your standard deduction of:";SD
980 PRINT :INPUT "Do you wish to itemize. Enter (Y/N).";X$
990 IF X$="N" OR X$="n" THEN GOTO 1130
1000 IF X$<>"Y" AND X$<>"y" THEN GOTO 980
```

```
1010 '
1020 REM ITEMIZED DEDUCTION
1030 CLS:PRINT TAB(30)"ITEMIZED DEDUCTIONS"
1040 PRINT :PRINT "Before you enter your total of itemized"
1050 PRINT"deductions, check them against the following:"
1060 PRINT :PRINT "Medical deductions must exceed.";INT(AG*.075)
1070 PRINT "Casualty/theft losses must exceed.";INT(AG*.1)
1080 PRINT "Miscellaneous deductions must exceed.";INT(AG*.02)
1090 PRINT "Only 40% of personal interest (credit cards,"
1100 PRINT"  car loans, etc.) may be deducted."
1110 PRINT :INPUT "Enter the total of your itemized deductions";TJ
1120 '
1130 REM DETERMINING TAXABLE INCOME
1140 IF FS<>3 THEN GOTO 1160
1150 IF SI$="Y" OR SI$="y" THEN SD=0
1160 IF TJ>SD THEN TH=AG-TJ ELSE TH=AG-SD
1170 AD=ND*1950 :TI=TH-AD
1180 '
1190 REM DELETION OF CAPITAL GAINS
1200 IF CG<>0 THEN TI=TI-CG
1210 GOSUB 1620
1220 '
1230 REM COMPUTATION OF CAPITAL GAINS TAX
1240 IF CG=0 THEN GOTO 1280
1250 IF BR<>33 THEN CT=INT(CG*.28) ELSE  CT=INT(CG*.33)
1260 TX=TX+CT
1270 '
1280 REM DISPLAY OF RESULTS
1290 CLS:PRINT TAB(28)"ESTIMATED TAX DATA-1988"
1300 IF TS=0 THEN PRINT "Total income = $";TT ELSE PRINT "Total
     income including taxable Social Security of $";TS;"= $";TT
1310 PRINT :IF TD<>0 THEN PRINT "Adjustments to income = $";TD ELSE
     PRINT "Adjustments to income = $";0
1320 PRINT "Adjusted gross income = $";AG
1330 PRINT :IF SD>TJ THEN PRINT
     "Your standard deduction amounts to $";SD ELSE PRINT "Your
     itemized deductions amount to $";TJ
1340 PRINT "Your allowance for dependents = $";AD
1350 IF CG=0 THEN PRINT "Your taxable income = $";TI ELSE PRINT "Your
     taxable income less capital gains = $";TI
1360 PRINT "Your tax will be $";TX
1370 PRINT "To avoid a penalty you must withhold/prepay $";INT(TX*.9)+
     1
1380 PRINT "You are in the";BR;"% bracket.";"This means that"
1390 PRINT"  for every change of $1 in your taxable income, your"
1400 PRINT"  tax will change by";BR;"cents."
1410 END
1420 '
1430 REM SUB PROGRAM SOCIAL SECURITY
1440 TG=CG+AI
1450 CLS:PRINT TAB(20) "Social Security Data"
1460 PRINT "Enter the amount of tax free interest that you"
1470 PRINT"expect to receive. Enter 0 if appropriate.";:INPUT TF
1480 IF FS<>3 THEN 1500 ELSE PRINT "Will you be living with your
     spouse at any"
1490 PRINT"time during 1988? (Y/N)";:INPUT SP$
```

```
1500 PRINT :PRINT Z$:INPUT X$:IF X$="N" OR X$="n" THEN GOTO 1450 :IF
     X$<>"Y" AND X$<>"y" THEN GOTO 1500
1510 TB=INT(TG+TF+(SS/2))
1520 IF FS<>2 THEN  GOTO 1540
1530 TC=TB-32000:IF TC<1 THEN GOTO 1580 ELSE GOTO 1590
1540 IF FS=3 THEN GOTO 1560
1550 TC=TB-25000:IF TC<1 THEN GOTO 1580 ELSE GOTO 1590
1560 IF SP$="N" OR SP$="n" THEN TC=TB-25000 ELSE TC=TB
1570 IF TC>1 THEN GOTO 1590
1580 TS=0:GOTO 1600
1590 IF TC<INT(SS/2) THEN TS=TC ELSE TS=INT(SS/2)
1600 RETURN
1610 '
1620 REM SUBPROGRAM TAX TABLES
1630 IF FS=5 THEN FS=2
1640 '
1650 REM COMPUTATION-LOW BRACKET
1660 FOR I=1 TO 10
1670 READ V,A,B,C,D,E
1680 IF V<>FS THEN GOTO 1700
1690 IF TI>A AND TI<=B THEN GOTO 1710
1700 NEXT
1710 TX=INT((TI-A)*C)+D
1720 BR=E
1730 IF FS=1 AND TI>43150! THEN GOTO 1870
1740 IF FS=2 AND TI>71900! THEN GOTO 1870
1750 IF FS=3 AND TI>35950! THEN GOTO 1870
1760 IF FS=4 AND TI>61650! THEN GOTO 1870
1770 GOTO 2020
1780 DATA 1,0,17850,.15,0,15
1790 DATA 1,17850,250000,.28,2678,28
1800 DATA 2,0,29750,.15,0,15
1810 DATA 2,29750,250000,.28,4463,28
1820 DATA 3,0,14875,.15,0,15
1830 DATA 3,14875,250000,.28,2231,28
1840 DATA 4,0,23900,.15,0,15
1850 DATA 4,23900,250000,.28,3585,28
1860 '
1870 REM COMPUTATION-HIGH BRACKET
1880 IF FS=1 THEN A=43150!:IF FS=2 THEN A=71900!:IF FS=3 THEN
     A=35950!:IF FS=4 THEN A=61650!
1890 TX=TX+INT((TI-A)*.05)
1900 BR=33
1910 IF FS=1 AND TI>89560! THEN GOTO 1970
1920 IF FS=2 AND TI>149250! THEN GOTO 1970
1930 IF FS=3 AND TI>74625! THEN GOTO 1970
1940 IF FS=4 AND TI>123790! THEN GOTO 1970
1950 GOTO 2020
1960 '
1970 REM COMPUTATION SUPER BRACKET
1980 IF FS=1 THEN A=89560!:IF FS=2 THEN A=149250!:IF FS=3 THEN
     A=74625!:IF FS=4 THEN A=123790!
1990 TB=TX+INT((TI-A)*.05)
2000 TC=TX+INT((ND*10920)*.05)
2010 IF TB<TC THEN TX=TB ELSE TX=TC
2020 RETURN
```

ETAX88.Bas works as is on all models.
(How about that?)

Bio.Bas

Plot Your Personal Biorhythms

Staff Project. Computers and biorhythms seem to go together rather naturally. Although many such programs exist, we think ours is somewhat more straightforward and adds a couple of interesting embellishments.

Almost everything in our universe seems to be cyclic. The planets in our own solar system, for example, assume the same position every 93,408 years. That's a rather large cycle; one of the shorter cycles would be the effect of the moon on the tides. Many shorter cycles have been researched and plotted. There are eight-year cycles, 9.2-year cycles, 9.6-year cycles and 18.2-year cycles. Then, of course, there is the well-known 11-year sunspot cycle. Some of the things that rise and fall with these cycles don't seem to make sense, yet they follow the cycle rather faithfully. The number of patents issued in the United States, for example, rises and falls on a 9.2-year cycle. Flooding, volcanos, and home-building all follow regular cycles.

It is not surprising then, that someone would eventually detect cycles in life. These have become to be known as "Biorhythms." The man who first developed the theory of biorhythms was a German doctor, Wilhelm Fliess. Fliess was a contemporary and friend of Sigmund Freud, and around the turn of the century he theorized the existence of a 23-day and a 28-day cycle in humans. He theorized that these cycles began at the time of birth and are present in every cell of the body throughout life. They show themselves in the ups and downs of an individual's physical and emotional vitality. It was further stressed that the cycles are not influenced by outside forces, and do not determine what is going to happen to you, but more likely, how you are going to feel about it.

Fliess collected information pertaining to his cycles and eventually published a book on the subject. He expected his theories to produce a revolution in biology and medicine, but it never happened. Sigmund Freud, who at first agreed with Fliess, later quietly dropped his support of the biorhythm theory.

Dr. Hermann Swoboda, a professor of physiology at the University of Vienna, believed that his own research confirmed the 23- day and 28-day cycles of Fliess and added yet another element to the theory. He stressed what he called "critical days." These were the days in which the cycle crossed zero going from high to low or low to high. He also published extensively, but neither his or Fliess' works received professional acclaim.

The third cycle, the 33-day intellectual cycle, was proposed first by Alfred Teltscher in the 1920's. He collected data from tests students took in Innsbruck, Austria, and concluded that their performance varied in accordance with a 33-day cycle.

Calculating these cycles had been an error-prone chore until the advent of the computer. Several devices were available to assist in the calculations. There were slide rules, a set of circular disks and even books of tables that were used to find where in a given cycle you were on a particular day. Hand-held programmable calculators made the process simpler, and in the late 1970's the introduction of the personal computer made the process easy and more accurate.

Basically, the theory states that the three cycles, like sine curves, all start at the moment of birth

and continue for life. During those times when the cycle is above zero you are expending energy, when the cycle is negative you are accumulating energy and that when the cycles cross zero you have what Swoboda called a "critical day." The physical (23-day cycle) and the emotional (28-day cycle) will repeat every 644 days (23 x 28 = 644). However, when considering all three cycles together, there is no repeat until you are 21,252 days old (23 x 28 x 33 = 21252). You are 58 years, 2 months and 7 days old when this occurs. This is the first time, after the day of your birth, that the cycles all start exactly at zero and all go in a positive direction. That day would be called a "triple critical" day since all three cycles cross zero at the same time. So what are we expected to do on such a critical day; stay in bed in a pre-natal position with the electric blanket turned up to 9? The literature on the subject is not very clear on that, but does suggest that it would be unwise to engage in dare-devil feats on such a day.

It appears to us, from the available literature on the subject of biorhythms, that it is a pseudo-science disparately searching for acceptance. If it can't do better than it has until now, it will forever be classified in the same category with astrology and numerology. Perhaps there is something to it and the proper connections just haven't been found yet. Who says, for example, that the length of the three cycles are exact and don't vary based on some other influence? Will some future researcher come up with yet a fourth, or even a fifth, cycle? Does the moon figure in at all? One source (Gittelson, see references) seems to imply that there may be a connection with the moon, but stops short of telling exactly what that connection may be.

With all this in mind, we have developed Bio.Bas in such a way that should you want to carry on with some of your own research, it should be easy to adapt. This is one area where a person with a computer and some easily obtainable data can perhaps make some meaningful contribution.

In our program, we have included the phases of the moon. This may or may not have any bearing on biorhythms, but it is there if you need it. It can also be eliminated if you so desire. In doing some research on the position of the moon, we found some very interesting facts which we were unaware of. For example, in addition to the several more obvious motions of the moon, there are more than 500 minor motions! To get the phase of the moon as nearly correct as possible, we took the New Moon of the 4th of January, 1880 as a starting point and then, by trial and error, adjusted the period of the moon so that it would coincide with New Moon dates in a current almanac. In the program, line 220 gives the New Moon variable (NM) a value of 29.55738. This period at first seems to be too long, since the moon revolves around the earth in 27 days, 7 hours, 43 minutes and 11 seconds. But, digging deeper into the astronomy books (see reference 3), we find this is the *synodic* period. The *real* revolution (sidereal) turns out to be 29 days, 12 hours, 44 minutes and 3 seconds. When you convert the sidereal time to decimal form, you get rather close to our value for NM. As stated earlier, we did it the other way around, by fixing a starting date in the past and then adjusting NM until the New Moon dates for the current year fell into place. All of which means that there may be a small, cumulative error that may make the New Moon date shift one way or another for dates far in the past or far into the future.

The moon phase is not a standard part of most biorhythm theory. We have included it as an embellishment. Another such embellishment we have added is the relative index. This is a number between 0 and 10, and is supposed to give you an indication of what kind of day you are having. It is based on the three cycles and averaged. In addition, if any cycle is crossing zero, that value is reduced to zero for the calculation. We don't know if this approach has any validity, after all, who is to say that the three cycles have equal weight? Again, the literature on the subject did not expand on that aspect, except to say that the physical and emotional cycles *may* have more effect than the intellectual cycle.

If biorhythm theory works, we wondered why it wasn't in more common use. Apparently, in Japan and Switzerland, they are using them in an effort to reduce industrial accidents. Workers who have critical days are given less taxing work to do. In this country, according to Gittelson (see reference 1), United Airlines used biorhythms on their ground crews at National Airport in Washington, D.C. and cut their accident and injury rates in half. Others consider it an unadulterated fraud. However, to dismiss it out of hand is as unscientific as it is to accept it without proof. It is obvious that more work in this area needs to be done.

The Program

Essentially, all Bio.Bas does is to calculate where in the three biorhythm cycles we are on a given day. It then inserts the proper symbols into A$, adds the index, age and date information at the end, and prints it. A$ is then cleared and the

next day is calculated. This way, A$ does not even need to be an array.

We start the program with something different. Lines 160 to 180 contain defined functions. The first, in line 160 is a defined function to determine "even." The next, in line 170, determines "odd," simply by saying that if it isn't even it must be odd. Both line 160 and 170 figure into line 180. In line 180, function day (FNDA(M)) is going to tell us if a month has 28, 30 or 31 days. It turns out that the even numbered months from January through July are 30 days long, except for February, and that the odd numbered months from August to December are 30 days long. A leap year February, of course, is a special case and will be treated separately, later. The defined function in line 180 will take the number of any month and return the number of days in that month. It will always return 28 days for February. You can follow through line 180 more easily if you remember that it is simply a series of true/false statements. For example, the parenthetical statement (M<8) will be true (-1) when M is less than eight, otherwise it will be false (0). The same thing applies to FNO(M). If M is odd then FNO(M) will be equal to -1 (true), otherwise it will be 0 (false). Variable M, of course, will always be the number of the month we are dealing with.

In line 200 we need to dimension variable M$ at 12 so that we won't get a subscript out of range error. M$ is the literal, three letter, name of the month.

Line 210 defines the variable Pi to as many places as we need it for this program. In line 220 we define the variable NM (New Moon) as the period of the moon from one New Moon to the next.

Lines 230 and 240 are DATA statements that contain the days of the week and the months of the year, respectively. These are read into arrays D$(I) and M$(I) in lines 270 and 280.

Following this set-up, we now print the CodeWorks heading on the screen and prompt for both screen and printer output. Line 390 asks us to input our birthdate in month, day, year format, separated by commas.

Line 400 sends us to the total days calculator subroutine at line 1390. This subroutine calculates how many days have elapsed from 0,0,0 to your birthdate. It makes allowance for leap years and returns the total number of days in variable D1. When we return to line 400, variable K1 is set equal to variable D1.

Next, in line 420, we input the date to start the biorhythm chart. This time we go first to a subroutine at line 1240 to find what day of the week that date represents. Two things happen at this subroutine. First, variable Z will be a number representing the day of the week. It will be used with D$(Z) to tell us if the day is Mon, Tue, etc. Next, it will make a string value out of the day of the month, and if the day is less than 10, will insert an extra space in front of the number. The day of the month will be held in variable D1$.

When we return to line 430, we go immediately to the total days calculator subroutine again, at line 1390. This time we find out how many days have elapsed from 0,0,0 to the date to start the chart. Variable D1 will again hold this number of days, and when we return from the subroutine to line 440, variable T will subtract K1 from D1 and tell us how many days old we are on the day we start the chart.

The date to start the chart is still intact in M,D,Y. Next, in line 470, we go once again to the total days calculator subroutine and find the initial phase of the moon for the start of our chart. Since the moon phase is based on a given date (4 Jan 1880) in all cases, we don't need to figure that repeatedly. It was figured out ahead of time and hard-coded into line 470. The number is 686707. (Don't worry about the exclamation point, let the computer put it there if it wants to.) Now, when we return from the subroutine, variable T1 will give us the days from which to calculate the phase of the moon.

Next we print some instructions and headings on the screen (and the printer if you selected to print too). The printer function is a brute force one in which if you selected to print, then PR=1 and the LPRINT lines are activated.

In line 510 we input how many days to chart, variable CP. The moon is new when it passes from negative, through zero towards positive. This was hard to explain in a short screen prompt, so we simply say (in line 620) that max. + is = 1st quarter of the moon. It would be full moon, of course, when the moon goes through zero from positive to negative.

In line 690 we initially clear out A$ before we start filling it with all the data it will contain later. Line 700 sets a counter (CT) to zero. It will be used to tell us when to quit, by comparing CT to CP after each line is printed.

The first symbol we will position in A$ will be the intellectual symbol "i." The problem here is to convert a portion of the 33-day cycle into a position on a sine curve. Not only that, but we need to scale it to fit the available space. At this point we already know how many days old you are in variable T. In line 730 we find out what the remainder is when we divide your age in days by

33. This decimal fraction is held in variable F. In line 740 we convert F into radians and let X represent that value. The position in A$ (P) is calculated in line 750, where we take the sine of X and scale it to our available space. In the second statement in line 750, we re-scale P into variable P1 which will change the number into one between 0 and 10 for our index value. Line 760 simply says that if the number is between 4.5 and 5.5 then make the index value (P1) zero (in accordance with the biorhythm theory). Next, in line 770, we set up the zero boundaries on the graph by printing ": :" centered on column 26 on the screen, which is the midpoint of the graph. Finally, in line 780, we position the character "i" in A$ at position P. Keep in mind that we are still working with one day, are working with the filling of A$, and haven't printed anything yet.

The code from 810 to 850 does what we just did, except this time we are positioning the emotional symbol "e." Very little has changed, except that we are now dividing age in days by 28 instead of 33. Also, our scaling in line 830 is slightly different. Because a curve already written into A$ can be over-written by the next curve, we decided that we would let the internal values be real, but would scale the shorter period curves with less amplitude. This way, you could see most of the curves, most of the time. Consequently, the intellectual cycle and the moon phase have the largest amplitude, then the emotional cycle and then the physical cycle. Because the moon phase is put in last, however, it will over-write anything in its path.

The code from line 880 to 920 positions the physical symbol and the code from 950 to 980 positions the moon symbol. Now that we have the four symbols in A$ we can figure out the relative index number in line 1000. Variable SC holds this number, which is the average of P1, P2 and P3. In line 1010 variable SU accumulates the value of SC so that an average index for the complete chart can be given at the end.

Now that we have all our values in A$ it is time to append more information to the end of that string. In line 1040 we let A$ equal itself, then add, with an appropriate space, the index value for that day, then the age in days, then the day of the week, followed by the month, day and year. We then, finally, print A$ as one line on the screen or printer. As soon as A$ is printed, we clear it out in preparation for the next day. This happens in line 1070. We also, in line 1080, clear out the index variable (SC) to make it ready for the next day.

Getting ready for the next day involves just a little housekeeping. In line 1110 we use our defined function and let D2 equal the number of days in the current month. In line 1120 we check to see if the month is February, and if so and the year is not 1900, and the year is divisible by four, we increment D2 by one for the leap year. Next, in line 1130, we increment the day (the chart started) by one. In line 1140 if our just incremented day is greater than D2 (days in the month) we reset day (D) to one and increment the month by one. In line 1150 if the month we just incremented is greater than 12, we increment the year by one and reset the month number to one.

Line 1160 sends us to the find day of week subroutine at 1240 to find the day of the week for the new day. Then line 1170 sends us to the total days calculator to find the new day for the phase of the moon. We don't need to go to this same subroutine to find the days lived, since we know it will be one more than the previous day. So, in line 1190 we simply increment the days lived by one. Next, in line 1200 we increment CT, the count variable to indicate that we have just printed one day. Next, we check it against the number of days we want to print (CP) and if they are equal or CT is greater, then we print the average and are done. Otherwise, we go back to line 730 and print out the next day. ■

References:

1. *Biorhythm, A Personal Science,* Bernard Gittelson, Arco Publishing Co. Inc. New York 1975,76

2. *Biorhythms in Your Life,* Daniel Cohen, Fawcett Publications, Inc. Greenwich, CT 1976

3. *The Flammarion Book of Astronomy,* Simon & Schuster, New York 1964

Bio.Bas works as is on MS DOS and Tandy II/IV machines. Change lines for Tandy I/III can be found at the end of the listing.

```
100 REM * Bio.Bas * Biorhythm program. CodeWorks Magazine
110 REM * 3838 S. Warner St. Tacoma, WA 98409
120 REM * (206) 475-2219 voice, (206) 475-2356 300/1200 modem
130 REM * (C)1988 80-NW Publishing Inc and placed in public domain
140 '
150 'define some useful functions
160 DEF FNE(M)=(M-2*INT(M/2)=0) 'defined function even
170 DEF FNO(M)=NOT FNE(M) ' defined function odd
180 DEF FNDA(M)=30+(M<8)*FNO(M)+(M>7)*FNE(M)+2*(M=2) ' defined
    function days in month
190 'CLEAR 2000 ' use only if your BASIC is prior to ver. 5.0
200 DIM M$(12)
210 PI=3.14159
220 NM=29.55738 'average period new moon to new moon, see text.
230 DATA Sun,Mon,Tue,Wed,Thu,Fri,Sat
240 DATA Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
250 '
260 ' read in the data
270 FOR I=1 TO 7:READ D$(I):NEXT I
280 FOR I=1 TO 12:READ M$(I):NEXT I
290 '
300 CLS
310 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
320 PRINT"              B I O R H Y T H M   P R O G R A M
330 PRINT"                plot your personal biorhythms
340 PRINT STRING$(60,45)
350 PRINT
360 INPUT"Do you want printed output too (y/n)";AN$
370 IF AN$="y" OR AN$="Y" THEN PR=1 ELSE PR=0
380 LINE INPUT"Please enter your name ";N$
390 INPUT"now enter your birthdate (MM,DD,YYYY)";M,D,Y
400 GOSUB 1390:K1=D1
410 '
420 INPUT"Enter date to start the chart (MM,DD,YYYY)";M,D,Y
430 GOSUB 1240:GOSUB 1390 'get initial day of week, then total days
440 T=D1-K1:T=ABS(T) ' T holds total days lived
450 '
460 ' find initial moon phase based on 4 Jan 1880 New Moon
470 GOSUB 1390:T1=D1-686707! ' T1 holds days of moon phase
480 '
490 IF PR=1 THEN PRINT"Turn on your printer and align the paper."
500 PRINT"How many days do you want to chart?"
510 INPUT"52 days will fit on a standard sheet";CP
520 PRINT TAB(15)"Biorhythm Chart for ";N$
530 IF PR=1 THEN LPRINT TAB(15)"Biorhythm Chart for ";N$
540 PRINT" "
550 IF PR=1 THEN LPRINT" "
560 PRINT"p is the 23-day physical cycle
570 IF PR=1 THEN LPRINT"p is the 23-day physical cycle
580 PRINT"e is the 28-day emotional cycle
590 IF PR=1 THEN LPRINT"e is the 28-day emotional cycle
600 PRINT"i is the 33-day intellectual cycle
610 IF PR=1 THEN LPRINT"i is the 33-day intellectual cycle
620 PRINT". is the phase of the moon (max + = 1st Qtr.)
```

```
630 IF PR=1 THEN LPRINT". is the phase of the moon (max + = lst Qtr.)
640 PRINT TAB(10);"-";TAB(26)"O";TAB(42);"+
    ";TAB(51);"Index";TAB(57);"Age    Date"
650 IF PR=1 THEN LPRINT TAB(10);"-";TAB(26)"O";TAB(42);"+
    ";TAB(51);"Index";TAB(57);"Age    Date"
660 PRINT STRING$(75,45)
670 IF PR=1 THEN LPRINT STRING$(75,45)
680 '
690 A$=STRING$(52,32)
700 CT=0
710 '
720 'find position of intellectual symbol
730 F=(T/33)-INT(T/33)
740 X=F*2*PI
750 P=((SIN(X)+1)*24)+2:P1=(P*10)/50
760 IF P1>4.5 AND P1<5.5 THEN P1=0
770 MID$(A$,25,3)=": :"
780 MID$(A$,P,1)="i"
790 '
800 'find position of emotional symbol
810 F=(T/28)-INT(T/28)
820 X=F*2*PI
830 P=((SIN(X)+1)*22)+4:P2=(P*10)/50
840 IF P2>4.5 AND P2<5.5 THEN P2=0
850 MID$(A$,P,1)="e"
860 '
870 'find position of physical symbol
880 F=(T/23)-INT(T/23)
890 X=F*2*PI
900 P=((SIN(X)+1)*20)+6:P3=(P*10)/50
910 IF P3>4.5 AND P3<5.5 THEN P3=0
920 MID$(A$,P,1)="p"
930 '
940 ' find position of the moon symbol
950 F=(T1/NM)-INT(T1/NM)
960 X=F*2*PI
970 P=((SIN(X)+1)*24)+2
980 MID$(A$,P,1)="."
990 '
1000 SC=((P1*.5)+P2+P3)/2.5
1010 SC=INT(SC+.5):SU=SU+SC
1020 '
1030 ' fill A$ with all the remainder of the infomration
1040 A$=A$+STR$(SC)+" "+STR$(T)+"  "+D$(Z)+" "+M$(M)+D1$+STR$(Y)
1050 PRINT A$
1060 IF PR=1 THEN LPRINT A$
1070 A$=STRING$(52,32) ' clear A$ out for next time
1080 SC=0
1090 '
1100 ' increment the date for next day
1110 D2=FNDA(M)
1120 IF M=2 AND (NOT Y=1900) AND (Y-4*INT(Y/4)=0) THEN D2=D2+1
1130 D=D+1
```

```
1140 IF D>D2 THEN D=1:M=M+1
1150 IF M>12 THEN Y=Y+1:M=1
1160 GOSUB 1240 ' to find day of week for the new day
1170 GOSUB 1390:T1=D1-686707! ' update the moon phase
1180 '
1190 T=T+1 ' increment the days lived count
1200 CT=CT+1:IF CT<CP THEN 730 ELSE PRINT "Average index for this
     period is ";SU/CP:IF PR=1 THEN LPRINT"Average index for this
     period is ";SU/CP
1210 END
1220 '
1230 REM * find day of week subroutine
1240 IF M<3 THEN K=1 ELSE K=0
1250 L=Y-K
1260 Q=M+12*K
1270 P=L/100
1280 Z1=INT(P/4)
1290 Z2=INT(P)
1300 Z3=INT((5*L)/4)
1310 Z4=INT(13*(Q+1)/5)
1320 Z=Z4+Z3-Z2+Z1+D-1
1330 Z=Z-(7*INT(Z/7))+1
1340 D1$=STR$(D)
1350 IF LEN(D1$)=2 THEN D1$=" "+D1$
1360 RETURN
1370 '
1380 REM * total days calculator subroutine
1390 D1=(Y*365)+D:L=M:M1=M
1400 IF M<=2 THEN 1440
1410 L=(L*.4)+2.3
1420 L=INT(L)
1430 D1=D1-L:Y=Y+1
1440 M=((M*31)+(Y-1)/4)
1450 M=INT(M):D1=D1+M
1460 IF Y=1900 THEN D1=D1+1
1470 M=M1
1480 IF M>2 THEN Y=Y-1
1490 RETURN
1500 END 'of program
```

Change lines for Tandy I/III

```
Changes in bio.tr3 compared to bio.bas

Changed->140 CLEAR 2000
Changed->470 GOSUB 1390:T1=D1-686707 ' T1 holds days of moon phase
Changed->640 PRINT TAB(10);"-";TAB(26)"0";TAB(42);"+";TAB(51);"Index"
;TAB(57);"Age
Changed->660 PRINT STRING$(63,45)
Changed->1040 A$=A$+STR$(SC)+" "+STR$(T)
Changed->1170 GOSUB 1390:T1=D1-686707 ' update the moon phase
```

Biorhythm Chart for Joe Magarac

p is the 23-day physical cycle
e is the 28-day emotional cycle
i is the 33-day intellectual cycle
. is the phase of the moon (max + = 1st Qtr.)

```
          -                    O                    +        Index Age    Date
-------------------------------------------------------------------------------
 i . e      p                  : :                            2  21243  Sat Mar  1 1958
 i. e     p                    : :                            1  21244  Sun Mar  2 1958
 .ie  p                        : :                            1  21245  Mon Mar  3 1958
 . iep                         : :                            1  21246  Tue Mar  4 1958
   . p                         : :                            1  21247  Wed Mar  5 1958
     .pe                       : :                            2  21248  Thu Mar  6 1958
        .pei                   : :                            2  21249  Fri Mar  7 1958
            . pi               : :                            3  21250  Sat Mar  8 1958
                . p            : :                            4  21251  Sun Mar  9 1958
                  .:p:                                        0  21252  Mon Mar 10 1958
                  : : . p                                     6  21253  Tue Mar 11 1958
                  : :    .ip                                  7  21254  Wed Mar 12 1958
                  : :       .iep                              8  21255  Thu Mar 13 1958
                  : :           .p                            9  21256  Fri Mar 14 1958
                  : :             .                           9  21257  Sat Mar 15 1958
                  : :               pe.                       9  21258  Sun Mar 16 1958
                  : :                 p  ei.                  9  21259  Mon Mar 17 1958
                  : :                   p   e  .              9  21260  Tue Mar 18 1958
                  : :                     p    e  .i          9  21261  Wed Mar 19 1958
                  : :                  p      e  . i          8  21262  Thu Mar 20 1958
                  : : p                  e   . i              7  21263  Fri Mar 21 1958
                p : :                    e  . i               5  21264  Sat Mar 22 1958
              p   : :   e    .   i                           6  21265  Sun Mar 23 1958
           p      :e:    .   i                               3  21266  Mon Mar 24 1958
         p      e  : .    i                                  4  21267  Tue Mar 25 1958
       p      e    . : :i                                    3  21268  Wed Mar 26 1958
      p   e    .    i: :                                     1  21269  Thu Mar 27 1958
      p e   .    i   : :                                     2  21270  Fri Mar 28 1958
      e p.     i     : :                                     2  21271  Sat Mar 29 1958
     e.   p          : :                                     2  21272  Sun Mar 30 1958
   .e   i    p       : :                                     2  21273  Mon Mar 31 1958
Average index for this period is  4.419355
```

Joe Magarac (our old buddy) born on 1 January 1900 was 58 years, 2 months and 7 days old on the 10th of March 1958.

Biorhythm Chart for Joe Magarac

p is the 23-day physical cycle
e is the 28-day emotional cycle
i is the 33-day intellectual cycle
. is the phase of the moon (max + = lst Qtr.)

```
          -                   0                   +          Index  Age    Date
-------------------------------------------------------------------------------
          i                . :e:                 p            4  16982  Mon Jul  1 1946
        i                .  e  : :                 p          5  16983  Tue Jul  2 1946
      i                .  e     : :                 p        4  16984  Wed Jul  3 1946
    i                .  e        : :                 p      3  16985  Thu Jul  4 1946
  i                .  e           p : :                     1  16986  Fri Jul  5 1946
  i .  e                 p         : :                     2  16987  Sat Jul  6 1946
  .i e                     p       : :                     2  16988  Sun Jul  7 1946
  . e                        p     : :                     1  16989  Mon Jul  8 1946
    . eip                        : :                       1  16990  Tue Jul  9 1946
      .p  i                      : :                       1  16991  Wed Jul 10 1946
       p. e   i                  : :                       2  16992  Thu Jul 11 1946
         p  .e    i              : :                       2  16993  Fri Jul 12 1946
            p   .e    i           : :                     3  16994  Sat Jul 13 1946
                p   .e            :i:                     3  16995  Sun Jul 14 1946
                    p   .e        :.:  i                  3  16996  Mon Jul 15 1946
                        :p:   .e    i                     4  16997  Tue Jul 16 1946
                        : :    p   .e  i                  7  16998  Wed Jul 17 1946
                        : :        p   .  i               8  16999  Thu Jul 18 1946
                        : :            p .  i             9  17000  Fri Jul 19 1946
                        : :               p e.i           9  17001  Sat Jul 20 1946
                        : :                  pe .         9  17002  Sun Jul 21 1946
                        : :                  p e .       10  17003  Mon Jul 22 1946
                        : :                 p e  .       9  17004  Tue Jul 23 1946
                        : :               p   e  .       9  17005  Wed Jul 24 1946
                        : :             p   e    .       8  17006  Thu Jul 25 1946
                        : :          p     e   .          8  17007  Fri Jul 26 1946
                        : : p          e   .i             7  17008  Sat Jul 27 1946
                      p : :    e     .i                   4  17009  Sun Jul 28 1946
                  p       :e:    . i                     3  17010  Mon Jul 29 1946
                p     e   :.:i                           4  17011  Tue Jul 30 1946
              p     e     .  i: :                         2  17012  Wed Jul 31 1946
```

Average index for this period is 4.741936

The three cycles are all high together on the 22nd of July 1946 for our friend Joe, a very rare occurrence.

ORDER FORM

Issues

- ☐ Renew Subscription $24.95
- ☐ NEW Subscription (starts with Nov/Dec 1987 issue)..... $24.95
- ☐ All 1st Year CodeWorks issues (Issue 1 through Issue 7) $24.95
- ☐ All 2nd year CodeWorks issues (Issue 8 through Issue 13) ... $24.95

☐ GIFT Subscription

Please give both your name and the name and address of the person who will receive the gift.

Clip or photocopy and mail to **CodeWorks,**
3838 S. Warner St.
Tacoma, WA 98409

Computer type:

Comments:

Diskettes

- ☐ 1st year programs on disk (specify type below) $20.00
- ☐ 2nd year programs on disk (specify type below) $20.00

- ☐ PC/MS-DOS 40 track DSDD
- ☐ CP/M 5 1/4 inch (specify format and computer type here _____

_____

- ☐ TRSDOS Model I 35 track SSSD
- ☐ TRSDOS Model III 40 track SSDD
- ☐ TRSDOS Model IV 40 track SSDD

588

HOW?

- ☐ Check/MO enclosed
- ☐ Bill me later
- ☐ Charge to VISA/MC_____ *Exp* _____

TO: ***(Please print clearly)***

Name _____

Address _____

City _____ *State*_____ *ZIP*_____

Charge card orders may be called in (206) 475-2219 9 am to 4 pm weekdays, Pacific time.

Index Update

Additions to CWINDEX.DAT

Misc, program, Arcsin.bas, issue 16, page 3
Notes, clear screen on H-89, issue 16, page 3
Notes, right-set numbers in string form, issue 16, page 4
Ranprint.bas, reference, issue 16, page 4
Misc, article, Compiled BASIC, issue 16, page 7
Pump.bas, main program, issue 16, page 9, simulating a hydraulic ram pump
Broker.bas, main program, issue 16, page 14, an investment simulation program
Lstat.bas, main program, issue 16, page 25, a lotto statistics program
Random files, issue 16, page 32, a new indexing program for the Randemo series
Ranidx.bas, main program, issue 16, page 34, part 1 of improved sorting for Randemo
Notes, a program to show color, issue 16, page 37
Notes, where to put DATA statements, issue 16, page 37
Notes, degrees and radians, issue 16, page 37
Notes, rounding with the TAB function, issue 16 page 37
Notes, input without the question mark, issue 16, page 38
Notes, using TIME$ to name your files, issue 16, page 38
Ecal.bas, correction to original listing, issue 16, page 38
Notes, the importance of backups, issue 16, page 38
Download, what's happening there, issue 16, page 40

If you are using Qkey.Bas to keep a running index of CodeWorks articles and notes, these are the changes to bring that index up to date through the last issue.

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
US Postage
PAID
Permit No. 774
Tacoma, WA