

CODEWORKS

Issue 18

Jul/Aug 1988

CONTENTS

| | | |
|------------------------|-------|----|
| <i>Editor's Notes</i> | ----- | 2 |
| <i>Forum</i> | ----- | 3 |
| <i>Beginning BASIC</i> | ----- | 7 |
| <i>Mediator.Bas</i> | ----- | 9 |
| <i>Outline.Bas</i> | ----- | 15 |
| <i>Random Files</i> | ----- | 23 |
| <i>Computing Notes</i> | ----- | 29 |
| <i>Conversions</i> | ----- | 30 |
| <i>Hard Disks</i> | ----- | 37 |
| <i>Order Form</i> | ----- | 39 |
| <i>Index Update</i> | ----- | 40 |

CODEWORKS

Issue 18

Jul/Aug 1988

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

(c) 1988 80-Northwest Publishing Inc. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, the publisher assumes no responsibility for errors or omissions. Unless otherwise noted, all programs presented in this publication are placed in public domain. Please address all correspondence to:

CodeWorks, 3838 South Warner St.
Tacoma, Washington 98409

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

The CodeWorks download operates around the clock (usually) and has the following protocol: 300/1200 baud, 8 bits, no parity and one stop bit.

Authors: We constantly seek material from contributors. Send your material and allow 4 to 6 weeks for editorial review. You may send IBM 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. VISA and Master Card orders are accepted by mail or phone. Charge card orders may also be left via our on-line download system.

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 free sample copy.

Editor's Notes

We finally did it. This entire issue was produced using PageMaker and a laser printer. It was scary at times. You push a little here and things pop out there. And it took a whole lot longer to do than it did manually.

Of course, that's because we were not as familiar with desktop publishing as we were the manual method. Supposedly, by the third issue, we should be doing it faster (and, I hope, better).

When it was all done, I had the feeling that it just wasn't as "tight" as we are accustomed to. There seem to be little splotches of white space peeking out here and there that we couldn't seem to control as well as we did manually. And making things fit and come out right is what it's all about in design and paste-up.

But why, you may ask, are we even bothering with desktop publishing? There are several good reasons. First of all, it is the state of the art thing to do. If you don't keep up you get lost in the dust. For another thing, our conventional typesetting gear was getting on in years, and the longer we wait, the less it is worth to anyone wanting to buy it. But that's not all. Something unexpected popped up while all this was going on. We suddenly realized that with desktop publishing we were no longer tied to a building with a composition room and a darkroom with plumbing and all that. With desktop publishing, you could work off of your kitchen table if you had to.

Well, that gave us ideas. Why not forsake the high rent district and move into something a little more economical. Desktop publishing will let us do that, and we are looking around now for that new place.

We hate to think of changing addresses, but heck, no matter how good or bad a business is going it's always smart to economize. Not only that, but the sale of our old typesetting equipment and darkroom will more than pay for the laser printer we need.

It's that time of year again when we have to sit down and decide if CodeWorks is going to shoot for another year. Do we just finish the current year and drop it or do we solicit your renewals and go for it again?

By now you must all be aware that we aren't the big publisher from the West. Actually, we are pretty small potatoes, but are having a good time trying to keep an interest in BASIC programming alive and well. We don't have to fight off subscribers with sticks. On the contrary, it's getting tougher and tougher to find new lists to promote.

We have the material and the dedicated people to help put it all together. The big question is whether subscription renewals and diskette sales will support the operation for another year. Yes, we look ahead for a whole year because we don't exactly cherish the thought of leaving you all high and dry in the middle of a subscription year. It's just not the proper thing to do.

What this says is that it all depends on you, the readers. We are ready, willing and able. If you will renew promptly (and maybe buy a disk or two of programs) we'll sneak through yet another year of CodeWorks programs and computer enjoyment for you.

We really want to. We hope you do too.

Irv

Forum

An Open Forum for Questions and Comments

I have subscribed to CodeWorks since it started and I am well pleased with its contents. In your Issue 17 you described a program, "Genealogy on Display" and I thought that it was just what I wanted, since I have recently become very interested in detailing my ancestors.

I wrote to the author of the program at the address given and he wrote back saying that his release is version 5.0 and it will only run on (MS DOS) compatibles. Quoting from his letter, "although they are BASIC programs, they make extensive use of the file structure of the IBM PC DOS, together with extensive use of the screen-positioning capabilities of PC BASIC. Because of the above, it would be a major undertaking to modify them to run on your Model IV (Tandy)."

Do you know of another similar program?

Walter Evans, Jr
Waco, TX

Read on.

I have just received Issue 17 of CodeWorks and as always, I think it is great. Several times I have been going to drop you a line but never found the time. What you said in the first two letters (about genealogy) did turn me on.

...I'm doing the Long family genealogy. My daughter got me started and I have four cousins who are involved with the research. First I got all the shareware genealogy programs and evaluated them. Genealogy on Display is a good program, but as you say, it will hold only 500 records.

The best program that I found was Family History System, by Phillip Brown (834 Bahama Drive, Tallahassee, Florida 32301). As you say, it is a large system of programs. In fact the programs are so large that they use two disks and you do have to switch the disks when you need some of the reports. The limit on the number of records is 9,999 names in 99 generations. That will take care of most families. With two floppy drives you are limited to storing about 1,000 names, depending on the information you have to save. You can keep a record of everyone's address. There are files to store marriage, education, occupation, military and health records. I now have 817 name records in the file. Mr. Brown has been very helpful when I have needed any help and always answers my letters promptly...

Seymour E. Long
Margate, FL

The system of programs you refer to are, again, for MS DOS and compatible computers. But there is still hope for early Tandy users, read on.

In Issue 17 of your magazine there was some discussion of genealogy programs. A great one can be had for \$10.00 on shareware basis from Arthur C. Hurlburt. This program is everything you claimed for Genealogy on Display and more. It is random access and has no limit on the number of names except for disk space. It runs only on the (Tandy) Model III with TRSDOS 1.3, but it will keep your Model III from becoming obsolete if you are into family histories.

I would suggest you invest the ten dollars by sending to the following address with a request for Version 1.1 of CLAN: Arthur C. Hurlburt, 1919 N. Clark, Davenport, Iowa 52804. I have 465 family names in my file with plenty of room to spare and the author claims to have over 700 with space to spare.

Lawrence J. Carley
Mt. Morris, MI

Since Tandy Model IV's will also work in Model III mode, this ought to be an answer for many of you. We are quite surprised at the interest shown in genealogy, and welcome any other information we can pass along for the general interest of all.

I was about to put the listing for Bio.Bas into an envelope and send it to you when I decided to have one more look, and sure enough, I found the error of my ways. I had a double bracket (parentheses?) in line 160 and that was the culprit. What gets me is that I read the line to you over the phone and I still didn't see it! My wife keeps telling me to have my eyes checked and I guess she's right. Both Bio.Bas and Etax88.Bas are running fine now.

Ray Bowers
Las Vegas, NV

Murphy's Law runs rampant! Didn't you know that after looking for an error for days some un-initiated person will walk up, point right to the problem, and ask, "What's that for?"

...I have no idea how to get my old CP/M programs

to my MS DOS 3.5 inch disks and no one I have asked has been able to tell me, including the Tandy customer service in Fort Worth.

**Alton Munro
Lufkin, TX**

If both machines have RS-232 ports, you can direct connect them. Then on the CP/M machine, you can use PIP to redirect the file to the RS-232 port and on the MS DOS machine you can use any terminal program to accept them and store them on the 3.5 inch disks. You will need to make minor changes to the BASIC programs you port over this way, but it sure beats typing them all in by hand. Also, you will probably need to save the programs on the CP/M machine in ASCII first before you try to move them over to MS DOS.

I congratulate you on your magazine. It is expensive, but it is a great learning tool. I have done a fair amount of BASIC programming, but it has been to meet a personal need; I can see that I use only the simpler routines and forms of code. There is a lot of meat in your publication; doubtless, I will ultimately feel compelled to get the back issues - when I understand all that's in the issues I have.

You seem sincere in encouraging questions. I need some help in developing a program to evaluate the performance of a stock portfolio over a given period of time. It is, of course, very valuable to know how you are doing in comparison to various market indices, mutual funds, etc. If one has a sufficient number of different stocks to be adequately diversified, a computer program is almost essential to handle the math...

**William V. Victor
Northridge, CA**

We too, are interested in such a program and have been gathering information on it for some time. When we get it all together we'll publish it, of course. We do have a mutual fund tracking program almost ready to go, but are having fits trying to make it work on all models. Aside from that, it's over 600 lines long and would take the best part of an issue to list. We'll keep working on it.

...Like many other of your subscribers, I enjoy learning to program in BASIC. Recently I undertook to write a billing and accounts receivable program for our local rural water district. Being slightly more advance than a novice I found many routines in CodeWorks that helped in making my efforts a success. Thanks for a well-written, helpful publication.

...Recently I purchased a Tandy 1400LT to use in

connection with genealogical research in the archives of the several states in which I am doing research. To make the 1400LT compatible with my (Tandy) 1000SX, I attempted to use a 5 1/4 inch drive which had been removed from a Tandy 1000SX as an external drive for the 1400LT. I connected the 5 1/4 inch drive to the 1400LT with a standard IBM cable only to find that the wiring configuration is different on the 1400LT. The local Radio Shack dealer checked with Tandy who informed him that Tandy had not released the wiring configuration for an external drive and has not placed an external drive for the 1400LT on the market.

Can you provide me with the needed connection information or a source from which I can obtain the needed information? Any help would be appreciated.

**J. Theron Woodward, Jr.
409 Tombfield Road
Camden, SC 29020**

No, we can't. But anyone having this information can contact you directly since we have included your complete address.

I am encountering a problem in running Bio.Bas on my Model III. I get the statement "Undefined user function in 1110."

**W. P. Frakes
Cuyahoga Falls, OH**

The problem is the CLEAR statement in line 190. For the Model III (and any other machine that needs to clear space) remove line 190 and put in a new line 140 CLEAR 2000. Yes, I did it again! CLEAR seems to be my nemesis. That's probably the umpteenth time I've done that and still can't learn the simple lesson that CLEAR clears all variables and defined functions.

Issue 17, page 37. 'scuse please... at bottom of page is statement... "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."

Is this new computer math? For one like myself, who counts on fingers, there does seem to be a slight proof-reading irregularity - the same thing I encounter when I let a computer do my thinking for me..

I like the idea of a MS DOS booklet... keep up the excellent work.

**Joseph A. Dickerson
Baltimore, MD**

You are right. We were wrong. Our figures lied about Joe's age. And we should be able to announce the DOS booklet in this issue.

...I wrote you some time ago about the availability of a (Tandy) Model I upper/lower case kit. There is a guy in your own backyard that has them:

Electronic Closet
Tim Worcester
8187 Blakely Court West
Bainbridge Island, WA 98110

One of your other subscribers found him for me. Hope this helps and thanks for the great programs!

Bob Salisbury
El Cerrito, CA

I have just bought a TRS-80 Model III. I was buying the 80-Micro magazines, and they quit on me! So, I have signed up as a subscriber to your magazine, and have also picked up the last year's issues. I didn't know what to expect, as far as the magazine content might have been, because I've seen quite a few where you pay much for so little information, and mostly advertising.

And as it turned out, I like it! And I'm going to see about getting the rest of the back issues that I missed, since I'm so close to the beginning anyway. There is only one question. Is there a way to program around commands like:

CMD "J",X\$ which changes a given date to a Julian date.

CMD "B","OFF" which turns off the break key.

POKE 16916,X which protects a given number of lines on the video screen from scrolling.

As far as the break key goes, I guess that's just optional, but my program needs the scroll protect and the CMD "J" command...

Pat Chong
Las Cruces, NM

We have no idea of how to change the CMD "J" and Julian date thing. If we ever find a routine to do it we will publish it.

As you said, the break key is just a convenience and need not be bothered with too much.

The scroll protect can easily be accomplished if you use our general purpose locate/print@ subroutine. Also see Beginning BASIC in Issue 17. It talks a lot about cursor positioning.

And, in a follow up letter from Mr. Chong, he gave us the following bit of code to change a date like 03/01/88 to a Julian date:

```
10 A$="03/01/88" ' Sample date
100B=0:M=VAL(LEFT$(A$,2)):DATA
```

```
31,28,31,30,31,30,31,31,30,31,30,31
110 IF M>1 THEN FOR X=1 TO (M-1):READ
A:B=B+A:NEXT X
120 B=B+VAL(MID$(A$,4,2))
130 Y=VAL(RIGHT$(A$,2)):IF B>59 AND INT(Y/4)=Y/4 THEN B=B+1
```

Poker still cheats! I speak of Poker7, which I ordered from you on disk when I got my new Packard Bell computer. I have not tinkered with the program, but run it as is on my MS DOS, GW BASIC machine.

I have caught it cheating twice. The first time everyone passed so the evidence was lost. But last night it ventured forth again, and this time the foul deed was manifest... The program refused to recognize my pair of Kings as openers... I would appreciate your thoughts on this latest skullduggery.

Also, I would like to do some experimenting with the bluff factor in Poker7, but am unable to trace where in the code the bluff is set. I am under the impression the BL is the variable. It seems, too, that the game might be made stronger by adding a subroutine that utilizes types of bluffs other than the "one pair, pat hand" bluff it uses now. For instance, after missing a possible straight or flush, it might, on an average of one time in thirty odd tries (approx.), bet as if it caught the desired card even though it had a busted hand.

Thanks for your attention. My best to you and Code-Works.

Arthur Melanson
Audubon, NJ

In a program the size and complexity of Poker, it is sometimes quite difficult to trace isolated incidents like not being able to open when you have legal openers. What I'm saying is that I can't find a logical reason for what happened, even though it has happened to me too, on occasion.

Reference the bluffing: BL is the bluff variable. It first appears in line 2010 and comes up at random when two high cards are held. Then, to make the bluffer bet like a mad man, in line 4130 and 4230, BL figures into the raises. Notice also that HC (High Cards) figure in too, in lines 4110 and 4220. So you see that the bluff has to start with the hand determination and carry over to the betting rounds, and that the two are tied together. Knowing this, you should be able to work your own bluff into a busted flush or straight, but remember that in that case, you would also have to keep the hand from showing a fold and keep that hand in the game.

I would like to cast a vote in favor of games. While

not much of a player of arcade type games, I do enjoy the multi-player games such as Network (Issue 6). My favorite for years has been Santa Paravia, typed in from the December 1978 Softside magazine...

Clifton N. Duval
Star Lake, NY

Santa Paravia was a good game, written by our old friend, George Blank, who was once on the staff at Softside and was later on the editorial staff of Creative Computing magazine. We would really like to see a good game along those lines; one that had a realistic approach to both economics and ecology, and one you could learn something from. But that's a pretty tall order, since even our best economists don't seem to have economics under control yet. But the idea is simmering, ever so slightly and quietly, on our back burner.

When I first looked at the mail for this issue I thought there wasn't too much there. You proved me wrong again! Thank you all for the thought-provoking questions and answers. Enjoy the good ole' summertime, and we'll see you again around the start of football season. -Irv



I learned one thing, there's no such thing as a little error in a billion dollar corporation.

We have committed to another year of CodeWorks!

Please help out by renewing early so that we can set some sort of operating budget.

Order our program diskettes. The first two year's disks are still available; this year's disk will be ready about the 1st of September.

Check out our new MS DOS book on page 39.

Tell your computing friends about CodeWorks.

We have a great year planned. Stay tuned.

Beginning BASIC

A Look at Defined Functions

BASIC has many built-in functions. There is the MOD function, the INT function, the MID\$ function and many others. These are all called intrinsic functions. They are like little subroutines, and when called will return a value which depends upon what the function was designed to do.

BASIC also provides for user defined functions. With these, you can define any kind of function you wish and it then becomes like the intrinsic functions in that you can supply it with a variable value and it will return whatever your function was designed to deliver. Sounds almost like a subroutine, doesn't it? In a way, it is like a subroutine, and in fact, you could make a subroutine out of any user defined function. However, with the user defined function you don't need to use the GOSUB command or the RETURN either, for that matter. You can simply treat the defined function as a variable. Let's take a little example.

Suppose you wanted to design a function that would take a first name and last name and print them together with an appropriate space between them. Here is one way to do it:

```
10 DEF FNC$(A$,B$)=A$+" "+B$
70 INPUT "What is your first name";F$
80 INPUT "What is your last name ";L$
90 PRINT "Your full name is ";FNC$(F$,L$)
```

With this little routine, we can talk a lot about defined functions in general, and this one in particular. To begin with, line 10 defines the function (in this case, function C\$(A\$,B\$)). A defined function starts with the function name and an argument set equal to an expression of the definition. In our case, we said that the function name is C\$, the argument is (A\$,B\$) and the definition of the expression is A\$+" "+B\$. Now when you call this function and give two string variables (any two string variables!) to it, it will return the first string and the last string separated by a space. Notice that even though our defined function calls out A\$ and B\$, we can feed it F\$ and L\$ and it works. Also, just because we have used A\$ and B\$ in the defined function does not mean that these variables are "used up." In no way do these variables conflict with variables of the same name elsewhere in

the program. Further down in your program, for example, you could have A\$="CAT" if you liked and it wouldn't affect the defined function in any way.

Defined functions will work not only with string variables, but with any legal variable in BASIC. They must be contained in one program line (not one screen line!) of less than 255 characters, and there can be no colons used to separate statements in that line. However, this restriction can be overcome easily because defined functions can be nested, or once defined, one defined function can become a part of another defined function as we will see shortly.

We have already seen that variables used in defined functions are "local" to the defined function and do not affect the rest of program variables. We have also seen that the specific variable we used in defining the function need not be used when calling the function. For this reason, these variables (A\$ and B\$ in our case) are called dummy variables. They just indicate a place in the function where a variable (any variable) can go. It stands to reason, of course, that if your function uses string variables you must call it using string variables.

In our example above, we could have changed line 90 to read: NAME\$=FNC\$(F\$,L\$) and then NAME\$ would have contained the full name. No GOSUB and no RETURN, which brings up the question of when to use a defined function. The example we have just shown is a good one. There are many others. In our Poker program a couple of years back we made extensive use of both the INT and the MOD functions to strip off the suit and value of the cards from a three-digit number. These were used very many times in the program in lines that were already crowded with information. So we defined FNM(X)=X MOD 100 and FNI(X)=INT(X/100) and then called FNM(M(I,J)) or FNI(M(I,J)) whenever we needed either suit or value. The M(I,J) array contained the three-digit card value in question. It saved a lot of coding, it made it easier to see and follow and most of all, it shortened some of those long lines.

You cannot use a verb command in a defined function, i.e., no PRINT, GOTO or anything like that. Nor can you use IF...THEN either. But, you can do some powerful logic operations in a defined function. Here is

an interesting one: Lumber 2 by 4's are sold in even two foot lengths. That is, you cannot buy a 7 foot 2 by 4, you must buy an 8 footer and cut off one foot. In a program recently, we had need to find the next greater two-foot length, where the length was given originally in inches. So we used this defined function:

```
10 DEF FN(M)=M MOD 24<>0
```

and then later in the program, where H1 was the length in inches, we used this line to get the inches up to the next two-foot length:

```
90 FOR I=1 TO 25:IF FNM(H1) THEN  
H1=H1+1:NEXT I
```

Now when H1 (in inches) was not evenly divisible by 24 we kept adding one to H1 until it was evenly divisible. You can read the second statement in line 90 like this: if it is true that H1 is NOT evenly divisible by 24 then add one to it and try again until it is. There may be a more clever way to have done this, but this one works. This is another case where the length of the lumber had to be determined many times in the program and the defined function was an efficient way to do it. After having gone through the function, H1 was always 2, 4, 6, 8, 10, 12, etc. feet long when divided by 12. It couldn't be anything but.

What about errors? Do you remember the DATA statements and the code that reads them? If there is an error in the DATA statement itself, BASIC will tell you that the error occurred in the line that read the line that was actually in error. It's that way with defined functions too. If you have a syntax error in the line that defines the function, then the error line will show as the line that called the function, not the line containing the defined function. And while we are at it, we may as well add that the line defining the function must have been read at least once by BASIC before it can be called. For this reason, you generally put your defined function near the start of the program; somewhere in the initialization phase of the program. And again, while we are there, we can add further that a defined function can be re-defined later in the same program. The last function that BASIC encountered will be the operative one, assuming that there are two or more of them with the same name.

In the last issue (Issue 17, Bio.Bas, page 34) we used a very intricate trio of defined functions to determine how many days there were in any given month. The functions were:

```
DEF FNE(M)=(M-2*INT(M/2)=0)  
DEF FNO(M)=NOT FNE(M)  
DEF FNDA(M)=30+(M<8)*FNO(M)+  
(M>7)*FNE(M)+2*(M=2)
```

The first function is to determine if a number is even. The second says that the number is odd if it is NOT even (since odd and even are mutually exclusive, we can say that.) The third function determines the number of days in any month (given the number of the month and excluding leap years.) The even numbered months from January through July are 30 days long, except for February, and the odd numbered months from August to December are 30 days long. Keep in mind that logic functions always return a -1 when true and a zero when false. With that in mind, you can go through the third defined function, above, and tell that a month has either 30, 31 or 28 days.

That's quite a bit of logic to go through, but once defined all you need do is give M a value from 1 to 12 and say PRINT FNDA(M) and you have the number of days in that month.

The other nice thing about defined functions is that once you have worked up some neat ones you will find yourself stripping them out of older programs and using them again in your current programs. In fact, it's a good idea to start keeping a library of such functions. That way, you won't have to keep re-inventing the same wheel. •

**If you are moving
from CP/M
or
TRSDOS to MS DOS
you can find everything about
starting out in MS DOS in
our new booklet,
"Starting with MS DOS"
and it's only \$7 postpaid
see page 39 for
ordering information.**

Mediator.Bas

Let Your Computer Help Settle Disputes

David Leithauser, New Smyrna Beach, Florida. Although disputes are not the most pleasant of subjects, they do exist and must be dealt with. Most disputes are also rather emotional affairs. A computer program, like Mediator.Bas, can assist in bringing both sides in a dispute to some reasonable settlement.

Mediator.Bas is a simple computer program for mediating disputes between two parties or groups. It is written in BASIC and conversion to various machines are given at the end of the program listing.

Using Mediator

When the program is run, it will first ask you for the name of party #1 and party #2, the two parties in the dispute. The names of the parties could be anything, such as TOM and SUE, Management and Labor, or USA and USSR. Once you have input the names of the parties, the program will refer to the parties by name for the remainder of the program.

The program next asks how many issues are to be resolved. Input any number. Mediator then asks how many of these are of the type that a numerical compromise can be achieved. These are issues such as how much of a pay raise employees should get, or how much money the defendant should pay the plaintiff in a civil suit. In some cases, Mediator may split the difference (not necessarily evenly) in numerical issues to achieve a fair settlement.

Next, Mediator will ask for a description of each non-numerical issue. Input a brief description of each issue in the form of a question. Typical examples might be "Who gets custody of the child" in a divorce case, or "Do employees get a paid vacation on their birthday" in a labor negotiation. Mediator will then ask for a similar description of each numerical issue. In numerical questions, it is important that the description be phrased so that the question can be answered by a single number. For example, if you were dividing up some money

between Tom and Fred, you should not phrase the question as "How much money should Tom and Fred each get," because this may involve a different number for each one. Instead, the question should be phrased as "How much money should Tom get." The amount of money that Fred gets would then be the remainder of the money.

Next, Mediator will ask for the position of the first party on each issue. For example, in a management-labor dispute, it might ask "For management, describe your position on issue of 'Do employees get a paid vacation on their birthdays.'" Management would probably input "NO." Mediator might then ask, "on the issue of how big a pay raise do the employees get, what is your desired value?" Management would probably answer zero, or even a negative number to indicate a pay cut. Mediator would then ask the same questions for the second party.

Once both sides have input their positions on each issue, Mediator will say: "It is now time for (name of first party) to rate the importance of issues." It will then provide a list of the issues and the positions of the two sides on each issue. It will also ask if you want a hard copy of this list. This will allow you to look over the issues at your leisure. If you respond by pressing the "Y" key and the return key, a list will be output by your printer.

Mediator will then ask which of the issues is most important to the first party. The representative of the first party should input the number of the most important issue from the list. The issues selected is given a value of 10 on a scale of 1 to 10. Mediator then asks party

number 1 to rate each of the remaining issues in importance from 1 to 10, as compared to the importance of the issue chosen as most important. For example, if a certain issue is half as important to party 1 as the most important issue, that issue should be given a rating of 5. An issue that is just as important as the main issue can be given a rating of 10, and an issue of little importance can be given a rating of 1. The user should understand that Mediator evaluates how important each issue is to a party in relation to the importance of the other issues for that party. Therefore, it does not improve the bargaining position of someone to say that all issues rate a 10. Giving a particular issue a high rating automatically reduces the importance rating of the other issues for that party. Anyone who lies and says that all issues rate a 10 is decreasing their chances of getting what is really important to them.

After the first party has input the importance rating of each issue, the process will be repeated for the second party. I suggest that each party input their importance ratings in secret, to prevent the second party from trying to hedge their answers based on the answers of the first party.

Once the second party has input their importance ratings, Mediator will output its decision on each issue. It will also output a satisfaction index for each party. This index indicates how much of what it wanted each party got, weighted by how important each issue is rated by that party. In most cases, the satisfaction index of both parties will be over 50 percent.

The sample run shows a hypothetical divorce case between Tom and Sue. The four issues involved are who gets possession of the house, who gets custody of each of the two children (Fred and Mary), and how much alimony Tom pays Sue each month. The answers that are input by the user are in boldface type.

Basic Principles of Mediator

Mediator evaluates how important each issue is to each party, and gives each party what is most important to that party. This tends to result in a satisfaction index of over 50 percent for both parties, a desirable win-win situation.

If a particular non-numerical issue is of equal importance to both sides, a decision is made which will tend to balance the satisfaction indexes of the two parties. For example, if the indexes stand at 45 percent for party A and 60 percent for party B, and the remaining non-numerical issue is of equal importance to both parties, the decision is made in favor of party A. If a numerical

issue is of equal value to both sides, the number is distributed to balance the satisfaction indexes.

Advantages of Mediator

The first advantage of Mediator is that it forces the two sides to sit down and evaluate how important each issue really is to them. It actually forces them to assign a numerical value to each issue relative to each other issue, so they can improve their chances of getting what they really want. This tends to cut through all the bluster and posturing involved when people claim that issues are non-negotiable.

The second advantage of Mediator is that it provides a completely objective mediator. No one can accuse a computer of having any biases in the disagreement. The program makes its decision based entirely on what each side wants and how badly they want it, not on some external preconception of what is the "right" decision.

Disadvantages of Mediator

Mediator is not capable of generating any creative new solutions to the problem, the way a human mediator might. It merely takes the positions of the two sides and tries to find the most equitable way to divide its decisions between the two parties.

Another problem is that Mediator does not really understand the issues, and therefore its decisions may not always be reasonable or practical. One side (or both) could input a totally unreasonable position on some issues to force the other side to devote all its efforts to preventing that side from getting its way on that issue. For example, in a management-labor dispute, labor could input that it wants a raise of \$1,000,000 per week, to force management to give all its importance points to that issue to insure that labor does not win that point. It is therefore necessary that a human mediator be present to oversee the process, to make certain that both sides are inputting "good faith" positions on each issue.

In view of these problems, and the extreme simplicity of this program, Mediator should be viewed as a potential tool in the mediation process and an interesting demonstration of computer aided negotiation, not as something that is about to replace human mediators. It could also be an interesting starting point for a more advanced system, perhaps something that could be combined with an expert system program. •

Listing for MS DOS and Tandy IV

```
100 REM * Mediator.Bas * for CodeWorks by D. Leithauser *
110 'CLEAR 2000 ' only if your BASIC is prior to ver 5.0
120 CLS
130 PRINT "COMPUTER MEDIATOR":PRINT: VERSION 1.1":
    PRINT
140 DIM N$(2), SR(2), RT(2)
150 FOR X=1 TO 2
160   PRINT "NAME OF PARTY # ";X;
170   INPUT N$(X)
180 NEXT X
190 INPUT "NUMBER OF ISSUES TO BE RESOLVED ";N
200 INPUT "NUMBER OF THESE ON WHICH A NUMERICAL COMPROMISE CAN BE
    ACHIEVED ";NU
210 NN=N-NU:DIM NP$(N), S$(NN,2), S(NU,2), R(N,2), RV(N,2)
220 FOR X=1 TO NN
230   PRINT "DESCRIBE NON-NUMERICAL ISSUE # ";X
240   LINE INPUT NP$(X)
250 NEXT X
260 FOR X=1 TO NU
270   PRINT "DESCRIBE NUMERICAL ISSUE # ";X
280   LINE INPUT NP$(X+NN)
290 NEXT X
300 FOR Y=1 TO 2
310   CLS
320   PRINT "FOR ";N$(Y)
330   FOR X=1 TO NN
340     PRINT "DESCRIBE POSITION ON ISSUE OF":PRINT CHR$(34);
        NP$(X);CHR$(34)
350     LINE INPUT S$(X,Y)
360     IF Y=2 AND S$(X,1)=S$(X,2) THEN PRINT "ERROR! BOTH PARTIES
        APPEAR TO AGREE ON THIS ISSUE.":PRINT "SOMEONE MUST HAVE
        MISUNDERSTOOD THE INSTRUCTIONS.":PRINT "REREAD MANUAL AND
        START OVER.":END
370   NEXT X
380   FOR X=1 TO NU
390     PRINT "ON THE ISSUE OF ";CHR$(34);NP$(X+NN);CHR$(34)
400     INPUT "WHAT IS YOUR DESIRED VALUE ";S(X,Y)
410     IF Y=2 AND S(X,1)=S(X,2) THEN PRINT "ERROR! BOTH PARTIES
        APPEAR TO AGREE ON THIS ISSUE.":PRINT "SOMEONE MUST HAVE
        MISUNDERSTOOD THE INSTRUCTIONS.":PRINT "REREAD MANUAL AND
        START OVER.":END
420   NEXT X
430 NEXT Y
440 FOR Y=1 TO 2
450   CLS
460   PRINT "IT IS NOW TIME FOR ";N$(Y);" TO RATE THE IMPORTANCE
        OF ISSUES."
470   PRINT "THIS IS A LIST OF THE ISSUES AND THE POSITIONS OF
        EACH GROUP:"
```

NOTE:

0=zero

O=oh

```

480 FOR X=1 TO NN
490 PRINT X;" " ;NP$(X)
500 FOR Z=1 TO 2:PRINT N$(Z) ;":" ;S$(X,Z) :NEXT Z
510 NEXT X
520 FOR X=1 TO NU
530 PRINT X+NN;" " ;NP$(X+NN)
540 FOR Z=1 TO 2:PRINT N$(Z) ;":" ;S(X,Z) , :NEXT Z:PRINT
550 NEXT X
560 INPUT "DO YOU WANT HARD COPY OF THIS (Y/N) " ;H$
570 H$=LEFT$(H$,1) :IF H$<>"N" AND H$<>"n" AND H$<>"Y" AND
H$<>"y" THEN 560
580 IF H$="N" OR H$="n" THEN 680
590 LPRINT "THIS IS A LIST OF THE ISSUES AND THE POSITIONS OF
EACH GROUP:"
600 FOR X=1 TO NN
610 LPRINT X;" " ;NP$(X)
620 FOR Z=1 TO 2:LPRINT N$(Z) ;":" ;S$(X,Z) :NEXT Z
630 NEXT X
640 FOR X=1 TO NU
650 LPRINT X+NN;" " ;NP$(X+NN)
660 FOR Z=1 TO 2:LPRINT N$(Z) ;":" ;S(X,Z) :NEXT Z
670 NEXT X
680 PRINT "WHICH OF THESE ISSUES IS MOST IMPORTANT TO " ;N$(Y) ;
690 INPUT M:IF M<1 OR M>N THEN PRINT "INVALID ANSWER!":GOTO 690
700 R(M,Y)=10:RT(Y)=10
710 PRINT "THE ISSUE OF " ;CHR$(34) ;NP$(M) ;CHR$(34)
720 PRINT "NOW HAS A VALUE OF 10 ON A SCALE OF 1 TO 10."
730 PRINT "PLEASE RATE THE REST OF THE ISSUES ON A SCALE OF 1 TO
10 COMPARED TO"
740 PRINT CHR$(34) ;NP$(M) ;CHR$(34)
750 FOR X=1 TO N
760 IF X=M THEN 810
770 PRINT CHR$(34) ;NP$(X) ;CHR$(34)
780 INPUT "RATING " ;R(X,Y)
790 IF R(X,Y)<1 OR R(X,Y)>10 THEN PRINT "INVALID ANSWER!":GOTO
780
800 RT(Y)=RT(Y)+R(X,Y)
810 NEXT X
820 NEXT Y
830 FOR X=1 TO 2
840 FOR Y=1 TO N
850 RV(Y,X)=INT((R(Y,X)/RT(X)+.005)*100)
860 NEXT Y
870 NEXT X
880 CLS
890 PRINT "THE FOLLOWING IS THE DECISION ON EACH ISSUE:"
900 FOR X=1 TO NN
910 IF RV(X,1)=RV(X,2) THEN 950
920 PRINT NP$(X) ;":" ;
930 IF RV(X,1)>RV(X,2) THEN PRINT S$(X,1) :SR(1)=SR(1)+RV(X,1)
940 IF RV(X,1)<RV(X,2) THEN PRINT S$(X,2) :SR(2)=SR(2)+RV(X,2)
950 NEXT X
960 FOR X=1 TO NU

```

```

970 IF RV(X+NN,1)=RV(X+NN,2) THEN 1010
980 PRINT NP$(X+NN);":":
990 IF RV(X+NN,1)>RV(X+NN,2) THEN PRINT S(X,1):SR(1)=SR(1)+RV(X+
  NN,1)
1000 IF RV(X+NN,2)>RV(X+NN,1) THEN PRINT S(X,2):SR(2)=SR(2)+RV(X+
  NN,2)
1010 NEXT X
1020 FOR X=1 TO NN
1030 IF RV(X,1)<>RV(X,2) THEN 1090
1040 PRINT NP$(X);":":
1050 IF SR(1)>SR(2) THEN PRINT S$(X,2):SR(2)=SR(2)+RV(X,2):GOTO
  1090
1060 IF SR(2)>SR(1) THEN PRINT S$(X,1):SR(1)=SR(1)+RV(X,1):GOTO
  1090
1070 IF RND<.5 THEN PRINT S$(X,2):SR(2)=SR(2)+RV(X,2):GOTO 1090
1080 PRINT S$(X,1):SR(1)=SR(1)+RV(X,1)
1090 NEXT X
1100 FOR X=1 TO NU
1110 IF RV(X+NN,1)<>RV(X+NN,2) THEN 1200
1120 PRINT NP$(X+NN);":":
1130 T=SR(1)+RV(X+NN,1):IF T<=SR(2) THEN PRINT S(X,1):SR(1)=T:
  GOTO 1200
1140 T=SR(2)+RV(X+NN,2):IF T<=SR(1) THEN PRINT S(X,2):SR(2)=T:
  GOTO 1200
1150 VD=RV(X+NN,1)/ABS(S(X,1)-S(X,2)):SP=SGN(SR(1)+ABS(S(X,2)-
  S(X,1))*VD-SR(2))
1160 FOR V=S(X,1) TO S(X,2) STEP (S(X,2)-S(X,1))/128
1170 S1=SR(1)+ABS(S(X,2)-V)*VD:S2=SR(2)+ABS(S(X,1)-V)*VD
1180 IF SP<>SGN(S1-S2) THEN PRINT V:SR(1)=S1:SR(2)=S2:GOTO
  1200
1190 NEXT V
1200 NEXT X
1210 FOR X=1 TO 2
1220 PRINT "SATISFACTION INDEX FOR ";N$(X);"=";SR(X)"%"
1230 NEXT X

```

Change lines for Tandy I/III

```

Changed->100 REM * Mediator/Bas * for CodeWorks by D. Leithauser *
Changed->110 CLEAR 2000 ' only if your BASIC is prior to ver 5.0

```

Sample Run for Mediator.Bas

COMPUTER MEDIATOR
VERSION 1.1

Name of party #1? TOM
Name of party #2? SUE
Number of issues to be resolved? 4
Number of these on which a numerical compromise can be achieved ? 1

Describe non-numerical issue #1

WHO GETS THE HOUSE

Describe non-numerical issue #2

WHO GET CUSTODY OF FRED

Describe non-numerical issue #3

WHO GET CUSTODY OF MARY

Describe numerical issue #1

HOW MUCH ALIMONY DOES TOM PAY SUE PER MONTH

For Tom

Describe position on issue of

"Who gets the house"

TOM

Describe position on issue of

"Who get custody of Fred"

TOM

Describe position on issue of

"Who gets custody of Mary"

TOM

On the issue of "How much alimony does Tom pay Sue per month"

What is your desired value ? 0

For Sue

Describe position on issue of

"Who gets the house"

SUE

Describe position on issue of

"Who gets custody of Fred"

SUE

Describe position on issue of

"Who gets custody of Mary"

SUE

On the issue of "How much alimony does Tom pay Sue per month"

What is your desired value? 1000

It is now time for Tom to rate the importance of issues.

This is a list of the issues and the positions of each group:

1) Who gets the house

Tom:Tom

Sue:Sue

2) Who gets custody of Fred

Tom:Tom

Sue:Sue

3) Who gets custody of Mary

Tom:Tom

Sue:Sue

4) How much alimony does Tom pay Sue per month

Tom: 0 Sue: 1000

Do you want hardcopy of this (Y/N) ? N

Which of these issues is most important to Tom? 4

The issue of "How much alimony does Tom pay Sue per month"

now has a value of 10 on a scale of 1 to 10.

Please rate the rest of the issues on a scale of 1 to 10 compared to

"How much alimony does Tom pay Sue per month"

"Who gets the house"

Rating ? 2

"Who gets custody of Fred"

Rating ? 6

"Who gets custody of Mary"

Rating ? 4

It is now time for Sue to rate the importance of issues.

This is a list of the issues and the positions of each group:

1) Who gets the house

Tom:Tom

Sue:Sue

2) Who gets custody of Fred

Tom:Tom

Sue:Sue

3) Who gets custody of Mary

Tom:Tom

Sue:Sue

4) How much alimony does Tom pay Sue per month

Tom: 0 Sue: 1000

Do you want hardcopy of this (Y/N)? N

Which of these issues is most important to Sue ? 3

The issue of "Who gets custody of Mary"

now has a value of 10 on a scale of 1 to 10.

Please rate the rest of the issues on a scale of 1 to 10 compared to

"Who gets custody of Mary"

"Who gets the house"

Rating ? 6

"Who gets custody of Fred"

Rating ? 7

"How much alimony does Tom pay Sue per month"

Rating ? 3

The following is the decision on each issue:

Who gets the house: Sue

Who gets custody of Mary: Sue

How much alimony does Tom pay Sue per month: 0

Who gets custody of Fred: Tom

Satisfaction index for Tom = 72%

Satisfaction index for Sue = 61%

Outline.Bas

Part One of a Three-Part Outlining Program

Terry R. Dettmann, Associate Editor. Outliners and outlining programs are useful in organizing your thoughts before you begin to write. Commercial programs of this type are available in the \$100 price range. In this series, Terry will build an outlining program that will run in BASIC or can be compiled. Along the way, we will learn something about linked list techniques and what makes them work.

One of the most useful tools to writing and thinking in general is the outline processor. On the IBM PC and Macintosh, this type of program has become quite sophisticated. But outline processors can be expensive and they aren't compatible with all machines. Over the next several issues, we're going to put together an outline processor with all of the basic features (and you'll have the source code!).

The current plan for these articles is to cover the subject in three segments. The first segment, we're going to lay some theoretical foundations for the outline program by explaining a technique known as 'List Linking' which we'll need to build the outline processor. A demonstration program in this article will show the basic linking procedure.

The second article in the series will show the screen display handling and build the screen control portion of the outline program. The last article will add the two together to give us a final program which is more than either demonstration program.

When I first started on the program, I assumed one article would be enough. But the program needed more complexity than I could explain in one article. After that, I thought two were enough. In trying to write two that made the subject understandable, I've now added a third article to lead into the other two. Whew! I just hope I can keep it to three. If we get more questions between issues than can be answered in a single article, I'll add more material as necessary to make sure everyone understands it.

When we're done, you'll have an outline processor which will allow you to build outlines, print them, and change them as needed. The program uses some pretty sophisticated techniques, but we'll take them slowly enough to make them simple to understand. Let's start by building a very simple outline program which illustrates list linking as we're going to need it for the full scale outline processor.

List Linking

By now, you should be familiar with arrays in programming. Arrays are used for multiple pieces of information which can be organized by number, but what if what we want to deal with isn't arranged by number?

Keeping track of generalized data, large lists of information, can often be done when each piece of information is related in some simple way to the other pieces of information in the system. For example, if we have a list of names, we could arrange them in ascending or descending sorted order. There's no natural numeric associated with this process, it's purely alphabetic. The fact that the letter 'A' comes before 'Z' isn't a matter of numbers.

To create a list of information, we use what's known as a 'Linked List'. Each item in the list (called a NODE) is 'linked' to the next node by a POINTER which tells the program where to find the next one. Graphically, we could show this like this:

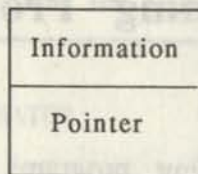


FIGURE 1

where the top part of the box is the information part of the node and the bottom part is the pointer. Let's say we're going to link together the pieces of information 'one', 'two', and 'three' in alphabetic order. Graphically, we might indicate this like this:

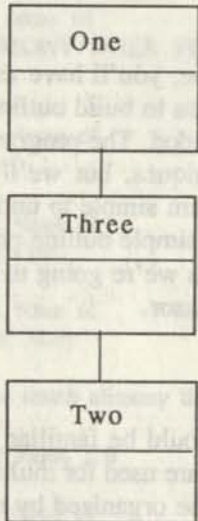


FIGURE 2

In this case, the 'one' node is linked to the 'three' node (the next one alphabetically) and the 'three' node is linked to the 'two' node. At the end of the list, we use a special marker to indicate the end of the list.

Conceptually, the picture is nice, but how does this help. We don't have linked lists in BASIC. Some languages like Pascal and C make implementing this type of structure pretty easy because they include special structures which can be used to create linked lists. BASIC doesn't though. BUT, we can make BASIC work as if it had them.

What if we entered the three pieces of information

'one', 'two', and 'three' in their numeric order. If the linked list is arranged in this order, then it would look like this:

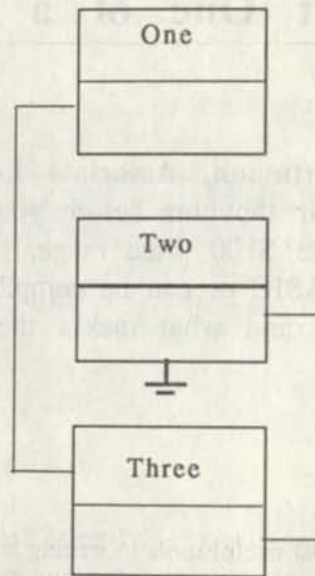


FIGURE 3

If we're entering them into a program, we could store them in an array (which I'll call LN\$ for lines). As we enter each line, we could store them in LN\$ at increasing array locations like this:

| Array | Information |
|-------|-------------|
| 1 | one |
| 2 | two |
| 3 | three |

FIGURE 4

If we wanted, we could run a sorting program on this and put them in alphabetic order. But we could also put them in order if we had a set of links which point from each item to the next in order. Let's define an array we'll call LK (for links). If LK(0) points to the smallest item alphabetically and then the corresponding LK array value points to the next, then our table would be:

| Array | Information | Links |
|-------|-------------|-------|
| 0 | | 1 |

FIGURE 5

| | | |
|---|-------|---|
| 1 | one | 3 |
| 2 | two | 0 |
| 3 | three | 2 |

In this case, LK(0) points to item 1 (the information 'one' which is the smallest in a sort). LK(1) points to array location 3 (item 'three' which is next in alphabetic order) and LK(3) points to array location 2 (item 'two', the last in alphabetic order). Notice that LK(2) is zero. This is used as the END OF LIST marker. When we reach it, we know there are no more items in the list. From a simple structure like this, we can build some very sophisticated software.

To illustrate the linked list in a more practical sense, I've included the sample program LINK.BAS in listing 1.

In our simple program, like the example we talked through above, we have arrays LN\$ to hold the lines (maximum of 20) and LK to hold the links. NX (which starts at 1) is the array location to put the next line when we read it from the keyboard.

In the main loop of the program (lines 200-290), we clear the screen, print the current list (subroutine 1000), and then prompt for an input line. An input line consisting of nothing but DONE (in caps) will end the program and cause it to print the list one more time (lines 400-420). If the input line is not DONE, then we'll call subroutine 1000 and add the line to the linked list.

To see how the linked list is built, we look at subroutine 1000. First, we start out by saving the next array location in variable J, then we put the new line in the next location of the array (LN\$(NX)) and set its link value to zero. We advance NX by one to get it ready for the next line.

If the value of LK(0) is zero, (remember, it points to the first actual information node), then we can assume nothing has been added yet and we can simply set LK(0) to point to the line we just added. If LK(0) is not zero, then things get more complicated. With something already in the list, we have to start with the first item in the list and compare it to the new item we want to add. To keep them in alphabetical order, we check them one at a time until we find one which belongs after the one we're looking for. We add the new one at that point.

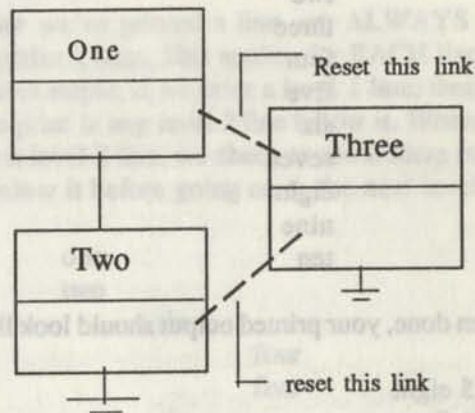


FIGURE 6

We start out by setting I to the array location of the 1st item in the list and K to array location of the pointer to that item (initially zero). Now we check, if the new item is less than the one we're checking, then we put it ahead in the list. If it's not smaller, then we move on to the next item and check for the end of the list (link value zero). If we're at the end, we simply add the new item to the end. If we're not already at the end, then we start again from our comparison.

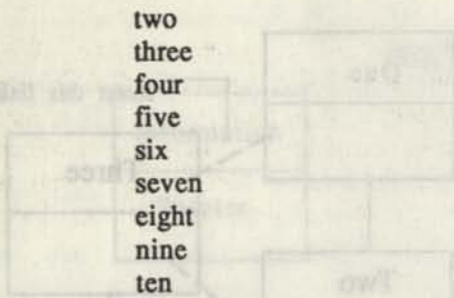
This method (or algorithm) is a pretty standard way to deal with inserting items in a linked list. There are several variations possible, but they are all basically the same.

The subroutine at line 2000 steps through the linked list one item at a time, in order, by following the links. This is called 'Traversing the List'.

We start the subroutine by checking to see if anything has been added to the list yet (line 2010). If not, there's nothing to print and we leave. If there IS something, we set I to point to the 1st item in the list and then print it. Next we let I point to the next item (I = LK(I)). If I isn't zero yet, we go back and print it.

This technique is powerful and can be quite fast for simple lists. Try running the program for the following series of entries:

one



When done, your printed output should look like this:

```

8 5 eight
5 4 five
4 9 four
9 1 nine
1 7 one
7 6 seven
6 10 six
10 3 ten
3 2 three
2 0 two

```

Since we entered each line in the list in alphabetical order, we can print them out in the same order by following the links. Try it on some other lists to see how it works. When you're satisfied that you understand this list linking concept, then read on to see how we can add additional links and create a more sophisticated data structure.

An Outline List

To create an outline, we have several levels of linked lists. At a given level in an outline, the items could be considered linked from one to the next in the order of the outline. We could also consider the first item under a given item to be linked to its parent. This sounds pretty complicated, but look at the following outline:

```

I Introduction
  A. Point 1
  B. Point 2
II Detail
  A. Point 3
  B. Point 4

```

If we used our graphic representation of the nodes, we could represent this outline like this:

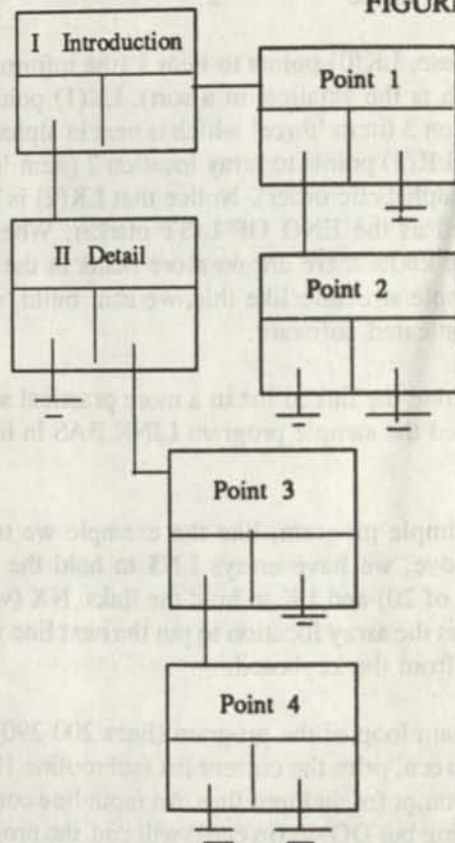


FIGURE 7

By adding a second pointer to the node, we can now go in two directions in linking, either to the next item at the same level in the outline or to the first item just below the current item. Traversing the list is now more complex since we'll have to be able to return to the previous level whenever we hit the end of any one level. Hitting the end of the highest level means we're done.

Starting with the first node of the outline, we could print all nodes by the following method:

1. print the node
2. print all nodes subordinate to the current node
3. go to the next node at the current level
4. if there is no next node, then return to the next higher node level

If you consider the whole procedure as 'print all nodes subordinate to the current node' starting with node zero, then this describes a RECURSIVE procedure (one which calls itself). Let's see how we put this together with lists linking to build a simple outliner in Listing #2, LIST.BAS.

Like our simple linked list program, this one starts out by declaring the line array LN\$ and then link array LK. Now however, the link array is declared as LK(100,1) so that there are two links (LK(I,0) and LK(I,1)). We've also declared the arrays LL (for the line level in the outline) and LV for the last line at a given level in the outline. We'll see what these contribute as we start working with the list.

Our main loop (lines 200-320) is basically the same as before:

1. print the list
2. wait for a command
3. if the line is a command, process it
4. otherwise add the line to the list
5. restart at step 1

To keep from having to work with more complexity than necessary to show how the list linking works, we've used command words UP and DOWN for moving up and down one level at a time within the outline. Subroutines 1000 and 1100 move the level up or down by one. Notice that the LV array keeps track of the current line number in the array when we move down, and then helps restore it to the current line when we move back up again.

Adding a line to the outline works pretty simply. We add it to the next open line in the array (CL = CL + 1 ... add one to the current line). If the level of the last line is not the same as the current level, we are linking a new level (line 1240), otherwise, we're linking on the current level (line 1220). The figure shows how the linking might look after five entries into LN\$ where we went DOWN after the second entry and UP after the fourth. As we've limited it (which we won't be able to do in the final program), this is simpler than the list linking we've talked about before. But now, subroutine 2000 to print the outline gets more complicated.

We start out as before at the beginning of the list. The variable VI is introduced to incorporate a variable indent for printing. Line 2025 prints a heading and then we start by printing the first line in line 2040. Now we

have to decide what to print next.

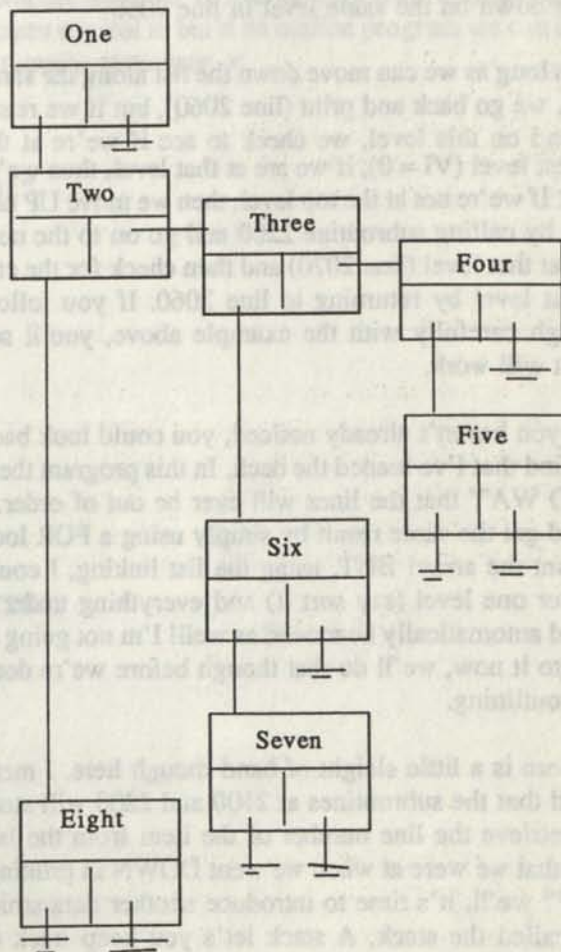
After we've printed a line, we ALWAYS print the lines under it next. This applies for EACH line we look at. For example, if we print a level 1 line, then the next line to print is any level 2 line below it. When we print the first level 2 line, we check to see if there is a level 3 line below it before going on to the next level 2 line.

```

one
two
      three
      four
      five
      six
      seven
eight
  
```

When we've linked this together, we'll get a picture like this:

FIGURE 8



To print this in the correct order, we'll follow the links as follows:

```
one -> two
two -> three
three -> four
four -> five
BACKUP ONE LEVEL to THREE
three -> six
six -> seven
BACKUP ONE LEVEL to TWO
two -> eight
```

To do this, first we check to see if the link to a lower level (LK(I,1)) is non-zero. If it is, we save the current location by calling subroutine 2100 and then move to the next level down and go back and print. We'll keep moving down to the lowest possible level as long as LK(I,1) is not zero. When it DOES reach zero, we'll move down on the same level in line 2050.

As long as we can move down the list along the same level, we go back and print (line 2060), but if we reach the end on this level, we check to see if we're at the highest level (VI = 0), if we are at that level, then we're done. If we're not at the top level, then we move UP one level by calling subroutine 2200 and go on to the next item at that level (line 2070) and then check for the end at that level by returning to line 2060. If you follow through carefully with the example above, you'll see that it will work.

If you haven't already noticed, you could look back and find that I've loaded the deck. In this program there is NO WAY that the lines will ever be out of order. I would get the same result by simply using a FOR loop to print the array! BUT, using the list linking, I could reorder one level (say sort it) and everything under it would automatically be moved as well! I'm not going to go into it now, we'll do that though before we're done with outlining.

There is a little sleight of hand though here. I mentioned that the subroutines at 2100 and 2200 will store and retrieve the line number of the item from the last level that we were at when we went DOWN in printing. HOW? we'll, it's time to introduce another data structure called the stack. A stack lets you keep track of

things where you want to get them back in the opposite to the order you put them in. In the example above, when I move down from array location 2 to 3, I put the location 2 on the stack (SP becomes 1, STK(1) = 2). Similarly, when I go from 3 to 4, I put location 3 on the stack (SP becomes 2, STK(2) = 3).

When I'm done with the lowest level after printing array location 5, I need to get the top of the stack (STK(SP) is now 3), this takes me back up one level to where I need to go. After I'm done printing at this level, I do it again (STK(SP) is 2) and get back to array location 2.

Visually, most people imagine a stack like a stack of dishes in a cafeteria. The last dish added to the stack is the first one taken off. This allows us to backtrack one level at a time as we need to. Let's illustrate with an example.

Again, we'll start the program and do the following steps:

1. type 'one'
2. type 'two'
3. type 'DOWN'
4. type 'three'
5. type 'four'
6. type 'DOWN'
7. type 'five'
8. type 'six'
9. type 'UP'
10. type 'seven'
11. type 'DOWN'
12. type 'eight'
13. type 'UP'
14. type 'UP'
15. type 'nine'
16. type 'ten'
17. type 'DOWN'
18. type 'eleven'
19. type 'twelve'
20. type 'UP'
21. type 'thirteen'
22. type 'fourteen'
23. type 'DONE'

If you've typed these lines correctly, then the program should display the following on the screen:

```
Line Number and Line
LINKS
```

| | | | |
|----|--------|----|----|
| 1 | one | 2 | 0 |
| 2 | two | 9 | 3 |
| 3 | three | 4 | 0 |
| 4 | four | 7 | 5 |
| 5 | five | 6 | 0 |
| 6 | six | 0 | 0 |
| 7 | seven | 0 | 8 |
| 8 | eight | 0 | 0 |
| 9 | nine | 10 | 0 |
| 10 | ten | 13 | 11 |
| 11 | eleven | 12 | 0 |
| 12 | twelve | 0 | 0 |

| | | | |
|----|----------|----|---|
| 13 | thirteen | 14 | 0 |
| 14 | fourteen | 0 | 0 |

The first column is printing the array location for the current line, then (with an appropriate indent) the contents of LN\$ at the current array location. The last two columns give the links. The first number is the link to the current level and the second number is the link to the next lower level. See if you can diagram it. You should see almost immediately how each line is linked to the next in order.

What have we learned?

In this article, we've learned some basic data structures which allow us to extend our program control. List linking will be the primary structure for controlling our outline program. The stack structure is needed to allow us to 'Traverse' the list structure for a complicated list.

In our next article, we're going to learn how to set up the screen display for our outline program so that we can combine what we've learned here with what we learn in screen control to build an outline program we can use as a thought processor. •

Link.Bas - Demo Program

```

10 REM - Simple List Linking
20 DIM LN$(20), LK(20)
30 NX = 1
200 REM - Main Loop
210 CLS:PRINT"Simple List Linking":PRINT
220 GOSUB 2000
230 PRINT
240 PRINT "DONE - End of input, print the list"
250 PRINT
260 LINE INPUT ">>>";IN$
270 IF IN$="DONE" THEN 400
280 GOSUB 1000
290 GOTO 200
400 REM - End of program
410 GOSUB 2000
420 END
1000 REM - Add line to linked list
1010 J=NX:LN$(NX)=IN$:LK(NX)=0:NX=NX+1
1020 IF LK(0)=0 THEN LK(0)=J:RETURN
1030 I = LK(0):K = 0
1040 IF LN$(J)<LN$(I) THEN LK(J)=I:LK(K)=J:RETURN
1050 K=I:I=LK(I)
1060 IF I=0 THEN LK(K)=J:RETURN
1070 GOTO 1040
2000 REM - Print linked list
2010 IF LK(0)<1 THEN RETURN
2020 I = LK(0)
2030 PRINT I;LK(I);LN$(I)
2040 I = LK(I):IF I=0 THEN RETURN
2050 GOTO 2030

```

List.Bas - Demo Program

```
10 REM - list Linking Demonstration
20 REM - Terry R. Dettmann for Codeworks Magazine
30 DIM LN$(100), LK(100,1), LL(100), LV(10)
40 CL = 0:LV = 1:LL = 0
200 REM - Main loop
210 CLS:PRINT "LIST LINKING DEMONSTRATION":PRINT
215 GOSUB 2000:PRINT
220 PRINT "ENTER LINE OR"
230 PRINT "  UP - move up one level"
240 PRINT "  DOWN - move down one level"
250 PRINT "  DONE - done with entry, print list"
260 PRINT
270 LINE INPUT ">>";IN$
280 IF IN$="UP" THEN GOSUB 1000:GOTO 200
290 IF IN$="DOWN" THEN GOSUB 1100:GOTO 200
300 IF IN$="DONE" THEN 400
310 GOSUB 1200
320 GOTO 200
400 REM - end of program, print the list
410 GOSUB 2000
420 END
1000 REM - Move up one level in the list
1010 IF LV=1 THEN RETURN
1020 LV = LV - 1
1030 LL = LV(LV)
1040 RETURN
1100 REM - Move down one level in the list
1110 IF LV=10 THEN RETURN
1120 LV(LV) = CL:LV = LV + 1
1130 RETURN
1200 REM - Enter a line in the list
1210 CL = CL + 1:LN$(CL) = IN$:LL(CL) = LV
1215 IF LL(LL) <> LV THEN 1240
1220 LK(LL, 0) = CL:LL = CL
1230 RETURN
1240 LK(LL, 0) = 0:LK(LL, 1) = CL:LL = CL
1250 RETURN
2000 REM - Print the list
2010 IF CL<1 THEN RETURN
2020 I = 1:VI = 0:LI = 0
2025 PRINT"Line Number and Line";TAB(60);"LINKS"
2040 PRINT I;STRING$(VI*5," ");LN$(I);TAB(60);LK(I,0);LK(I,1)
2045 IF LK(I,1) <> 0 THEN LI=I:GOSUB 2100:VI=VI+1:I=LK(I,1):GOTO
    2040
2050 I = LK(I,0)
2060 IF I<>0 THEN 2040
2065 IF VI=0 THEN RETURN
2070 GOSUB 2200:I = LK(LI,0):VI = VI - 1
2080 GOTO 2060
2100 REM - Add li to the stack
2110 IF SP>=10 THEN RETURN
```

Random Files

Finally, a Ranidx that works!

Terry R. Dettmann, Associate Editor. After two unsuccessful attempts we have finally got Ranidx.Bas completely checked out. The "Shell" calls in this program are for MS DOS users; Tandy IV people will need to change this to "System" calls in three places. Tandy I/III users should continue using Ranindex.Bas since we haven't found system calls for those machines.

After the last issue appeared, we found (and some of you pointed out) some errors in the random indexing program. All we can do is ask you to accept our apologies for letting the errors through. In this issue, we're going to learn from them by correcting the errors and by using them to show how you can avoid similar errors.

The basic problems with the indexing programs stemmed from special cases which were unfortunately masked in testing because of the data base used to test the system. What I'm going to do is show the changes in the program, line for line and explain them, and then give a full listing of the corrected version of the program for your use.

First, let's start at line 250. How many of you caught this one? If you look at your original listing, the line reads:

```
250 IF FP$(1)="DELETED" THEN 270
```

However, it should read:

```
250 IF INSTR(FP$(1),"DELETED")>0 THEN 270
```

The reason (which hasn't really been pointed out here before) is that the first field (FP\$(1)) might be greater than 7 characters long (the length of the word DELETED). If it is, then the string FP\$(1) will be the seven characters of the word DELETED for a deleted record plus blank spaces to fill out the field.

When we're working with random files, fields are always blank padded to fill them out completely. This

means we have to take into account the possibility that this field is more than seven characters long by using the INSTR function to locate the string if it's there. This also raises another point, the first field cannot be less than 7 characters long! If it is, our unique marker (the word DELETED) can't be fit into the field and it will be truncated. We could use a shorter marker or another method, but so far this is good enough.

Our next change is not a correction, it's a debugging change only. We've changed line 510 from:

```
510 CLS:PRINT"All Done":CLOSE:END
```

to

```
510 PRINT"All Done":CLOSE:END
```

By eliminating the CLS, what is left on the screen when we close will still be visible. You can add it back in when you're confident everything is working alright.

With these changes done, everything else deals with our final step in the sorting process, the multi-file merging process which starts at line 4000. Our first change is to get rid of the single output file opened in line 4020. We can't output to this file until the whole merge is completed, so we change lines 4020-4030 from:

```
4020 OPEN "R",1,FI$,2
```

```
4030 FIELD 1, 2 AS XX$:NR = 1
```

to

4020 N=0
4030 NR=1

The variable N is introduced to handle a series of output files which will be in the same format as the temporary files. New line 4045 calls subroutine 440 where we'll create this:

4045 N = N + 1:GOSUB 4400

Subroutine 4400 will concentrate on getting the right file prepared for use by the program:

```
4400 REM — open intermediate file n
4410 FX$ = "TMP"+MID$(STR$(N),2)+".XXX"
4420 OPEN "O",1,FX$
4440 RETURN
```

Each intermediate output file will now be a simple output file (like the temporary files) instead of a random file (like the final index). We have to do this since if the file size is too large (we can't sort it all in a single pass) we may have to merge and remerge several times.

Next, we have to provide for temporary file handling, so we change our closing step of the merge (if no open file has another index record to read) by changing line 4060 and adding line 4075. The original line 4060:

4060 GOSUB 4200:IF NOT FOUND THEN 4080

is changed to refer to line 4075 instead of line 4080 so we can close out files and link in subroutines 4500 and 4600 which will deal with our temporary files and setup the output file for the next merge step. Lines 4060 and 4075 now read:

```
4060 GOSUB 4200:IF NOT FOUND THEN 4075
4075 CLOSE:GOSUB 4500:GOSUB 4600
```

Subroutine 4500 deletes the temporary files which have been merged into the current output file:

```
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
```

and subroutine 4600 renames the output file to make it ready for the next merge cycle:

```
4600 REM — rename output file
4610 TF = N:GOSUB 3200
4620 SHELL "REN "+FX$+" "+FT$
4630 RETURN
```

New line, 4085 recognize that more than one output merge file has been created and restarts from the beginning of the merge cycle (line 4000):

4085 IF N>1 THEN TN=N:GOTO 4000

Once only one merged file is left, line 4086 calls subroutine 4700 where we read the merge file and write out the index file in one pass and then delete the merge file:

4086 GOSUB 4700

The actual work is done by subroutine 4700 as follows:

```
4700 REM — build the final output file
4710 OPEN "R",1,FI$,2:RN = 1
4720 FIELD 1, 2 AS XX$
4730 TF = 1:GOSUB 3200
4740 OPEN "I",2,FT$
4750 IF EOF(2) THEN 4800
4760 INPUT #2,IX,IX$
4770 LSET XX$ = MKIS$(IX)
4780 PUT#1, RN:RN = RN + 1
4790 GOTO 4750
4800 CLOSE
4810 SHELL "ERASE "+FT$
4820 RETURN
```

One of the most subtle errors though was an array naming error which only affected one record in the file.

You can see the change in looking at the old copy of lines 4140-4141:

```
4140 OPEN "I",K+1,FT$:  
INPUT#K+1,IX(K),DA$(K)  
4141 PRINT"FILE: ";K;" ENTRY=";IX(K);DA$(K)
```

vs. the new lines:

```
4140 OPEN  
"I",K+1,FT$:INPUT#K+1,IDX(K),DA$(K)  
4141 PRINT"FILE: ";K;" ENTRY=  
";IDX(K);DA$(K)
```

The error, not obvious at the start, is that the array IX is used instead of the array IDX. Since only one entry is affected, the result was always to lose a record in each pass through the merge cycle. OOPS!

The rest of the changes are part of the debugging code which was used to find the error. I'm leaving them in the program for your interest:

old lines:

```
4265 PRINT"FILE: ";LW;"  
ENTRY=";IX(LW);DA$(LW)  
4310 PRINT "ITEM (";NR;" ) = ";IX$  
4320 LSET XX$ = MKI$(IX):PUT  
#1,NR:NR=NR+1
```

new lines:

```
4251 PRINT"IX = ";IX;" IX$ = ";IX$  
4265 PRINT"FILE: ";LW;"  
ENTRY=";IDX(LW);DA$(LW)  
4266 PRINT"IX = ";IX;" IX$ = ";IX$  
4310 PRINT "ITEM (";IX;" ) = ";IX$  
4320 PRINT #1,IX;" ";IX$
```

There is never an excuse for an incorrect program so no apology is sufficient for presenting one. If a design is executed correctly, the program will also come out correct. However, even the best designs are subject to human failures, and ultimately, we're all human. I can only apologize for being more human than I thought I was.

Despite the problems with Ranidx.bas, you should know that a preliminary version of the random files system has been in use for over a year handling a 25000 name mailing list. This series of articles is built on a rewrite of the preliminary version which cleans up the code and develops it in a simpler fashion and corrects mistakes from the original. Sometimes though, the rewrites (as in this case) are not all that they're intended to be. Even now, with extended indexing, CodeWorks is preparing to move the mailing list system to the Random Files program both to have a better working environment AND to provide for a more complete testing platform for future additions to the program. If we find other errors in using this code, you can be sure you'll be the first to know.

Next time, we'll pick up again and add some more to our random system. With indexing in hand, we have a powerful system which allows considerable flexibility in data base layout and design. It doesn't match any of the standards and will never come to be the next Dbase or Rbase, but you do have the source code and can make it into anything you wish.

Main Listing Ranidx.Bas

```
10 REM - RANIDX.BAS - Random File Indexing - VERSION 2.0 JUN 88  
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 INSTR(FP$(1),"DELETED")>0 THEN 270
255 IF SX>0 THEN IF INSTR(FP$(SX),SX$)=0 THEN 270
260 GOSUB 1000
265 IF NX>=MX THEN GOSUB 2000:TF=TN:GOSUB 3200:GOSUB 3300:NX=0:
    TN=TN+1
270 NEXT RN
280 IF NX>0 THEN GOSUB 2000:TF=TN:GOSUB 3200:GOSUB 3300
290 GOSUB 4000
500 REM - End of Program
510 PRINT"All Done":CLOSE:END
550 REM - Save the program
560 'SAVE "ranidx.bas"
570 RETURN
600 REM - input a character
610 C$=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$:INPUT#K+1,IDX(K),DA$(K)
4141 PRINT"FILE: ";K;" ENTRY=";IDX(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$(LW)
4251 PRINT"IX = ";IX;" IX$ = ";IX$
4255 IF EOF(LW+1) THEN DA$(LW)="~~~":RETURN
4260 INPUT#LW+1,IDX(LW),DA$(LW)
4265 PRINT"FILE: ";LW;" ENTRY=";IDX(LW);DA$(LW)
4266 PRINT"IX = ";IX;" IX$ = ";IX$
4270 RETURN
4300 REM - save to index
4310 PRINT "ITEM (";IX;" ) = ";IX$
4320 PRINT #1,IX;" ";IX$
4330 RETURN
4400 REM - open intermediate file n
4410 FX$ = "TMP"+MID$(STR$(N),2)+".XXX"
4420 OPEN "O",1,FX$
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 - build the final output file
4710 OPEN "R",1,FI$,2:RN = 1
4720 FIELD 1, 2 AS XX$
4730 TF = 1:GOSUB 3200
4740 OPEN "I",2,FT$
4750 IF EOF(2) THEN 4800
4760 INPUT #2,IX,IX$
4770 LSET XX$ = MKI$(IX)
4780 PUT#1, RN:RN = RN + 1
4790 GOTO 4750
4800 CLOSE
4810 SHELL "ERASE "+FT$
4820 RETURN
5000 REM - read data map

```

More on Laser Printing

Heard any good computing horror stories lately? Wanna hear one?

We first ordered a NEC LC890 from one of those mail-order hardware outfits. They claimed they ship orders within 24 hours. A month later, with an anxious buyer of our regular typesetting gear getting impatient, the printer still had not arrived.

We finally told them to stuff it, and bought an Apple LaserWriter IINT locally. Problem was that it doesn't have a parallel port on it. We had to either stick an AppleTalk board in our PC and try to run it that way (for an extra \$350 or so) or run it off of our serial port.

It turns out that Apple calls a null modem adapter a "modem eliminator" and that's what we needed. Only problem was that no one seemed to know how it had to be wired up between a PC AT and the Apple LaserWriter.

After three days of utter frustration, trying the many combinations that exist (and not knowing all the while that we might have a defective printer) we finally found the right combination.

All we can say at this point is that we are glad that's over - and it sure works fine now.

Anyone having a similar problem can write or call and we'll fill you in on the details of the hookup.

Now all we have to do is learn how to use it to make a better looking issue. We're working on it.

Actually, our computer is lying to the printer and telling it that it's a Linotronic 100/300 typesetter. And the printer obviously believes it!

But that's what you gotta do sometimes.

```

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

Here are some important Tandy Model I/III addresses. If you find some of these in programs you are trying to convert you will know what they do and you should be able to program around them. The starting location is given in all cases. Some of these addresses would span maybe two or three more locations after the one given. Not all of them have exact counterparts in MS DOS, while others are unnecessary in MS DOS. As we find corresponding locations for MS DOS we will publish them in these notes. If you have any more to add to this list, please let us know about them.

14308 - address of cassette port relay
14316 - I/O addresses
14336-15359 - keyboard memory
15360-16383 - video display memory
16384-16895 - BASIC vectors
16396 - BREAK key jump vector
16409 - Caps lock switch
16412 - cursor blink switch
16416 - cursor address
16419 - cursor character
16424 - max lines per page plus 1

16425 - number of lines printed plus 1
16427 - line printer max line length less 2
16429 - DOS entry point
16455 - address of lowest usable memory location
16546-16547 - contains line number currently being executed
16548 - points to beginning of BASIC program
16561-16562 - holds end address of free string space
16598-16599 - holds end of memory address
16633 - end of BASIC program pointer
16633 - points to end of BASIC program
16635 - end of simple variables
16635-16636 - used for memory calculations
16637 - end of array variables pointer
16637-16638 - calculates FRE(X\$)
16872 - RS-232 input buffer one byte
16880 - RS-232 output buffer one byte
16888 - baud rate code
16889 - parity/word length/stop bit
16890 - RS-232 wait switch
16896-20991 - TRSDOS
16913 - cassette baud rate switch
16916 - video display scroll protect

Continues on page 38

Conversions

Observations while moving from Tandy III to MS DOS

Robert A. Hood, Bremerton, Washington. In last issue's Forum we mentioned that this article was in that issue. Space got us and it was left out. Here Mr. Hood tells about his conversion from a Tandy Model III to one of the Tandy MS DOS machines. We think you will find that there is more the same between them than there is different.

I recently purchased an MS-DOS computer with two drives and MS-DOS 3.20 and GW-BASIC 3.20. My previous system was a Tandy Model III, and I also had access to a Tandy Model IV. I found it desirable to convert many programs to MS-DOS.

I have found several areas of difference between the two computer systems. They are:

1. USR(?) calls
2. Syntax
3. Screen size
4. ASCII codes
 - (a) CHR\$(?)
 - (b) POKE
 - (c) PEEK
5. Control keys
6. CMD"?"
7. Error codes
8. Timer loops

A detailed explanation for each of the above areas is presented in this article. All the differences may not have been found, and therefore, I would like to encourage you to send any more you find to the CodeWorks Forum.

Getting Started

The first task is to get the BASIC program which you wish to convert transferred to an MS-DOS formatted disk. There are two ways to do this. One is to type the program listing in from the MS-DOS keyboard, remembering to insert a space before and after each keyword and variable. Except for short programs, this can be a time-consuming task and in addition to making corrections for conversion it will be necessary to make corrections for typing errors.

The other method is to use a commercial program to transfer from one system to the other. I am presently using PC Cross Zap, by Hypersoft. (Trscross, by Powersoft, is another excellent program for this pur-

pose. - ED) PC Cross Zap transfers TRS-DOS 1.3 and LDOS 5.1 programs for the Model III and transfers TRS-DOS 6.2 programs for the Model IV. For the Model III it also inserts a space before and after each keyword and makes some other syntax corrections. After the program has been transferred, make a backup copy on the same disk using extension .M3 for the Model III and .M4 for the Model IV. The original copy then becomes the working copy to be converted. This is done so that if somehow the working copy of the program becomes badly botched you can go back to the original and start again.

Next, if you have a printer, make a printed listing of the program (preferably double spaced.) This listing may be marked up to show program flow and changes made. You should now review the program listing, looking for ON ERROR GOTO statements. The first change to be made is to REMARK all the ON ERROR GOTO statements in the program and to mark the statements in the listing for later replacement. This is necessary to make the normal program errors appear on the video when they occur instead of going to the error trap routine.

USR(?) Calls

This is the most difficult problem in the conversion of programs. This function is used to call machine language subroutines from the BASIC program. To convert, it is necessary to know exactly what the subroutine being called does. Then you must write a BASIC routine or an MS-DOS machine language routine to replace it. If you are unable to do either of these tasks and the routine cannot be omitted, then there is no purpose in attempting to continue conversion of the program.

Syntax

This problem will occur often. Fortunately, it is easily found. Just run the program and a syntax error message will be displayed for each line in which it occurs. I have found the following errors: (see table 1)

The Model IV also uses PRINT@(row,col) and

MS-DOS uses LOCATE (row,col). If the PRINT@ statements are calculated values, the following subroutine may be used to convert the PRINT@ values for Model IV.

```
1000 ROW=INT(C/80):IF ROW<1 THEN ROW=1
1010 COL=C MOD 80:IF COL<1 THEN COL=1
1020 RETURN
```

Screen Size

The Model III screen is 16 lines at 64 characters per line, the Model IV screen is 24 lines at 80 characters per

line and MS-DOS is generally 25 lines at 80 characters per line. For program conversion it is best to use 24 lines at 80 characters per line. This causes very few problems with the display from a Model IV. For the Model III it is often necessary to re-format the video display as lines which previously wrapped around to the next screen line will now be extended over the wider MS-DOS screen. Also, the output on the MS-DOS screen without modification will not be properly centered. The extra eight lines available allow more data to be displayed and may be used to improve the original Model III display.

Table 1

| Model III | Model IV | MS-DOS |
|-------------------|-------------------|-------------------|
| No keyword spaces | Needs spaces | Needs spaces |
| Print@ used | Print@ used | Locate used |
| Print Using % % | Print Using \ \ | Print Using \ \ |
| Print Using [[| Print Using ^ ^ | Print Using ^ ^ |
| Kill"file/ext:0" | Kill"file/ext:0" | Kill"A:file.ext" |
| Load"file/ext:1" | Load"file/ext:1" | Load"B:file.ext" |
| Save"file/ext:1" | Save"file/ext:1" | Save"B:file.ext" |
| Filename in caps | Filename u/l case | Filename u/l case |
| THEN optional | THEN for each IF | THEN for each IF |

Table 2

List of Equivalent Statements

| Model III | Model IV | MS-DOS |
|------------|----------------------|----------------|
| CHR\$(8) | Backspace & erase | CHR\$(29)+" "; |
| CHR\$(14) | Cursor ON | Locate „1 |
| CHR\$(15) | Cursor OFF | Locate „0 |
| CHR\$(21) | Special Char. toggle | Not required |
| CHR\$(22) | Alt. Char. toggle | Not required |
| CHR\$(23) | Half-wide screen | WIDTH 40 |
| CHR\$(24) | Cursor left | CHR\$(29) |
| CHR\$(25) | Cursor right | CHR\$(28) |
| CHR\$(26) | Cursor down | CHR\$(31) |
| CHR\$(27) | Cursor up | CHR\$(30) |
| CHR\$(28) | Cursor home | CHR\$(11) |
| CHR\$(29) | Erase line restart | See note 1 |
| CHR\$(30) | Erase to end of line | See note 2 |
| CHR\$(31) | Erase to end display | See note 3 |
| CHR\$(127) | Plus or minus | CHR\$(241) |
| CHR\$(192) | Spade | CHR\$(6) |
| CHR\$(193) | Heart | CHR\$(3) |
| CHR\$(194) | Diamond | CHR\$(5) |
| CHR\$(195) | Club | CHR\$(4) |
| CHR\$(196) | Happy face | CHR\$(1) |
| CHR\$(197) | Frown face | CHR\$(2) |

| | | | |
|------------|------------------|------------------|------------|
| CHR\$(198) | < with underline | < with underline | CHR\$(242) |
| CHR\$(199) | > with underline | > with underline | CHR\$(243) |
| CHR\$(200) | Alpha | Alpha | CHR\$(224) |
| CHR\$(201) | Beta | Beta | CHR\$(225) |
| CHR\$(203) | Delta | Delta | CHR\$(235) |
| CHR\$(204) | Epsilon | Epsilon | CHR\$(238) |
| CHR\$(207) | Theta | Theta | CHR\$(233) |
| CHR\$(208) | Iota | Iota | CHR\$(168) |
| CHR\$(215) | Pi | Pi | CHR\$(227) |
| CHR\$(217) | Sigma | Sigma | CHR\$(229) |
| CHR\$(218) | Tau | Tau | CHR\$(231) |
| CHR\$(220) | Phi | Phi | CHR\$(232) |
| CHR\$(224) | Omega | Omega | CHR\$(234) |
| CHR\$(225) | Square root | Square root | CHR\$(251) |
| CHR\$(226) | Divide | Divide | CHR\$(246) |
| CHR\$(227) | Sigma | Sigma | CHR\$(228) |
| CHR\$(228) | Approx. equal | Approx. equal | CHR\$(247) |
| CHR\$(229) | Delta | Delta | CHR\$(127) |
| CHR\$(233) | Percent | Percent | CHR\$(37) |
| CHR\$(235) | Infinity | Infinity | CHR\$(236) |
| CHR\$(237) | 6 over 9 | 6 over 9 | CHR\$(21) |
| CHR\$(241) | Paragraph symbol | Paragraph symbol | CHR\$(20) |
| CHR\$(242) | Cents | Cents | CHR\$(155) |

The following Model IV Print CHR\$() are preceded by Print CHR\$(0);

| | | | |
|---------|------------------|-----------|------------|
| POKE 1 | English pound | CHR\$(6) | CHR\$(156) |
| POKE 2 | Vertical line | CHR\$(26) | CHR\$(179) |
| POKE 3 | 'over e | CHR\$(3) | CHR\$(130) |
| POKE 4 | U umlaut | CHR\$(10) | CHR\$(154) |
| POKE 5 | circle over A | | CHR\$(143) |
| POKE 6 | top right corner | | CHR\$(191) |
| POKE 7 | O umlaut | CHR\$(9) | CHR\$(153) |
| POKE 8 | slash O | | CHR\$(237) |
| POKE 9 | 'over u | CHR\$(13) | CHR\$(151) |
| POKE 10 | ~over n | | CHR\$(164) |
| POKE 11 | ^over u | | CHR\$(96) |
| POKE 14 | A umlaut | | CHR\$(142) |
| POKE 16 | ~over N | | CHR\$(165) |
| POKE 17 | o umlaut | CHR\$(1) | CHR\$(148) |
| POKE 18 | Slash O | | CHR\$(237) |
| POKE 20 | B | | CHR\$(66) |
| POKE 21 | u umlaut | CHR\$(10) | CHR\$(129) |
| POKE 24 | a umlaut | CHR\$(8) | CHR\$(132) |
| POKE 25 | 'over a | | CHR\$(133) |
| POKE 26 | circle over a | CHR\$(11) | CHR\$(134) |
| POKE 27 | 6 over 9 | | CHR\$(21) |
| POKE 30 | ,under C | | CHR\$(128) |
| POKE 31 | | | CHR\$(126) |

1. L=CSRLIN:LOCATE L,1:PRINT STRING\$(79,32);:LOCATE L,1

2. C=POS(0):LOCATE ,C:PRINT STRING\$(79-C,32);:LOCATE ,C

3 L=CSRLIN:C=POS(0):LOCATE ,C:PRINT STRING\$(79-C,32);:FOR

J=L+1 TO 23-L: PRINT STRING\$(79, 32);: NEXT J: LOCATE L, C

POKEs to non-screen addresses

| Function | Model III Poke | Model IV Poke | MS-DOS |
|-----------------|------------------|---------------|---------------|
| U/L case | POKE 16409,0 | POKE 116,0 | See note 1 |
| Sp. Char Switch | POKE 16420,1 | POKE 2964,8 | Not used |
| Set Mem. size | POKE 16561,0-255 | Clear memsize | Clear memsize |
| Set Mem. size | POKE 16562,0-255 | Clear memsize | Clear memsize |
| Scroll protect | POKE 16916,1-7 | POKE 2964,1-7 | See note 2 |

1. Any number other than 0 sets Caps only for Models III/IV. For MS-DOS DEF SEG=0:POKE 1047,32 sets num-lock, POKE 1047,64 sets caps lock, POKE 1047,96 sets both and POKE 1047,0 resets all.
2. The number poked (1-7) determines the number of video display lines to protect from scroll on Models III/IV. For MS-DOS you may use VIEW PRINT n1 to n2, where n1-n2 are the range of lines to be unprotected.

PEEKs to ROM Addresses

The following PEEKs may be used in Models III/IV

| Function | Model III Peek | Model IV PEEK | MS-DOS |
|----------------|----------------|---------------|------------|
| Printer status | PEEK(14312) | INP(248) | INP(889) |
| Check for key | PEEK(14400) | PEEK(2300) | See note 1 |
| Program start | PEEK(16548) | PEEK(28318) | See note 2 |
| Program start | PEEK(16549) | PEEK(28319) | See note 2 |
| Program end | PEEK(16633) | PEEK(29087) | See note 3 |
| Program end | PEEK(16633) | PEEK(29088) | See note 3 |

1. This PEEK checks to see if a specific key has been pressed and if so takes appropriate action. The following example for Model III demonstrates the checking of the left and right arrow keys where K=32 is the left arrow and K=64 is the right arrow. Note that the values returned from PEEK(14400) are not the ASCII values. These are values for specific keys: 1<ENTER>, 2<CLEAR>, 8<up-arrow>, 16<down arrow>, 32<left arrow>, 64<right arrow>, 128<space bar>.

```
500 K=PEEK(14400):IF K=32 OR K=64 THEN 1000 ELSE RETURN
```

The routine is the same for the Model IV, except that the PEEK address is (2300) and the values are K=8 and K=9 for the keys.

For MS-DOS the 8 and 9 are the <BACK SPACE> and <TAB> keys and the routine below may be substituted for the Model III/IV routine.

```
500 IK$=INKEY$:FOR J=1 TO 100:IF IK$<>"" THEN 510 ELSE NEXT J
510 IF IK$<>"" THEN IF ASC(IK$)=8 OR ASC(IK$)=9 THEN 1000
520 RETURN
```

2. This PEEK returns the least significant byte and the most significant byte of the BASIC program start address.
3. This PEEK returns the least significant byte and the most significant byte of the BASIC program end address.

Assignment of Control Keys

The use of the arrow keys, Shift-arrow keys and CLEAR key with the INKEY\$ function for control keys on the Models III/IV need to be changed for use on MS-DOS because GW-BASIC returns a two-character string for the arrow keys and does not have a CLEAR key. The routine below demonstrates use of down-arrow on the Models III/IV.

```
600 IK$=INKEY$:IF IK$="" THEN 600
610 IF ASC(IK$)=10 THEN 300
```

and for the down-arrow key in GW-BASIC, use this routine.

```
600 IK$=INKEY$:IF IK$="" THEN 600
610 IF LEN(IK$)=2 AND ASC(RIGHT$(IK$,1))=80 THEN 300
```

The following list provides the ASCII values for various keys:

| Key used | Model III | Model IV | MS-DOS |
|---------------|-----------|----------|--------|
| Left arrow | 8 | 8 | 75 |
| Right arrow | 9 | 9 | 77 |
| Down arrow | 10 | 10 | 80 |
| Up arrow | 91 | 91 | 72 |
| Shift l-arrow | 24 | 24 | 135 |
| Shift r-arrow | 25 | 25 | 136 |
| Shift d-arrow | none | 26 | 134 |
| Shift u-arrow | 27 | 27 | 133 |
| CLEAR | 31 | 31 | none |

The following routine returns the ASCII value of any key or key combination:

```
800 CLS:PRINT" IK$          LEN(IK$)          ASC(IK$)
810 IK$=INKEY$:IF IK$="" THEN 810
820 PRINT TAB(3) IK$,LEN(IK$),ASC(RIGHT$(IK$,1))
830 GOTO 810
```

Missing Model III Commands

The Model III has 15 CMD functions which are not directly supported by MS-DOS. Some of these commands need to be converted for use on MS-DOS. CMD functions accessed from BASIC programs are:

| Function | Model III | Model IV | MS-DOS |
|------------------------|--------------|---------------|-------------|
| Return to DOS | CMD"A" | System | System |
| Enable/disable break | CMD"B" | See note 1 | See note 1 |
| Display directory | CMD"D:0" | CAT 0 | Files A: |
| Display directory | CMD"D:1" | CAT 1 | Files B: |
| Execute DOS command | CMD"I","cmd" | System "cmd" | Shell "cmd" |
| Chg. date display form | CMD"J" | | Not used |
| Load machine program | CMD"L" | System "file" | Not used |

| | | | |
|------------------------|--------|--------------|------------|
| Sort a string array | CMD"O" | None | See note 2 |
| Check printer status | CMD"P" | INP(248) | INP(889) |
| Move to DOS and return | CMD"S" | System "cmd" | Shell |
| Video & printer output | CMD"Z" | See note 3 | See note 4 |

1. Enter BREAK ON or BREAK OFF from the DOS prompt.
2. SHELL "SORT [/R] [/+n] [<input pathname] [>output pathname]" reads input from keyboard or file specified by input pathname, sorts the data, and writes it to screen or file specified by output pathname. [] brackets indicate optional sort parameters, /R reverses the sort (z to A). /+n begins the sort at column n (default is 1).
3. Use SYSTEM "LINK *DO *PR"
4. This feature is not supported by MS-DOS but may be simulated.

Methods to provide dual output to both video and printer.

1. To simulate CMD"Z" and to alternately display and print text lines, the simplest method is to add LPRINT statements for all PRINT statements where dual output is desired. Since MS-DOS has a full screen editor and can edit line numbers, this is easy to do. Simply duplicate the PRINT line number and add an L to the PRINT in the duplicated line.

2. To alternately display and print video pages the following BASIC subroutine may be used in MS-DOS. It replaces LCOPY and has the advantage of allowing the user to specify the number of lines to be printed.

```

100 NL=N:GOSUB 5000 ' N=number of lines to print
5000 IF NL<1 OR NL>25 THEN NL=25
5010 DEF SEG=&HB800
5020 FOR ZX=0 TO NL*160 STEP 160:ZP$=""
5030 FOR ZY=0 TO 158 STEP 2
5040 ZP$=ZP$+CHR$(PEEK(ZX+ZY))
5050 NEXT ZY
5060 LPRINT ZP$
5070 NEXT ZX
5080 RETURN

```

The above subroutine may be modified to print a variable number of video screen lines by omitting printing of blank lines. To omit printing blank lines make the following change to the routine:

```

100 GOSUB 5000
5010 FOR ZX=0 TO 3998 STEP 160:ZP$=""
5035 IF PEEK(ZX+ZY)=32 THEN ZW=ZW+1
5055 IF ZW=80 THEN 5065
5065 ZW=0

```

To force a blank line to be printed with the above changes, use "PRINT CHR\$(255)" in place of "PRINT"

3. GW-BASIC in MS-DOS allows the assignment of devices. Another method that can be used to output to both screen and printer is to open the screen for output as #1 and open LPT1 (the printer output) for output as #1, then PRINT #1 to both of them. Don't forget to CLOSE #1 when you are done.

Timer Loops

Many BASIC programs use FOR...NEXT loops to time program delays. Because the Model III and IV have a slower clock speed than most MS-DOS machines, the loop length must be increased when using GW-BASIC. Some typical clock speeds are:

| | |
|-----------|-----------------|
| Model III | 2.02752 Mhz |
| Model IV | 4.055 Mhz |
| 8086 PC | 4.77 - 7.16 Mhz |
| 80286 PC | 8 - 10 Mhz |
| 80386 PC | 16 - 20 Mhz |

This means that when converting from the Model III, multiply the value of the loop counter range by 2.35 or 3.53 and from the Model IV multiply the value of the loop counter range by 1.18 or 1.77.

Miscellaneous Differences

1. Model III/IV special characters CHR\$(244)+CHR\$(245)+CHR\$(246) produce a hand pointing right. No such feature is provided for in MS-DOS.
2. On Models III/IV the statement 200 F=180:INPUT"Value (default=180)";F produces F=180 if ENTER is pressed. For GW-BASIC use the statement 200 INPUT "Value (default=180)";F:IF F<1 THEN F=180.
3. Models III/IV search all drives for a requested filename. MS-DOS searches only the current drive. However, by using the PATH internal command, any drive or sub-directory may be searched. The PATH can also be included in the MS-DOS Autoexec.Bat file so that it is automatically invoked on power up.
4. LOF() contains the number of the last record for Models III and IV. LOF () contains the length of the file in bytes for MS-DOS machines.
5. For the Models III/IV the function RND(X) produces random integers 1 to X. In GW-BASIC use INT(RND*X+1).
6. Model III uses [for exponentiation. Model IV and GW-BASIC use the caret (^).
7. The following statements are equivalent:

| | |
|-----------|------------------------------------|
| GW-BASIC | 100 LOCATE X,Y:RETURN |
| Model III | 100 PRINT@((X-1)*64)+(Y-1),:RETURN |
| Model IV | 100 PRINT@((X-1),(Y-1)),:RETURN |

Hard Disks

Questions and Answers

Al Mashburn, Technical Advisor. In response to many requests, Al has put together a few questions and answers about hard disks. If enough interest is shown in this subject Al is prepared to dig deeper and get into the more technical aspects of hard drives.

Rather than do another dry two-page article on hard drives, I thought I'd go for the question-answer format. Truth in print requires me to tell you that some of the questions are ones I thought you should have asked.

q. Why do I need a hard drive?

a. Because it is the quickest way to have something better than your friends. Seriously there are people that just plain do not need a hard drive. If you only use one application and it doesn't need a lot of storage, you may live happily ever after using only floppies. Those of you with TRS-80s most likely do not and will not have one. Most of the programs you use are designed to fit and run on one floppy, putting data on the second floppy. But when you get into the MS-DOS world, things change. For instance QuickBASIC 4 comes on three diskettes. Our business mailing list is over one megabyte long and would be a real pain to try to span over three or four floppies.

q. What are the differences between floppy drives and hard drives?

a. Well, let's start with the physical differences. A floppy disk is made of a flexible piece of plastic with a coating of oxide on it to store information much like a cassette tape. The head of the drive rubs right on the surface just like the heads on a tape recorder. The disk is held in a cover that has a soft lining on the inside to pick up oxide that has been rubbed off or any dirt that may get in. The 3 1/2 inch disk has a metal cover that slides over the head opening in the cover to keep dirt out, the 5 1/4 inch disk is just open to the world. The head moves up and down the opening to access the different tracks or cylinders of

information. On a single sided drive a pad is on the other side of the disk to make sure that the head is actually contacting the surface of the disk. On a double sided drive the heads do this job for each other. The floppy disk turns at 360 RPM.

The hard drive is a much different animal. The first difference is the reason they call it a hard drive. The hard drive is made up of one or more "platters" of aluminum with an oxide or sometimes a metal coating on them. The heads never come into contact with the platter (well almost never, when they do it is called a crash.) The oxide coating is very thin and can store bits on it at a much greater density than a floppy. The disk spins at 3600 rpm, and the heads "fly" over it at a distance of about 1/4 the diameter of a human hair (red). Obviously this makes the hard drive susceptible to shock and you should take care when transporting a computer with a hard drive installed.

With the greater speed and the higher bit density, much higher data transfer rates are found with a hard drive. Rates of 850,000 bytes per second are quite within reach of a desktop computer with a hard disk, compared with the floppy's 100,000 bytes. So speed is another difference between the floppy and hard disk.

q. What all do I need?

a. Generally speaking, no matter what type of computer you have you will only need two things. One is a controller. This is the interface between the computer and the drive. It has to know how to talk to both of them, so the data can be transferred back and forth. Until the IBM/AT class of computers, the controller had to be configured for what type of drive it was going to run with and if you added a second drive, it had to be the

same type. The newer controllers are capable of having two completely different styles of hard drives hooked up to them. For instance, I run one 30 meg and one 40 meg drive on my system at home. The second thing you need, of course, is the hard drive.

q. How big of a hard drive do I need?

a. As big as you can afford, with exceptions. If you own an MS-DOS machine, you can't have too big of a drive. DOS can only "see" 32 megs at this time, (although I have heard DOS 3.4 will break the barrier) but with special drivers you can either break up big drives into 32 meg chunks, or fool DOS into seeing bigger drives.

You TRS-80 people are the exceptions. The biggest drive that is really usable by you is 5 megs. The reason for this is kind of a two-part whammy. One is that the TRS-80 can only access 4 floppy and 4 hard drives (on the TRS-80 you break one hard drive into 4 "logical" hard drives). The other is that it can only keep track of so many "chunks" on a drive. Let me define a chunk.

Lets say that on a TRS-80 floppy drive, the sectors are 256k long. The system will usually group them into one 512k "chunk." What this means to you and me is that the smallest file I can write is going to be 512k long. Even if it is only a 100 byte BASIC file, it is still going to take up 512k. Now if your system can only keep track of so many chunks, and somebody hooks up a hard drive, what's a system to do? You, the guy in the back, right! You make the chunks bigger! Hey, no problem right? Wrong. Let's say we have a 5 meg drive, broken into four 1.25 meg drives. The chunk size on

these drives could be as much as 8k long. Obviously if you were trying to use a 10 meg drive, they would be 16k long. That means that the 100 byte BASIC program takes up 16k on the drive. You can see that there is a real limit for the TRS-80.

q. Can I use my hard drive for my TRS-80 and my MS-DOS machine?

a. Yes, although it isn't cost effective for just two computers.

Tandy has a network system called Network4 that can let you hook up to 63 MS-DOS or TRS-80 Models 3 or 4 up to a single hard drive. One machine has to be a "slave" and do nothing but be the traffic cop for the system but the others have access to the drive just like it was installed in their computers. Like I said, it is not cost effective for just two computers, but for a school or business that has a lot of either or both computers it makes a lot of sense. The adapters to put the computers on the net are only a couple of hundred dollars which is a lot cheaper than the \$500 that many vendors want for just a 5 meg drive for the Model 3 or 4.

If there is enough interest in it, a later article can cover the first steps on putting in a new hard drive. Starting with formatting and going through the setting up of sub-directories and paths. For once the TRS-80 guys have this one easy, since the hard drive is treated like a big floppy there is little or no difference in operation. The MS-DOS people have a real bag of worms on their hands if the drive isn't set up right in the first place.

Computing Notes, from page 29

16919 - time-date
16928 - route destination device designator
16930 - route source device designator
17129 - Level II BASIC pointer to 2nd program line
20992-28671 - Disk BASIC & DOS utilities
26810 - Disk BASIC pointer to 2nd program line
28672-65535 - user memory not used by DOS

Those of you who do not have WHILE and WEND as statements in your BASIC can easily program around them. WHILE basically says to do something while a certain condition exists, then quit. Check out the following code, which reads from a sequential file:

```
100 WHILE NOT EOF
110 INPUT #1, LNS$
120 WEND
130 CLOSE
```

which says that as long as we are not at EOF in the file, to read in LNS\$. If we do reach EOF, then goto 130 and CLOSE the file.

Without WHILE and WEND, your code could look like this:

```
100 IF EOF THEN 130
110 INPUT #1, LNS$
120 GOTO 100
130 CLOSE
```

Handy Order Form

| Qty | Item description | Price | Total |
|-----|---|---------|-------|
| | RENEW Subscription Nov/Dec 88 through Sep/Oct 89 | \$24.95 | |
| | All third year issues, Nov 87 through Sep 88 | \$24.95 | |
| | All second year issues, Nov 86 through Sep 87 | \$24.95 | |
| | All first year issues, Sep 85 through Sep 86 | \$24.95 | |
| | 1st Year Program Disk (issues 1 through 7) (Specify computer type below) | \$20.00 | |
| | 2nd Year Program Disk (issues 8 through 13) (Specify computer type below) | \$20.00 | |
| | 3rd Year Program Disk (issues 14 through 19) (Specify computer type below) <i>Available after 1 Sep 88</i> | \$20.00 | |
| | NEW! "Starting with MS DOS" 40-page book explains all | \$7.00 | |

Postage and handling charges already included in all prices.

Total _____

We can supply program diskettes for PC/MS DOS, TRSDOS 1.3 (Tandy Model 3), TRSDOS 6.x (Tandy Model IV) and most CP/M MBASIC formats, on 5 1/4 inch diskettes. When ordering diskettes, specify your computer type.

COMPUTER TYPE _____

Check/MO enclosed

Charge to my VISA/MC _____ exp _____

Ship to: Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to:
CodeWorks, 3838 S. Warner St. Tacoma WA 98409

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

VISA/Master Card only, we don't take American Express

788

Index Update

Additions to CWINDEX.DAT

Ledger.bas, reference, issue 17, page 4
Misc, reference to Locate x,y, issue 17, page 4
Misc, reference to ArcSIN in issue 16, issue 17, page 5
Misc, reference to TRSDOS 6.2 and date, issue 17, page 5
Beginning BASIC, direct cursor positioning, issue 17, page 6
Misc, program, Cursor1.bas, issue 17, page 8
Misc, program Cursor2.bas, issue 17, page 9
Misc, program, Cusror3.bas, issue 17, page 10
Random files, issue 17, page 11, sorting big files in Randemo
Ranidx2.bas, main program, issue 17, page 12, big sorts
Misc, program, easydate.bas, issue 17, page 17,

standardizes date
Dmaker.bas, main program, issue 17, page 18, aid to decision making
Etax88.bas, main program, issue 17, page 25, estimating quarterly taxes
Bio.bas, main program, issue 17, page 30, plot your biorhythms

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
POSTAGE
PAID
Permit 774
Tacoma, WA

CODEWORKS

Issue 19

Sep/Oct 1988

CONTENTS

| | |
|------------------------------|----|
| <i>Editor's Notes</i> | 2 |
| <i>Forum</i> | 3 |
| <i>Beginning BASIC</i> | 7 |
| <i>Hard Disks</i> | 9 |
| <i>NFL88.Bas</i> | 12 |
| <i>Stat88.Bas</i> | 18 |
| <i>Correl.Bas</i> | 22 |
| <i>Outline.Bas</i> | 30 |
| <i>Random Files</i> | 37 |
| <i>Renewal Form</i> | 39 |
| <i>Index/Download</i> | 40 |

Issue 19

Sep/Oct 1988

Editor/Publisher

Irv Schmidt

Associate Editor

Terry R. Dettmann

Circulation/Promotion

Robert P. Perez

Editorial Advisor

Cameron C. Brown

Technical Advisor

Al Mashburn

(c)1988 80-Northwest Publishing Inc. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, the publisher assumes no responsibility for errors or omissions. Unless otherwise noted, all programs presented in this publication are placed in public domain. Please address all correspondence to CodeWorks, 3836 South Warner Street, 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. 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, WA.

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.

Well, after all the hoopla of moving and switching to desktop publishing, we finally settled for a different office in the same building. So, no address or phone number changes and that's nice. And with a little more experience under our belts, this issue ought to look a little better than the last.

During the move I ran across and old book, entitled "Computers and How They Work." It was published by Ziff-Davis Publishing Co., in 1959. I remember it as being one of the first books about computers, and in those days there weren't very many. I took a few minutes to browse through the book before I realized what extraordinary changes have taken place in computing in the past 30 years.

The book gave a lot of coverage to magnetic core memories and shift registers. A core memory of 16K was considered state of the art. There were no video display terminals; everything printed out on a typewriter-like printer. Punched paper tape looked like the input medium of choice in those days.

Nowhere in the book could I find evidence of the concept of having the program and free memory occupy the same space. The program looked like it was wired up on some kind of plug-in panel. There was a whole chapter on "how computers remember." High speed tape and card punching were covered as though they were new concepts, and they probably were. Although they talked about serial and parallel adders, serial or parallel input or output was never mentioned.

The coup de grace of the whole

affair was near the end of the book, where they talked about then current systems. Here I'll quote directly from the book: "The IBM 704 requires 24 microseconds to perform addition and 240 microseconds to perform multiplication or division. Provisions are made for storing up to eighty-six instructions. Input-output rates via magnetic tape may be as high as 2500 words per second. This popular system carries a selling price of \$1,000,000 to \$2,500,000."

In case you missed it, those figures are in the millions! For contrast, the book itself, which was a very well done hard-cover book, had a price printed on its jacket - \$4.95.

Isn't it amazing that the price of computers has come down so much, while the price of books has gone up? Not only that, but computers offer much more than ever before for less money, while books seem to offer less for more money. Oh, well.

This issue marks the end of our third year at CodeWorks. It has been interesting and fun, and we hope you have found it worth your while too. It also means that we must, once again, solicit your renewals for another year. We appreciate the many of you who have already renewed and ordered our third year diskette. We would like to encourage the rest of you to renew as soon as possible, so that you don't miss out on the neat things we have planned for you during the coming year.

Aside from all that, it's football season again, and Autumn, and what could be nicer than that?

Irv

Forum

An Open Forum for Questions and Comments

...CodeWorks has been very helpful to me in understanding how a computer and BASIC programming work. Also, thanks for the tips on Genealogy. I have ordered "Clan" from Mr. Hurlburt (Forum, Issue 18) and am looking forward to using it.

Walter Evans, Jr.
Waco, TX

...Add one more to the list of people interested in Genealogy. I have seen a number of ads for programs to do genealogy, but none of them to be used with a Model III. Your note about the program from Arthur C. Hurlburt, to run on a Model III with TRSDOS 1.3 was of great interest. I am ordering a copy from him immediately. I have been told that the Mormon church has published a program which is supposed to be very good, but one needs an IBM PC or equivalent to run it, so I have never looked into it.

D. B. McRae
Grantsville, UT

I am glad to see that the genealogy business is creeping to the front. William A. Korroch (3806 Churchill, Lansing, MI 48911) has generated a genealogy program "Pedigree" that enjoys public domain. If you are interested in learning more about (the program) I am sure Bill will be pleased to "go the second mile or more." Thanks to you and your crew for CodeWorks!

S. A. Langell
North Canton, OH

Thanks for the information, but you didn't say what computer it runs on. From the other information you have sent we assume it runs on a Tandy Model III, although it does not say so explicitly.

This letter is in reference to one of your programs, Card.Bas, which appears in Issue 2. It is a very useful program. However, the program would be much more useful if, using option 4, you could sort on a secondary key as well as a primary key. In one of your future issues, could

you please publish a program that uses a secondary sort?

Anthony Ivy
Tuscaloosa, AL

We are working on version 2 of Card.Bas now, which basically does nothing more than give you the secondary sort you have asked for. In addition, we have worked up a little demo program to show how two-level sorting works. It will probably appear in a future installment of Beginning BASIC.

I have found a tremendous resource of interesting programs in your CodeWorks. Not only good programs but also their depth have given me a wonderful review of the whole BASIC language. One seems to need only a few commands when coding his routine chores and forgets the others and the power they provide...

J. M. Davis
Sun City, AZ

Isn't it the truth? We have found that the more tools you have in your toolbox, the simpler and less complex your code becomes.

You mentioned that you knew of no way to recover files saved with the "P" option (in MS DOS). B & J Enterprise, PO Box 485, Daleville, AL 36322 lists disk #1049 to unprotect them at \$3.50 (public domain catalog). I've not used it but thought you'd like to know.

Mrs. Ira Hynes
San Francisco, CA

Thanks. We solved the problem by never using the "P" option.

...May I make a few comments that might be worth considering? Many owners of the TRS-80 (I, II, III and IV) are either engineers or, at least technically competent. I think that most are not afraid to go into their machines and make repairs and alterations. I myself have modified a Model III and a IV to include an internal fan, built-in power supplies to accommodate four floppy disk drives, installed four half-inch high

drives (Teac seems to be the best) and, on the Model III, addition of Shuffleboard III CP/M.

The above thoughts lead me to believe that a really good attempt to change the I, III and IV into 16, or even 32, bit machines would be received with enthusiasm. Our babies should be able to do all that they ever did, plus anything new.

This is not really as crazy as it sounds. There are many of us out here. In fact, some of the best of us can be found in Australia, Canada, The Netherlands and many foreign countries. I am sure a fair percentage of us would be willing to commit ourselves to a comprehensive program of updating and modernizing. There is one thing most of the powers that be seem to forget: there are an awful lot of TRS-80's around.

One thing that might be a problem is the DOS. But, even for this, I think that MISOSYS and Logical Systems have shown a reasonable (more than could reasonably be expected) support and interest in the TRS-80. True, Logical Systems is bowing out, but even now, not completely. You can still get answers and friendly help simply by writing or calling them.

It seems to me that a program of the type suggested could extend the life of the old girls for many years, in fact, even into the 21st century...

**Maxwell L. Hall
Chicopee, MA**

I used to feel the same way about the Model A Ford, but I don't any more. Comments, anyone?

How does one print quotation marks? You have to use them in your programming but I don't know how to print a quotation mark. It cuts the balance of my sentence off if I try to use one and I find nothing in the manuals on it.

**Mrs. Ira Hynes
San Francisco, CA**

CHR\$(34) is the quote mark. You can insert it into your print statement wherever you want a quote mark to appear. As in: PRINT "Here is how to put a";chr\$(34);"mark in your print statement."

Do you know of any people who have typed at least some portion of the Bible (NIV or RSV) onto 5 1/4" disks using a TRS-80 Model III/IV and Superscripts? I am interested in swapping what I am doing for what they are doing. So far, I have most of the book of Genesis on disk. It is in files

of three chapters per file which can be merged into longer files using the Copy and/or Move commands. So I can keep track of where I am, the book name and chapter appear at least once on each "video" page. My goal is to type the first five books (Genesis through Deuteronomy). I am especially interested in corresponding with those who are working on the New Testament. Thanks for your help.

**Samuel Laswell
74214 Lambert Drive
South Haven, MI 49090**

No, but someone else might.

Please do not frighten me by considering the discontinuing of CodeWorks. Your publication is the only good resource for CP/M programs. I am just old fashioned enough to stick to my (Heath) H89A and CP/M, even though most computer publications have completely forgotten 8-bit and CP/M. I am still the slowest element in the system, so the hardware and software speed does not bother me.

Still having problems with conversion of the LOCATE command to CP/M. Just got to keep working at it to get it straight. Beginning BASIC in Issue 17 helped some, but I cannot get the demo program to run correctly. Keep up the good work. Cheers.

**B. T. Jeavons
Ocean Springs, MS**

We won't give up if you don't. Will someone with an H89A send us the exact syntax to make our LOCATE/Print@ routine work so that we can pass it along?

I have the following disk cleaning program which I use to clean disks in my TRS-80 Model IV. It works very well with the IV and I tried to convert it to my XT Clone using TRSCROSS but it stayed the same as before and would not convert.

```
10 CLS:INPUT"REMOVE ALL DISKS IN ALL
DRIVES AND PRESS ENTER":A$
20 INPUT"HOW MANY DRIVES DO YOU
HAVE":D
30 IF D<0 OR D>4 THEN 20
40 FOR C=0 TO D-1
50 PRINT"INSERT CLEANER INTO
DRIVE":C;
```

```
60 INPUT"AND PRESS ENTER";A$
70 FOR R=1 TO 9700:OUT 244,15:NEXT R,C
```

I would like to know if anyone at your facility or any of your readers might have a similar program for the IBM PC or clones? I could use such a program whether it is written in GW-BASIC, QuickBASIC or MS-DOS.

Clyde W. Preble
Mill Valley, CA

Why go through all that? At the DOS ready prompt, just put the cleaner disk into the drive you want to clean and issue a DIR - or are we missing something? To make the program work with MS-DOS you will need to identify your drives by letter (string), not by number. That's why it wouldn't convert with TRSCROSS.

Having been a CodeWorks subscriber from Issue 1 and having enjoyed and learned much from each issue, I am reluctant to terminate my subscription. However, it seems necessary at this time. I just don't have time to read, copy and use your articles which are not applicable to my present experience.

A total surprise brought it about.

My three sons thought old dad ought to have and gave me a new computer for my birthday. It has 256K of memory, external drive, hard drive, printer and programs for multi-tasking, word processing, publishing, spreadsheet, drafting and drawing. The only thing I had to get was an interpreter so I could transport the useful and fun things from the two previous systems.

At first the whole thing was so formidable that I almost wished they hadn't done it, but now, after a few months of getting acquainted with everything, it is special and may become profitable as well as fun.

I guess I have become a user. There just doesn't seem to be time to read and learn about programming. With that said, it is time for #194 to say thanks for the CodeWorks experience.

Louis B. Kelley #194
Crescent City, FL

I can understand the lure of commercial programs, having switched to desktop publishing, with a hand scanner for graphics and a CAD program to do drawings. But I'm trying to let it not overwhelm me. Thanks for your three years of support and good luck.

Something moved me to re-read Editor's Notes of the May/June issue and from there the Forum...

...About sixty years ago I had my first and only encounter with a hydraulic ram as a lad of 11 at my grandfather's. It was fascinating as it rammed the water up a high hill and onto the back porch of his home. You will do what you will, but April Fool's jokes have no place in a good publication like CodeWorks.

I remember quite vividly an article in the late 30's about a Lilliputian radio, if the publication still exists, they've eaten their words many times over. 'Course I fell for the article hook, line and sinker.

September, a year ago, I sold my weekly newspaper, The Carlisle Mercury, in its 120th year of continuous publication. In August 1962 we changed our way of printing from letterpress to offset...

In March 1987 I purchased Apple's Laser Plus, two Mac's, one with 20 meg hard disk and by April we had shoved the (typesetters) to the sidelines. Type faces are not as esthetic, but time and technology will cure that I'm sure! Costwise it's a time saver and frees one from a silver coated paper and its chemicals. Considering the size type faces used in CodeWorks there are not too many who can tell the difference. It might be the way to go...

Warren R. Fisher
Carlisle, KY

And we did. Since you got your laser printer they have added Postscript, 13 type families in 35 fonts (all in the printer ROM) and the ability to set type sizes in 5 to 127 point in half-point increments. The Apple LaserWriter II also does smoothing on those little "stair-step" lines you get on your screen (but only with Draw-type graphics, not with bit-mapped). The first commercial job (a technical manual) I did with this system I had to do over three times to get it right, but the experience was worth it all and should show in this issue. Sometimes you simply have to jump in with both feet - and kick a little bit. And no more stinking chemicals to go sour and wasted, expensive silver coated paper to wrap around the processor rollers. Isn't technology wonderful?

Although we don't normally do product reviews, we do from time to time get products that may be of interest to you. It is especially interesting to note that two of the three following items are for Tandy Models III/IV, after we had thought that support for these machines had faded.

John M. Gregg, of **TRY-O-BYTE, 1008 Alton Circle, Florence, SC 29501 (803) 662-9500** sent us his 1988 version of **TAX ESTIMATOR** for the TRS-80 Models III, IV, MS DOS, and Tandy Color Computer. The program is available to users for the cost of shipping and handling, \$5.00. John also included a copy of his "Report to Users," Summer 1988, and since it carried the heading "Volume 1 Number 1," we assume this report will be an on-going thing.

Subscriber Tim Sewell has sent us information on **The File Cabinet - Public Domain Software for your TRS-80,** PO Box 4295, San Fernando, CA 91342. He says it's a "Download through the mail." Over the years, The File Cabinet has collected TRS-80 software from all over the country. The programs have been checked, sorted, and cataloged into the largest collection of TRS-80 public domain software you

will find. There are programs separated into categories such as, Utilities, Games, Education, Business and Communication. He even has a high resolution catalog available with a READ-MAC picture file catalog in the works. A two-disk catalog of TRS-80 Model IV software is available for only \$5.00 which is refundable with your first order. Tim also has information (which you can ask for) on a TRS-80 Model I/III catalog.

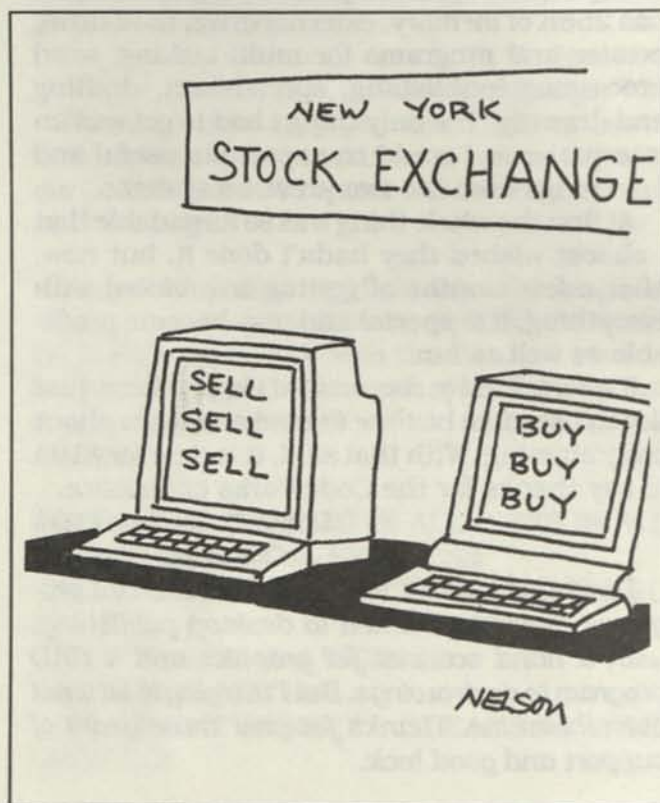
Prime Solutions Inc., 1940 Garnet Ave., San Diego, CA 92109 makes some rather fantastic claims for their program, "Disk Technician+." It is an artificial intelligence program that, according to Prime Solutions, will predict, detect and repair hard disk problems on the most fundamental level possible: that of the single bit soft error. In other words, it is software that repairs hardware! The program requires PC or MS-DOS versions 2.1 or higher and at least 384K of RAM. But how can it do all that? Disk Technician+ writes and reads to every single byte and bit on the hard disk, occupied or not, using special proprietary testing and repair algorithms. This process makes certain that every byte and bit is tested for soft error rate.

continues on page 38

It's **RENEWAL** time.

If you have not already renewed your subscription then this is your **LAST ISSUE!**

Use the order form on page 39 to keep it coming!



Beginning BASIC

Exploring the PRINT USING command

It is very likely that Print USING is one of the most powerful statements in the BASIC language. It is equally likely that most programmers do not use but a portion of the capability of that statement. In this installment we will take an in-depth look at Print USING. Before we go on, however, let's say that everything you can say about Print USING can be said for LPRINT USING as well. In our discussion we will simply refer to Print USING, keeping in mind that you can also LPRINT USING.

Print USING is a formatting statement. You first define a format and then assign a variable to use that format. The syntax of the Print USING command is: PRINT USING format;variable. The trick part of this command is in the format part. You can do very many interesting things with it, as we will see shortly. Print USING is used to format output, either to the screen or to the printer. It automatically right-sets numbers, so that dollars and cents come out in nice columns, with the decimal points all lined up. Let's take an example:

```
100 A=12.34
110 PRINT USING "##.##";a
```

will print: 12.34

That's simple enough, but watch what happens in the next example:

```
100 A=12.34:B=1.35:C=123.45
110 PRINT USING "###.##";A
120 PRINT USING "###.##";B
130 PRINT USING "###.##";C
```

This will print:

```
12.34
 1.35
123.45
```

Now let's look at the above example in a little more detail. The number signs are used to represent numeric positions. The decimal point will operate *at the position where you put it* in the format string. If you only had one number sign after the decimal point and variables A, B and C were two decimal places, then the Print USING statement would use automatic four-fifths rounding. That's kind of nice. If you have more positions to the right of the decimal than you have in your variable, those positions will be filled with zeros. If your format had three number signs and variable A was four places before the decimal, the Print USING command will print the value of variable A but will put a percent (%) sign in front of it to indicate to you that there was an overflow. As we saw earlier, if there are less positions in the variable than there are positions in the format, the number will always be right-justified.

If you are printing large numbers (especially in dollar amounts) you can put just one comma *anywhere* between two of the number signs before the decimal point and the output will be grouped in three's with a comma separating the groups:

```
100 A=123456789.23
110 PRINT USING "#####.###";A
```

will print: 123,456,789.23

You can even put other characters in the format if you like. Here's an example:

```
100 A=123.45
110 PRINT USING "@#####.##";A
```

and it will print: @ 123.45

But what if you want to actually print a number sign in the output? Well, you can do that too, by putting an underscore before the

position you want to print out, as in:

```
100 A=123.45
110 PRINT USING "_#####.###"
```

which will print: # 123.45

You can place a plus sign at the beginning or the end of the format. This will cause the sign of the number, *either a plus or a minus*, to be printed depending on where in the format you placed the plus sign. Here's an example:

```
100 A=90.02:B=-80.99
110 PRINT USING "+###.###";A
120 PRINT USING "+###.###";B
130 PRINT USING "##.###+";A
```

will print:

```
+90.02
-80.99
90.02+
```

If you place a minus sign at the end of the format it will cause a minus sign to be printed *after* negative numbers only. Positive numbers will not be affected.

Asterisks (*) placed at the beginning of the format will cause leading spaces to be filled with asterisks. The number of asterisks also indicate that many more print positions in the format. This one is especially useful in writing checks, where you want asterisks leading right up to the dollar amount.

Two dollar signs (\$\$) placed at the beginning of the format will cause *one* dollar sign to be printed to the immediate left of the first numeric digit, as in: \$12.34. One dollar sign at the beginning of the format will print in the first numeric position of the format. For example, if the format is "\$###.###" and the amount is 12.34, the output will be \$ 12.34 (with a space between the dollar sign and the first digit.) Note that the dollar sign itself, in this case, acts like an additional numeric position in the format.

At this point you are probably asking if you can combine what we have covered so far. The

answer is yes! You can have dollar signs, decimal points, asterisks, commas and plus or minus following the amount, all in one format.

Here's one that isn't used very much in everyday operations. If you put four carets (^) after the last number sign, the output will be printed in exponential notation. If the variable value is .00023 and the format is "##.###^^^^", then the output will look like this: 2.30E-04

You can use the same format to print several variables:

```
100 A=12.34:B=123.45:C=1002.34
110 PRINT USING "#####.###";A;B;C
```

will print: 12.34 123.45 1002.34

Did you notice that the format is always inside quotes? That makes it a format *string*. This means that you can define the string somewhere early in your program, like in line 130 `XX$="###.###"`, then later you can simply say: `PRINT USING XX$;A`

You can also use more than one format. Look at this:

```
100 A=123.45:B=23.12
110 PRINT USING "$$###.### ##.###";A;B
```

which will print \$123.45 23.12

and hints that you can build a complete format line for a report, which you can.

Up to now we have only used numbers in our format. How about string variables? Well, you can use Print USING with strings too, but the format changes slightly. Here's an example:

```
100 A$="CodeWorks"
110 PRINT USING "\      \":A$
```

will print: CodeWorks

Not too impressive, is it? But you will learn to appreciate it when you use it in a program. The backslashes and the space between them define the length of the string that will be printed. If we

take two spaces from between the backslashes in the above example, the output will look like this: CodeWor

If you insist on printing the entire string, you can say: PRINT USING "&";A\$ and the whole string, regardless of length, will be printed. The ampersand (&) tells it to do that.

Some BASICs (usually before version 5.0) use the percent sign (%) instead of the backslash to define the string Print USING.

If, in the above example we had said: PRINT

USING "!";A\$ then only the C (first letter) of CodeWorks would have been printed. We have never figured out a practical use for this little twist.

So there it is. All the things you wanted to know about Print USING. You'll have to admit, it's one powerful command. You will appreciate it most when you have to format a printed report, and that is, after all, what it is for. In some languages they go through agony to do that, but BASIC hands it to us on a platter. All we have to do is use it.

Hard Disks

More about hard disks from Al

Al Mashburn, Technical Advisor.

So you just got that new 20 Meg hard drive, and you are going to put all your stuff on it and life will be great from now on, right?

Of course not, you know by now that life isn't that simple, and if it was, you would have to pay taxes on it. You have to do some planning *before* you put any files on it, or you will pay for your sins later. What we are going to do today, is start fresh and set up a new hard drive from scratch. If you already have a hard drive and you have found it to be a mess because you didn't do it right the first time, back up all your files and follow along.

I am going to start off by making a couple of assumptions. One, that the drive you are using is less than or equal to 30 megs, and the second is that the low level format has been done. If the drive is bigger than 30 megs, you will need special software that lets you use all of it. I suggest SpeedStore. You can find it or other, just as good, programs in the back of most PC magazines. If the drive is not low-level format-

ted, you now know why that guy was cheaper than the rest. You are on your own until you get the low-level done, because there are too many variations to cover here.

Before we lay a byte on the disk, lets do some deciding. What DOS version are you going to use? If you said anything below 3.1, go to the back of the room. There are just too many improvements in the new versions of DOS to waste the time formatting your disk with an old one. Bite the bullet and buy (yes, I said buy) a copy of DOS and get the manuals so you have them when you need them. At this time most of us have the choice of DOS 3.1 or 3.2. The main difference between them is the support of 3 1/2 inch disks in DOS 3.2. This support also costs 25k more of memory so it's up to you to decide if you are going to go to the smaller disks any time soon. If you buy PC-DOS from IBM and you don't have a true blue IBM computer, remember to bring your version of BASIC from your old system. The BASIC for IBM makes calls to ROM's that aren't in anything but an IBM. Also

If you don't have an IBM, *don't* use PC-DOS 3.3. Some cases of constant hard drive crashing on compatibles has been reported. In any case, never SYS a newer version of DOS to your hard drive, always re-format. I know some guys walk on water and never have a problem, but it's just a good idea. If the company that makes your computer has an MS-DOS version of 3.3 then you are OK to use it.

The first thing we have to do is the "high" level format. This is basically the same as formatting a floppy, except that it takes a lot longer. Assuming that we are doing the first drive, it will be drive 'C'. The command is `FORMAT C: /S`. The /S puts the system files on the drive so it will "boot" without a floppy. Depending on your DOS version you will get a stern warning that you are about to possibly end the world or at least lose all your data, and should we proceed? Some of the old Tandy systems would not even let `FORMAT` do a hard drive. They included a program called `HFORMAT` to do it. Same thing.

You can also add a /V to the format command, this will let you put a label on the drive. Personally if I see one more directory that starts with "Volume on drive C is Als_Disk", I am going to spit. Also if there is a label on the drive (if you are re-formatting an old drive), the `FORMAT` program might ask you what the name is, just as extra insurance that you do want to do this. If you forgot what the volume name is, just do a directory and read the top line.

If all is right with the world, the computer will start formatting the hard drive. Go get a cup of something, this will take a while. By the way, for those of you new to computers who may be having a hard time figuring out just what formatting does, think of it this way. The hard drive is like a giant warehouse. No matter how high it goes, you can't really store anything in it until you put some shelves up to store things on. Formatting builds the "shelves" on the hard drive so that data can be stored on them. The ID information in each of the empty sectors is like bin labels on shelves, identifying the position in the warehouse. The FAT (file allocation table) is like the master inventory showing what is on each of the shelves.

When the drive is formatted, it is time to copy files onto it. Do a `DIR` and make sure that `COMMAND.COM` is there. If not, copy it from the DOS disk. Now reboot, and make sure that the drive will indeed boot up. If it doesn't, and you get a `NON-SYSTEM` disk error, then you probably didn't put the /s in the format command. Back to square 1. If it does boot, great! Ya did good. Now type `PROMPT PG`. This will make the prompt show us what sub-directory we are in, so we can tell where we are. Now type `MD DOS`. This will make a sub-directory called `DOS`. Now type `CD \DOS`. This will change the active directory from the `ROOT` (no name) to the `DOS` directory. Your prompt should look like `C:\DOS>`. If not, make sure you did the `PROMPT` thing and watch that the back-slash is indeed a back-slash, and not the slash!

Put the DOS disk back into the floppy drive and type `COPY A:*.*` to copy all the files from the floppy to the current directory. I won't walk you through anymore of these so if something goes wrong later, go back to these examples. Do a `DIR`, and you should see all your DOS files. If not, go back and try again, ya screwed up.

If all is OK, type `CD \` to go back to the root directory. The Prompt should look like `C:\>`. Now make another directory called `BIN`. Go into that directory (the prompt should be `C:\BIN>`) and copy all of your favorite utilities. This would include mini text editors, listing utilities, color changers, etc.

Go back to the root directory. Make one more directory called `MENU`. You don't need to go there right now, stay in the root. It is time to make an `AUTOEXEC.BAT` file. `AUTOEXEC.BAT` will automatically be run right after the computer boots up if it exists, and there are some things we want to be done before we do anything else.

Type `COPY CON:AUTOEXEC.BAT` and press `ENTER`. You should see the cursor sitting on the left side of the screen blinking at you. Type these lines exactly, except for the `<ENTER>` at the end of each line. That just means to press `ENTER`.

```
PROMPT $P$G <ENTER>
PATH C:\DOS;C:\BIN;C:\MENU <ENTER>
CD\MENU <ENTER>
MENU <ENTER>
```

Now press the F6 key and press ENTER. You should get the "1 file(s) copied" message. What you have done is make a file that makes sure every time your computer is booted up, it has a prompt that tells you where you are, and a path set up so the computer can find the files it uses all the time. If you don't have a clock on board, you will want to put the DATE and TIME commands in, if you do and the clock needs software to set it, put the software in the BIN sub-directory and make sure you put the command to make it work *after* the PATH command, or it won't find it. *Do not* put the clock software in the root directory, we are trying to keep it clean.

Making sure that we are still in the root directory, type

COPY CON:CONFIG.SYS and press enter.
Now type

```
FILES=20 <ENTER>
BUFFERS=20 <ENTER>
DEVICE=\DOS\ANSI.SYS <ENTER>
F6 (the F6 key) <ENTER>
```

DOS always looks for a CONFIG.SYS file when it boots up. What this one says is that we want to have as many as 20 files open at one time, and we want reads to grab 20 buffers worth of data at a time. We also loaded ANSI.SYS. This file allows extended video commands, including changing colors, moving graphics, and re-defining keyboard keys. You may not need it right now, but it is a good idea to have it loaded, and it takes almost no memory. As an aside, if you have ever contacted a BBS and gotten all sorts of strange characters coming all over the screen, the BBS was probably putting out ANSI graphics which need ANSI.SYS to translate them into screen graphics.

Now we are on the last leg of setup, the menu. There are all sorts of "shell" programs out there that are supposed to make your computing easier, but I find that after you know DOS a little,

they just get in the way. The menu system we are going to do here is simple, and doesn't get in the way at all. Best of all you decide what you want it to do for you.

The menu we are going to do will only do two things, format a floppy disk and "park" the hard drive. Parking the drive just means pulling the read-write heads into a position where there is no data. Then if the computer is dropped or hit very hard and the heads "crash" into the media, there won't be any damage to the data. If the heads happened to be over your root directory when that happened, well you get the picture. Most all hard drives will come with a program to park the head, the one I use is called ZPARK, so substitute your software's name where you see ZPARK.

First we do the batch file that displays the menu. Type

```
COPY CON:MENU.BAT <ENTER> then
```

```
ECHO OFF <ENTER>
CLS <ENTER>
TYPE MENU.TXT <ENTER>
F6 (the F6 key) <ENTER>
```

You should see the "1 file(s) copied" message. If so, type

```
COPY CON:MENU.TXT <ENTER> then
```

```
My Menu System <ENTER>
```

```
1) Format a floppy disk <ENTER>
<ENTER>
<ENTER>
<ENTER>
<ENTER>
<ENTER>
<ENTER>
<ENTER>
<ENTER>
<ENTER>
```

```
10) Park Hard drive <ENTER>
```

```
F6 (the F6 key) <ENTER>
```

After the "1 file(s) copied" message, type

COPY CON:1.BAT <ENTER> then

```
FORMAT A:<ENTER>
MENU <ENTER>
F6 (the F6 key) <ENTER>
```

Let's look at what this simple batch file does. First it runs the format program, just like you had typed in the FORMAT command yourself. When it is done, it runs the MENU batch file again. If you have already typed these things in, you will notice that when the menu is displayed, the DOS prompt shows afterward. That's the nice thing about this type of menu, you can use it if you want, or go ahead and type the commands yourself. The menu never gets in the way. Let's do the last batch file, then you can fill in those spaces in MENU.TXT with your own, often used, commands. Type

COPY CON:10.BAT <ENTER> then

```
ZPARK (replace with your parking software
name) <ENTER>
F6 (the F6 function key) <ENTER>
```

Notice that in this batch file, we didn't run MENU again. That's because parking is the last thing you do before shutting off the machine.

OK, now it's up to you, this magazine is for learning and doing so now that you have learned, do! One suggestion would be to make a BASICP subdirectory, then add BASIC to the menu, changing directories to BASICP, and then running BASIC, returning to the MENU subdirectory when you exit BASIC and re-running MENU.

That is just one suggestion, it's your system and you know what you need. If you have set up your drive as outlined here, you have a good organized system to build on. Go for it!

NFL88.Bas

Our Oracle Tries Again

It's hard to believe, but it is that time again. As I write this two pre-season games have already been played.

We are printing both NFL88.Bas and STAT.Bas for those of you who may not have been with us during previous years. You old-timers, however, need only pay attention to the new schedule, contained in the DATA statements at the end of NFL88.Bas.

An easy way to incorporate these DATA statements into your existing program is to use Maker.Bas (from our very first issue - and also in the "March Sampler Issue" we put out at one time) to enter them. Then, if you numbered your lines properly when you used Maker, you can simply use the MERGE command to incorporate

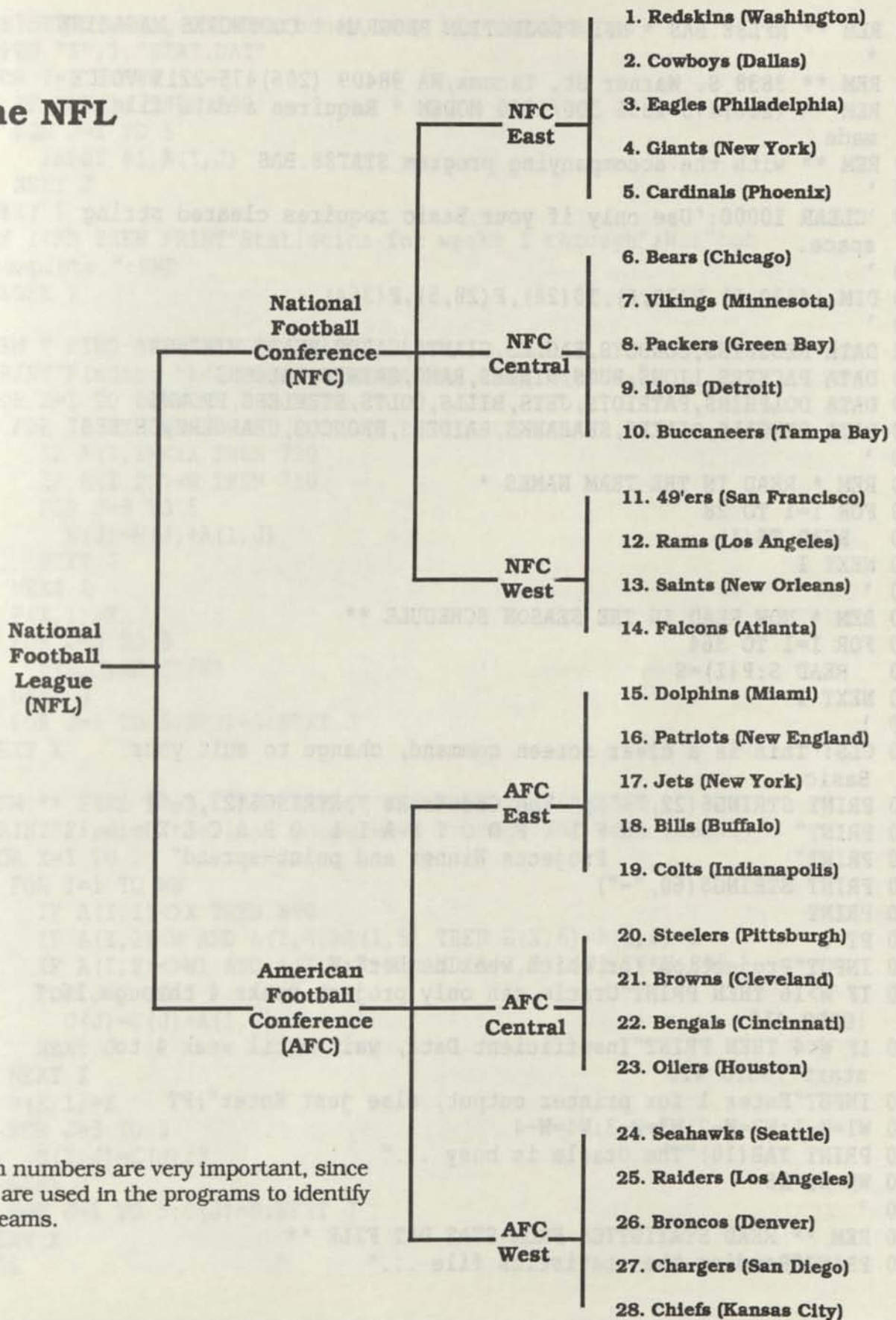
the DATA lines into your version of NFL88.Bas.

We are not listing changes for Tandy III or IV, simply because we believe most of you have those changes from last year's issues. In the event you don't have them and would like them, drop us a line and we will send them to you. There have been no changes (other than the schedule information contained in the DATA lines) that need to be made. Well, you might want to change the name of the program from NFL87 to NFL88, but that's no big deal.

We will again, as in past years, put the statistics (starting with week 4) on the download so that you can get them from there if you like.

Let's see if Oracle can do better than 60%!

The NFL



Team numbers are very important, since they are used in the programs to identify the teams.

```

100 REM ** NFL88.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 STAT88.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 88-89 NFL schedule for weeks 4 thru 16
1440 DATA 1,5,14,2,3,7,12,4,6,8,17,9,10,13
1450 DATA 11,24,15,19,20,18,21,22,16,23,25,26,27,28
1460 DATA 4,1,2,13,23,3,5,12,18,6,7,15,8,10
1470 DATA 9,11,24,14,19,16,21,20,22,25,26,27,28,17
1480 DATA 1,2,4,3,20,5,6,9,10,7,16,8,26,11
1490 DATA 12,14,13,27,15,25,17,22,19,18,24,21,28,23
1500 DATA 5,1,2,6,3,21,9,4,8,7,10,19,11,12
1510 DATA 13,24,14,26,27,15,22,16,18,17,23,20,25,28
1520 DATA 1,8,2,3,4,14,21,5,11,6,7,10,9,28
1530 DATA 24,12,25,13,17,15,16,18,19,27,26,20,23,22
1540 DATA 1,23,5,2,14,3,4,9,6,16,7,11,8,18
1550 DATA 15,10,12,13,26,19,20,17,22,21,27,24,28,25
1560 DATA 13,1,2,4,12,3,11,5,10,6,9,7,8,14
1570 DATA 15,16,17,19,20,22,21,23,18,24,25,27,28,26
1580 DATA 6,1,7,2,3,20,4,5,19,8,10,9,25,11
1590 DATA 13,12,27,14,18,15,16,17,21,26,22,28,23,24
1600 DATA 1,11,22,2,3,4,5,23,6,10,19,7,9,8
1610 DATA 27,12,26,13,14,25,16,15,17,18,20,21,24,28
1620 DATA 21,1,23,2,5,3,4,13,8,6,7,9,10,14
1630 DATA 11,27,12,26,15,17,18,22,16,19,28,20,25,24
1640 DATA 1,3,2,21,5,4,6,12,13,7,8,9,18,10
1650 DATA 11,14,19,15,24,16,20,23,27,22,26,25,17,28
1660 DATA 2,1,3,5,28,4,9,6,7,8,10,16,13,11
1670 DATA 14,12,21,15,25,18,19,17,22,23,26,24,20,27
1680 DATA 1,22,3,2,4,17,8,5,6,7,9,10,12,11
1690 DATA 14,13,15,20,18,19,23,21,24,25,16,26,28,27
1700 ` END of 1988-89 schedule data

```

```

100 REM * STAT88.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 NFL88.
130 REM * If no file exists then in command mode,type OPEN"O",1,"STAT
    AT.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 1988-89 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

```

Yes! We are working on a program to predict the outcome of the post-season games and the superbowl. It will be a standalone program, but will work with the Stat.Dat file that this program makes and maintains. Look for it in the November/December issue, which should give you just about enough time to get it in before the post-season playoffs start.

Correl.Bas

A Correlation Program with Lead/Lag Indication

Staff Project. Correlation is an interesting study. In this program we not only find the coefficients of correlation, but also provide for a novel way to check to see if one set of data "leads" or "lags" the other set.

The study of correlation theory is fascinating. It is complex, and often misused, and sometimes smacks of slight of hand or numerology. When two measures of the same thing go together so that it is possible to predict one measure from the other, they are said to be correlated.

The idea of correlation came first to Sir Francis Galton (1822-1911), who, in his own words was waiting at a roadside station for a train, "*poring over a small diagram in my notebook.*" What he envisioned was a method of expressing multiple causality in a single formula. Galton had been working with the phenomena of characteristics in families being carried through to later generations. He later wrote, "*It had appeared from observation, and it was fully confirmed by this theory, that such a thing existed as an 'Index of Correlation'; that is to say, a fraction, now commonly written as r , that connects with closer approximation every value of deviation (from the median) on the part of the subject, with the average of all the associated deviations of the Relative as already described. Therefore the closeness of any specified kinship admits of being found and expressed by a single term. If a particular individual deviates so much, the average of the deviations of all his brothers will be a definite fraction of that amount; similarly as to sons, parents, first cousins, etc. Where there is no relationship at all, r becomes equal to 0; when it is so close that Subject and Relative are identical in value, then $r=1$. Therefore the value of r lies in every case somewhere between the extreme limits of 0 and 1. Much more could be*

added, but not without using technical language, which would be inappropriate here."

Galton was an amateur in many fields, and was a first cousin to Charles Darwin. Perhaps it is because there was yet so much to be discovered in those days that amateurs could make such meaningful contributions. His correlation theory certainly made an impact on our present-day study of statistics.

There are several types of correlation: auto-correlation and multiple correlation among them. By far the most commonly used is the Pearson product-moment correlation coefficient, called r , and is the one we will consider here. This coefficient (r) can have values ranging from -1 through zero to +1. The sign of the coefficient indicates the direction of the relationship, with both -1 and +1 indicating perfect correlation and zero indicating no correlation whatever. A measure that decreases when its opposite measure increases will produce a negative correlation, and when an increase in one produces an increase in the other, it produces a positive correlation.

An example of perfect negative correlation would be the height above ground of two children on a seesaw, where there is no way the two can be moving up or down together; one must always be going up while the other is going down and vice versa. On the other hand, the relationship between the radius and the circumference of circles is an example of perfect positive correlation since a change in one will always produce

a change in the same direction in the other. Not all things vary in such perfection; there are degrees of correlation.

If you consider an X,Y plot with values plotted in, then the correlation coefficient, r , is the square root of the explained variation from the mean of Y divided by the total variation from Y. Or, r is the square root of the explained variation divided by the total variation. It may seem, at first, that a correlation coefficient of 0.6 would indicate a good correlation between two variables. To find out what percentage of the variation is explained, simply square the value. This will indicate that only 36 percent of the variation is explained leaving 64 percent unexplained, which is not as good as it first looked.

One of the more common mistakes made is that if two values are closely correlated, then there must be a cause and effect relationship. That is simply not true. Author Kimble (see references) cites a good example, he says: "*There is a positive correlation between the number of storks' nests in Holland, year by year, and the birth rate in that country, but this does not prove the theory that storks bring babies. What it means is actually sort of the reverse. As the number of babies increases, for whatever reason, they need more houses to live in. Houses have chimneys and that is where storks build their nests.*"

In yet another example it was found that the tree rings in the Southwest showed a variation that correlated well with the sunspot cycle. One would naturally assume that the sunspots, somehow, had an effect on the growth of tree rings. For all we know, there may even be such a correspondence. But as far as correlation theory goes, it would be just as valid to assume that the tree rings caused the sunspots! This, of course, sounds ludicrous but further emphasizes the fact that correlation is *not* causation.

For those interested, the references given at the end of this article treat the subject in great mathematical detail, going into accounting for variance, coefficients of determination, tests of

significance, standard error of estimates, and more.

It is interesting to note that r is a dimensionless term; one not expressed in terms of anything. Another interesting fact is that the relative magnitude of the data sets being compared makes no difference to correlation. For example, the series

1, 2, 3, 4, 5, 6

will correlate perfectly with

10.1, 10.2, 10.3, 10.4, 10.5, 10.6

or even with

41, 46, 51, 56, 61, 66.

The Program

Our program, Correl.Bas, not only calculates the correlation coefficient of two sets of data, it allows you to create as many sets as you wish and test for correlation between any of them. In addition, we have included a feature that allows you to shift either data set in relation to the other and test for leading or lagging correlation. The amount of this shift is controllable and variable within limits. Just because correlation does not indicate causation doesn't mean that if you find a leading stock market indicator that seems to be consistent you can't use it to your advantage. That's the whole idea of playing around with this program in the first place. You never know what to expect or what you will find. Making sense out of what you find is another matter, however.

Program Notes

Line 140 should remain remarked for BASIC versions past 5.0, otherwise, remove the remark. Line 150 sets the dimensions of four single dimension arrays. Arrays A and B are dimensioned at 50, and S and T for one-half that number. This means you can have up to 50 data points in each of the files you will be creating. If you find the need for more data per file, increase

the number in the A and B arrays to whatever you need and make the S and T arrays half that amount. Line 160 sets an error trap for "file not found." This error would occur when you ask for a file that does not exist. Further down in the program you will note that it does not create the file then, it simply re-runs the program so that you can start over and ask for a file that does exist (or make one first.)

The main menu of the program allows you to create and save data sets or read data sets and find correlations. The third alternative is simply a quit function. When you create data files you can create as many as you wish (being sure to give them each a unique name). The second option in the main menu lets you read in any two of the data sets you have created and check them for correlation.

The block of code from lines 340 to 480 is where files of data are created and saved. In line 340 you need to tell the program what the name of the file will be, and in line 350, how many data points you will be entering. Keep in mind that all your data sets need not be the same length, but the program will trim the longest file you call in to correlate to the length of the shortest, since correlation demands the same number of points in each data set. The file is saved immediately after you have entered the last of your data points. This happens in lines 410 to 450. The file is sequential, not random. You can remove a file by going to DOS level and killing it, or you can simply write a new file over the old one by using the same name. Lines 470 and 480 give you the option of continuing to add more data sets (files) or quitting.

If you pick option 2 from the main menu, you will come to the next section of code, starting at line 510. The lines from 520 to 600 ask for a file name, input the file and put the data into array A. It also prints the data from that file on the screen so you can see it. The lines from 630 to 710 do exactly the same thing, but with the data going into array B this time. One other thing is different between these two sections of code, that being the variables N1 and N2. These tell how many data items were in each file.

Lines 750 and 760 use N1 and N2 to determine which is the shorter of the two files, and let both N3 and N5 represent this number. The data in the longer file is not lost, it's just that data in the longer file past the length of the shorter file will not be used in this case. (You may want to use the longer file with an even longer one in a later comparison for correlation.) The shift range variable, K, is initialized to 3 in line 760. This will allow a shift of data up to one-third of the file's length. The reason for the "Press enter to continue" in line 770 is so that you can examine the data on the screen before going on.

After you have read in both files and pressed enter to continue, you come to line 790, which clears the screen. Line 800 then clears out the temporary arrays T and S, and line 810 sets N3 equal to N5. Back in line 760 we had set N5 equal to N3, so why do this? Well, when we start shifting the data in either of the files, N3 will get changed and we want to always know what the length of the shorter file is and N5 will always hold that number.

Line 820 tells you which two data files you are working with. Lines 850 through 890 are a sub-menu that tells you what you can do with the data you have just read in. You can find the correlation between the two, shift the first set of data and find correlation, shift the second set of data and find correlation, adjust the shift range or return to the main menu.

Note that in three places in this menu we are using variable data in the menu prompts themselves. This makes the menu prompt a little more informative than just a simple print statement would. In both lines 860 and 870 the name of the file is inserted into the menu prompt, and in line 880 the fraction representing the shift range is included inside the prompt itself. In addition, line 880 tells not only the fraction, but the total number of data points in the file. These "dynamic" menu prompts are interesting and easy to do, yet make the operation of the program easier to follow.

Note first that each of the following sections of code, corresponding to the different sub-menu items, each send program flow back to line 790

until you choose to return to the main menu. This means that once you have selected a pair of data files to work with, you can "play" with them with any of the sub-menu choices until you are satisfied.

Lines 940 through 970 are where the shift range can be adjusted (Option 4 of the sub-menu). Here, you can enter 2 for 1/2, three for 1/3, etc., to adjust how many of the total number of data points you can shift. You can change the default amount of shift by changing the value of K in line 760, so that it will always come up with 1/10th, for example. Keep in mind that data in your two files should correspond, that is, if one of them represents a value per month the other should also, otherwise, your results may not have any meaning (nonsense correlation). Also, when you shift, say three periods, then the other file will have three periods cut off its opposite end since we always need to be comparing an equal number of items.

The section of code from line 1000 to 1110 finds the unshifted correlation of the two data sets. The loop at line 1000 counts as many times as there are items in the shorter of the two data sets. When the loop is done, B contains the sum of all the values in the first data set, C contains the sum of the values in the second data set, D contains the sum of the squares of all values in the first data set and E contains the sum of the squares of all values in the second data set. F contains the sum of the products of each data pair (one from each data set.) Line 1070 sends us to the subroutine to calculate the coefficient of correlation.

In the section of code from 1140 to 1330 we find the coefficient of correlation again, but this time we shift data set one by the amount S by looking at A(I+S) in each iteration of the loop. Since we want to print out a range of coeffi-

cients, the array S() will hold the results until we have them all calculated, then the loop at 1280 will print them all out for us. The same thing happens in the next section of code, from lines 1360 to 1550, except that we shift the second data set by the value of T and store the results in the T() array until we print them out. Because we are calculating the coefficient of correlation several times inside these loops, line 1620 in the subroutine must clear out the intermediate values held in variables B, C, D, E and F after each calculation.

There is no guarantee that you will find the correlation you are looking for when you shift the data; if there is no correlation it just won't show up. But if one set of data indeed leads or lags the other it will show. It will also tell you how many periods of lead or lag to consider. For this reason, the periods corresponding to both sets of data should be the same.

The last little section of code is the error trap. If your BASIC is prior to version 5.0 remark line 1650 and un-remark line 1660. The trap checks for "file not found" errors and simply runs the program again to allow to you ask for the correct file or make one first.

Be careful when making assumptions about correlation. It gets rather involved and can be tricky, but in any case, have fun with it.

References:

How to Use (and misuse) Statistics, Gregory R. Kimble, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1978

Forecasting Methods for Management, Steven C. Wheelwright & Spyros Makridakis, Fourth Edition, John Wiley & Sons, New York, 1985

Theory and Problems of Statistics, Murray R. Spiegel, McGraw-Hill Book Co., New York, 1961

Correl.Bas for MS-DOS & Tandy IV

```

100 REM * Correl.Bas * CodeWorks Magazine 3838 S. Warner St.
110 REM * Tacoma, WA 98409 (c)1988 & placed in public domain
120 REM * (206)475-2219 voice and (206)475-2356 300/1200 modem
130 `
140 `CLEAR 1000 ` use if your BASIC is prior to ver. 5.0
150 DIM A(50),B(50),S(25),T(25)
160 ON ERROR GOTO 1650
170 CLS
180 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
190 PRINT"          C O R R E L A T I O N   P R O G R A M
200 PRINT"          finds coefficient of correlation in data sets
210 PRINT STRING$(60,45)
220 PRINT
230 PRINT TAB(10);"1 - Create and save data sets
240 PRINT TAB(10);"2 - Read data and find correlation
250 PRINT TAB(10);"3 - Quit
260 PRINT
270 INPUT"The number of your choice";Q
280 ON Q GOTO 330,510,300
290 GOTO 270
300 CLS:PRINT"Done":END
310 `
320 ` input data
330 CLS
340 INPUT"What will you name this file ";F1$
350 INPUT"How many data points will you enter ";N1
360 FOR I=1 TO N1
370   PRINT"Enter point ";I;:INPUT A(I)
380 NEXT I
390 `
400 `save the data in a file
410 OPEN "O",1,F1$
420 FOR I=1 TO N1
430   PRINT #1, A(I);
440 NEXT I
450 CLOSE
460 `
470 INPUT"Enter more data files (y/n)";Y$
480 IF LEFT$(Y$,1)="Y" OR LEFT$(Y$,1)="y" THEN 330 ELSE 170
490 `
500 ` read in two files
510 CLS
520 INPUT"What is the filename of the 1st data set ";F1$
530 OPEN "I",1,F1$

```

```

540 FOR I=1 TO 50
550   IF EOF(1) THEN 590
560   INPUT #1,A(I)
570   PRINT A(I);
580 NEXT I
590 N1=I-1
600 CLOSE
610 PRINT
620 `
630 INPUT"What is the filename of the 2nd data set ";F2$
640 OPEN "I",1,F2$
650 FOR I=1 TO 50
660   IF EOF(1) THEN 700
670   INPUT #1,B(I)
680   PRINT B(I);
690 NEXT I
700 N2=I-1
710 CLOSE
720 PRINT
730 `
740 `trim the longest file to the length of the shortest file
750 IF N2=<N1 THEN N3=N2 ELSE N3=N1
760 N5=N3:K=3
770 INPUT"Press Enter to continue ";Q1
780 `
790 CLS
800 FOR I=0 TO 25:T(I)=0:S(I)=0:NEXT
810 N3=N5
820 PRINT"You are working with data files ";F1$;" and ";F2$
830 PRINT
840 PRINT"You can:"
850 PRINT TAB(10);"1 - find their un-shifted correlation
860 PRINT TAB(10);"2 - Shift ";F1$;" and find correlation
870 PRINT TAB(10);"3 - Shift ";F2$;" and find correlation
880 PRINT TAB(10);"4 - Adjust shift range. It's now 1 /";K;" of ";
   N3
890 PRINT TAB(10);"5 - Return to main menu
900 INPUT"Your choice";Q1
910 PRINT
920 ON Q1 GOTO 1000,1140,1360,940,170
930 GOTO 900
940 PRINT"Shift range is how far shifting will occur.
950 PRINT"Enter 2 for 1/2, 3 for 1/3, etc."::INPUT K
960 IF K<2 THEN 950
970 GOTO 790
980 `
990 `find unshifted correlation

```

```

1000 FOR I=1 TO N3
1010   B=B+A(I)
1020   C=C+B(I)
1030   D=D+(A(I)^2)
1040   E=E+(B(I)^2)
1050   F=F+(A(I)*B(I))
1060 NEXT I
1070 GOSUB 1580 ' to calculate the coefficient
1080 PRINT
1090 PRINT"The coefficient of correlation is ";USING "##.##";CC
1100 INPUT"Press Enter to continue";Q
1110 GOTO 790
1120 '
1130 'shift F1$ data and find correlation
1140 N4=INT(N3/K)
1150 FOR S=0 TO N4
1160   N3=N5-S
1170   FOR I=1 TO N3
1180     B=B+A(I+S)
1190     C=C+B(I)
1200     D=D+(A(I+S)^2)
1210     E=E+(B(I)^2)
1220     F=F+(A(I+S)*B(I))
1230   NEXT I
1240   GOSUB 1580
1250   S(S)=CC
1260 NEXT S
1270 PRINT"The un-shifted coefficient of correlation is
";USING "##.##";S(0)
1280 FOR I=1 TO N4
1290   IF I=1 THEN P$="period" ELSE P$="periods"
1300   PRINT USING "##.##";S(I);:PRINT TAB(8);"when ";F1$;
   " leads ";F2$;" by";I;P$
1310 NEXT I
1320 INPUT"Press Enter to continue";Q1
1330 GOTO 790
1340 '
1350 'shift F2$ data and find correlation
1360 N4=INT(N3/K)
1370 FOR T=0 TO N4
1380   N3=N5-T
1390   FOR I=1 TO N3
1400     B=B+A(I)
1410     C=C+B(I+T)
1420     D=D+(A(I)^2)
1430     E=E+(B(I+T)^2)
1440     F=F+(A(I)*B(I+T))

```

```

1450 NEXT I
1460 GOSUB 1580
1470 T(T)=CC
1480 NEXT T
1490 PRINT"The un-shifted coefficient of correlation is
";USING "##.##";T(0)
1500 FOR I=1 TO N4
1510 IF I=1 THEN P$="period" ELSE P$="periods"
1520 PRINT USING "##.##";T(I);:PRINT TAB(8);"when ";F2$;
" leads ";F1$;" by";I;P$
1530 NEXT I
1540 INPUT"Press Enter to continue";Q1
1550 GOTO 790
1560 `
1570 `calculate cc subroutine
1580 AA=((N3*D)-(B^2))*((N3*E)-(C^2))
1590 BB=SQR(AA)
1600 CC=((N3*F)-(B*C))/BB
1610 CC=INT(CC*100)/100
1620 B=0:C=0:D=0:E=0:F=0
1630 RETURN
1640 `
1650 IF ERR <>53 THEN ON ERROR GOTO 0
1660 `IF (ERR/2)+1 <>54 THEN ON ERROR GOTO 0
1670 PRINT"There is no file with that name."
1680 INPUT"Press Enter to start over";Q1
1690 RUN 100
1700 END ` of program

```

Correl.Bas change lines for Tandy I/III

```

Changed->100 REM *Correl/Bas *CodeWorks Magazine 3838 S.Warner St.
Changed->140 CLEAR 1000 ` use if your BASIC is prior to ver. 5.0
Changed->1030 D=D+(A(I) [2])
Changed->1040 E=E+(B(I) [2])
Changed->1200 D=D+(A(I+S) [2])
Changed->1210 E=E+(B(I) [2])
Changed->1420 D=D+(A(I) [2])
Changed->1430 E=E+(B(I+T) [2])
Changed->1580 AA=((N3*D)-(B[2] )*((N3*E)-(C[2] )
Changed->1650 `IF ERR <>53 THEN ON ERROR GOTO 0
Changed->1660 IF (ERR/2)+1 <>54 THEN ON ERROR GOTO 0

```

Outline.BAS

Screen control for the outline program

Terry R. Dettmann, Associate Editor. The second of a series of three articles on an outlining program.

Last issue, we introduced some basic concepts which are fundamental to writing a useful outline program. We covered list linking and related issues, now we'll move on and cover the screen display structure. The sample program, `outline.bas`, illustrates what we're going to do and forms a shell for our final outline program. We'll modify this shell and add the actual outline code in the next issue.

Let's start right at the beginning where we've set up the program. Line 30 defines all variables to be integers unless specifically typed otherwise. Doing this makes the program faster (particularly in loops). While I could just use the percent sign (%) to mark only those variables as integer that I want to be integer, I find that it makes sense to force it this way because I'm basically lazy. If I don't do this I often forget to put the % on the variable name.

Line 40 introduces some constants that we'll need for the program. Some in particular will need to be changed to work on other computers (screen width, screen length, and so forth). The definitions are self explanatory. Line 50 introduces some arrays that we'll need in the final program to keep track of our outline lines. Line 60 defines some important characters which we'll need later in input routines and line 70 introduces the TRUE/FALSE variables which we use for decision making.

Next we go off to the subroutine at line 3550 to tell the program that no outline has been defined yet and then to line 3450 to set the program's COMMAND MODE. How the program will treat what you type in will depend on what

mode the program is in. We'll define several entry modes for the computer and whether we're entering a command or a line for an outline will depend on this mode. We could design a modeless program (one where no matter what you enter, the program interprets it), but that's quite a bit more complicated and we'll leave that for another series of articles.

The lines from 200 through 240 now are the main part of the program. It's pretty simple really when you get right down to what's going to happen:

- 1) Print the screen title and information area at the top of the screen (subroutine 1100)
- 2) Display whatever part of the outline should presently be visible (subroutine 1000)
- 3) Enter a command (subroutine 2000) and figure out what it is (subroutine 2500)
- 4) Execute whatever command was entered

The possible commands are:

| Subroutine | Command |
|------------|--------------------------|
| 3400 | Go into the ADD mode |
| 3500 | Go into the EDIT mode |
| 3600 | Go into the DELETE mode |
| 4000 | Load a new file |
| 4100 | Save the current outline |
| 500 | End the program |
| 3450 | Go into the COMMAND mode |

Each possible operation with the program is controlled by setting an appropriate mode.

Subroutines to execute the operations are provided in the 5000 series of line numbers (most are just stubs right now with no function, just there to hold down the space and allow the program to run at this level).

Let's start looking at the implementation routines now. At line 400 we have our screen position subroutine GOTOXY. We pass it the variables X (row) and Y (column) and it places us at the right place on the screen. If you're working with a non-MS DOS system, you'll replace this with the standard PRINT @ routine as always. Line 500 is our standard end routine, just to let you know it happened by the correct path.

The subroutine at 1000 is interesting because it shows us a little of the way the outline itself will go. I is our line counter (we're going to print no more than 20 lines to the screen). NM is set to the top line to show on the screen. From this, we simply print one line at a time onto the screen (subroutine 1200) and then move to the next line as determined by our linking scheme. If you don't remember what LIST LINKING is all about, then you better go back and review last issue's article in this series. Play with the programs until you understand them. While moving through the list, if we get to the end of a list of lines and there are no lines to go back to (the stack pointer SP is minus one), then there's nothing more to print. If we're at a line which we've printed and there are subordinate lines to this one ($LK(2,NM) \geq 0$) then we move down a level. If there's another line after the current one, we move to that, otherwise we move up a level.

Subroutine 1100 prints our top title line and the command menu line with the allowed commands. Subroutine 1200 prints a single outline line to the screen. I is the screen line number and NM is the outline line number. It's printed in the form:

```
<SPACES>NN. <LINE>
```

where <SPACES> is a number of spaces determined by the indent level of the particular line

(2 spaces per level) and <LINE> is the line itself. NN represents the line's position in the outline.

Subroutines 1300 and 1350 are used for backtracking by implementing a STACK data structure. Whenever we move down a level, we add the last line number to the stack with subroutine 1300. When we're done at this new level and want to go back, the top number on the stack is the line number we have to go back to.

When we reach line 2000, we're starting to really interact with the user. Subroutine 2000 implements a simple command interaction:

```
Prompt for a command  
Wait for an answer
```

Line 2010 goes to the command entry point, line 2015 clears that line and positions to the entry point, and line 2020 prompts for the command. We set the number of characters to accept equal to the width of the line minus 10 and then branch to the input subroutine (2100). Finally subroutine 2500 figures out what was entered.

If you've been reading my programs in Codeworks, you've seen subroutine 2100 before in many guises. Being dissatisfied with normal entry procedures and wanting more control, I created this entry routine to do the basic operations in a way that I could see and control them the way I wanted.

Line 2110 blanks the entry string (IN\$). Line 2120 gets a single character and if it's a RETURN or ENTER key, the subroutine ends. In line 2130 we check for a backspace to see if we need to make a correction (subroutine 2300 handles that). Line 2135 allows only PRINTABLE characters into our entry string and line 2140 allows the line to grow to no more than NC characters. If all the tests are passed, then line 2150 adds the character to the string and prints it on the screen.

Subroutine 2200 is our single character entry routine. It's been built to support arrow key

recognition (subroutine 2250 which at present does nothing). If an arrow key is found, it's handled immediately and another character is looked for. Subroutine 2300 handles BACK-SPACES when they occur (blank the character on the screen and eliminate it from the end of the string). Subroutine 2400 is our CLEAR TO END OF LINE routine. It just prints BLANK characters to the screen.

Subroutine 2500 checks for a command (first letter matches a command letter) if we're in the command mode. If we're in another mode we process it (line 2540). Subroutines 3000, 3100, 3200, and 3300 for moving on the screen with arrows are stubbed for the moment (we'll worry about them next time). Subroutines 3400, 3450, 3500, and 3600 are our mode switching routines. Each sets the correct mode AND a string which will appear on the screen to tell us which mode we're in. Subroutine 3550 is initialization for an empty outline.

The only other routines which are of any importance are the load and save routines. Subroutine 4000 loads an outline file. We enter the filename, add a '.OUT' extension to it and then let subroutine 5400 load it and create the outline from it. Subroutine 4100 does the same except it calls subroutine 5500 to save the file.

Subroutines 5100, 5200, and 5300 are the stubs for the most important functions in the program, the actual adding, editing, and deleting of lines. This will be the primary subject of the next article of the series.

The load routine (subroutine 5400) has a pretty simple structure (just wait 'til we get deep enough into it though!). It reads one line at a time from the input file. It determines the line's

level in the outline and then puts it into the list of lines (LN\$0). Once the line has been added, it lines it into the list (subroutine 5700). By repeating this process line for line, we eventually get the whole outline in. Subroutine 5600 is key here. It determines the line's level by counting the blank spaces at the front of the line and eliminating them as it counts. Subroutine 5700 is the most complicated process. Each line has to be added into the list at the appropriate level, linked to the lines before it and after it. Let's follow the subroutine's decisions and see what it does:

- 5705 if there is no line presently linked, then make this the first one.
- 5710 if the new line is at the current level (LL for LAST LEVEL) then link it in at this level (line 5800).
- 5720 if the new line is at a lower level, then start a new level for it at line 5850
- 5730 the new line should be linked to the line one level up

Each linking procedure goes to line 5900 at its end to do common linking tasks for the current line.

If we're going to save the current outline, we loop through the lines one at a time printing each one with one blank per outline level as we go. Everything about the routine is organized to follow the outline through level by level just as we did when printing a portion of it to the screen.

This installment we've laid out the screen control for our outline program. We'll tie it to the actual outline generation and control in the next issue. See you there.

Listing for Outline.Bas


```

10 REM - Outline.bas, a Program for Codeworks Magazine
20 REM - by Terry R. Dettmann
30 DEFINT A-Z
35 'n=max number of lines, nl=number of existing lines
36 'tl=top line on screen, cl=current line
37 'wd=screen width, ln=screen length
40 N = 200:NL = 1:TL = 0:WD = 80:LN = 24:CL = 0:SP = -1
45 'ln$( )=text for each node, lk( )=dynamic linking array
46 'lv( )=indent level of current text line
47 'nm( )=sequence number of current text line
50 DIM LN$(N), LK(2,N), LV(N), NM(N), LL(10), STK(10)
55 'cr$=carriage return, bs$=backspace
60 CR$=CHR$(13):BS$=CHR$(8):ESC$=CHR$(27)
70 FALSE = 0:TRUE = NOT FALSE
80 GOSUB 3550
190 CLS:GOSUB 3450
200 REM - Main Program Loop
205 GOSUB 1100
210 GOSUB 1000
220 GOSUB 2000
230 ON CMD GOSUB 3400,3500,3600,4000,4100,500,3450
240 GOTO 200
400 REM - gotoxy
410 LOCATE X,Y:RETURN
500 REM - End of program
510 CLS:PRINT "Thank you for coming":END
1000 REM - Display the current outline segment on the screen
1010 I=0:NM=TL
1020 IF I>= 20 THEN RETURN
1030 GOSUB 1200
1035 IF SP=-1 AND LK(0,NM)=-1 AND I>0 THEN RETURN
1036 I = I + 1
1040 IF LK(2,NM)>=0 THEN GOSUB 1300:NM=LK(2,NM):GOTO 1020
1050 IF LK(0,NM)>=0 THEN NM=LK(0,NM):GOTO 1020
1060 GOSUB 1350:IF NM<0 THEN RETURN
1070 IF LK(0,NM)>=0 THEN NM=LK(0,NM):GOTO 1020
1080 GOTO 1060
1100 REM - Print Title Line
1110 X=1:Y=1:GOSUB 400:PRINT "Codeworks Outline Processor";
1120 X=1:Y=WD-15:GOSUB 400:PRINT MD$;
1130 X=LN-2:Y=1:GOSUB 400:PRINT STRING$(WD,"-");
1140 X=LN-1:Y=1:GOSUB 400:PRINT "(A)dd (C)ommand (D)elete (E)dit
(L)oad (Q)uit (S)ave";
1150 RETURN
1200 REM - print a single line
1210 X = I + 2:Y=3:GOSUB 400
1220 PRINT STRING$(LV(NM)*2," ");

```

```

1230 PRINT USING "#.# ";NM(NM);:PRINT LN$(NM);
1240 RETURN
1300 REM - push level onto stack
1310 IF SP>=10 THEN RETURN
1320     SP = SP + 1:STK(SP) = NM
1330     RETURN
1350 REM - pop top level off stack
1360 IF SP<0 THEN NM=-1:RETURN
1370     NM = STK(SP):SP = SP - 1
1380     RETURN
2000 REM - command entry
2010 X=LN:Y=1:GOSUB 400
2015 NC=WD:GOSUB 2400:GOSUB 400
2020 PRINT "Command=>";
2030 NC=WD-10:GOSUB 2100
2040 GOSUB 2500
2050 RETURN
2100 REM - input a line
2110 IN$=""
2120 GOSUB 2200:IF C$=CR$ THEN RETURN
2130 IF C$=BS$ THEN GOSUB 2300
2135 IF C$<" " OR C$>"~" THEN 2120
2140 IF LEN(IN$)>=NC THEN 2120
2150 IN$=IN$+C$:PRINT C$;:GOTO 2120
2200 REM - read one character
2205 ARROW = FALSE
2210 C$=INKEY$:IF C$="" THEN 2210
2220 GOSUB 2250:IF ARROW THEN 2200
2230 RETURN
2250 REM - check for arrow keys
2260 RETURN
2300 REM - backspace
2310 IF LEN(IN$)=0 THEN RETURN
2320 IN$=LEFT$(IN$,LEN(IN$)-1)
2330 X=CSRLIN:Y=POS(0)-1:GOSUB 400
2340 PRINT" ";:GOSUB 400
2350 RETURN
2400 REM - Clear to end of line
2410 PRINT STRING$(NC," ");:RETURN
2500 REM - Parse the command line
2510 CS$="AaEeDdLlSsQqCc"
2515 IF MD<>0 THEN 2540
2520 CMD = INT((INSTR(CS$,LEFT$(IN$,1))+1)/2)
2530 IF CMD<>0 THEN 2590
2540 ON MD GOSUB 5100,5200,5300
2590 RETURN
3000 REM - move up one line

```

Missing Code

When I talked about things popping out here and there in the last issue I had no idea that it was taking me literally.

It turns out that four lines of one of Terry's programs got pushed right off the page and into limbo. And no one caught it, until some of you called to ask what was going on.

The code was from the program List.Bas in Issue 18. There were four lines missing from the end of that program. The four lines are:

```

2120 SP=SP+1:STK(SP)=LI:
      RETURN
2200 Rem-Remove LI from
stack
2210 IF SP=0 THEN LI=0:
      RETURN
2220 LI=STK(SP):SP=SP-1
      RETURN

```

We're just now catching on to the way this new fangled way of publishing works. This time, we checked last line numbers for each program to make sure they were all there.

```

3010 RETURN
3100 REM - move down one line
3110 RETURN
3200 REM - move up one level
3210 RETURN
3300 REM - move down one level
3310 RETURN
3400 REM - set add mode
3410 MD=1:MD$="ADD MODE ":RETURN
3450 REM - set command mode
3460 MD=0:MD$="COMMAND MODE":RETURN
3500 REM - set edit mode
3510 MD=2:MD$="EDIT MODE ":RETURN
3550 REM - initialize new outline
3560 LN$(0) = "Outline Title":LK(0,0)--1:LK(1,0)--1:LK(2,0)--1:
      LV(0)=0:NM(0)=1
3570 RETURN
3600 REM - delete lines
3610 MD=3:MD$="DELETE MODE ":RETURN
4000 REM - Load file
4010 X=LN:Y=1:GOSUB 400
4020 NC=WD:GOSUB 2400:GOSUB 400
4030 PRINT "Filename=> ";
4040 NC = WD-10:GOSUB 2100
4050 IF IN$="" THEN RETURN
4060 FF$ = IN$ + ".OUT"
4070 GOSUB 5400:RETURN
4100 REM - Save file
4110 X=LN:Y=1:GOSUB 400
4120 NC=WD:GOSUB 2400:GOSUB 400
4130 PRINT "Filename=> ";
4140 NC = WD-10:GOSUB 2100
4150 IF IN$="" THEN RETURN
4160 FF$ = IN$ + ".OUT"
4170 GOSUB 5500:RETURN
5100 REM - Add
5110 RETURN
5200 REM - Edit
5210 RETURN
5300 REM - Delete
5310 RETURN
5400 REM - Load
5410 OPEN "I",1,FF$
5420 IN=0:LL(0)=0:LL=0
5430 IF EOF(1) THEN 5495
5440 LINE INPUT#1,IN$
5450 GOSUB 5600

```

```

5460 LN$(IN) = IN$
5470 GOSUB 5700
5480 IN = IN + 1
5490 GOTO 5430
5495 CLOSE:RETURN
5500 REM - Save
5505 OPEN "O",1,FF$
5510 I=0:NM=0:SP=-1
5520 REM - start of loop
5530 GOSUB 6000
5535 IF SP=-1 AND LK(0,NM)=-1 AND I>0 THEN CLOSE:RETURN
5536 I = I + 1
5540 IF LK(2,NM)>=0 THEN GOSUB 1300:NM=LK(2,NM):GOTO 5520
5550 IF LK(0,NM)>=0 THEN NM=LK(0,NM):GOTO 5520
5560 GOSUB 1350:IF NM<0 THEN RETURN
5570 IF LK(0,NM)>=0 THEN NM=LK(0,NM):GOTO 5520
5580 GOTO 5560
5600 REM - determine the line's level
5610 LV(IN) = 0
5620 IF MID$(IN$,1,1)<>" " THEN RETURN
5630 LV(IN) = LV(IN) + 1
5640 IN$ = MID$(IN$,2)
5650 GOTO 5620
5700 REM - link line into structure
5705 IF IN=0 THEN LK(0,0)=-1:LK(1,0)=-1:LK(2,0)=-1:NM(0)=1:GOTO
5900
5710 IF LV(IN) = LL THEN 5800
5720 IF LV(IN) > LL THEN 5850
5730 LK(0,IN) = -1:LK(1,IN) = LL(LV(IN)):LK(2,IN) = -1
5740 LK(0,LL(LV(IN))) = IN:NM(IN) = NM(LL(LV(IN)))+1
5750 GOTO 5900
5800 LK(0,IN) = -1:LK(1,IN) = IN-1:LK(2,IN) = -1
5810 LK(0,IN-1) = IN:NM(IN) = NM(IN-1)+1
5820 GOTO 5900
5850 LK(0,IN) = -1:LK(1,IN) = -1:LK(2,IN) = -1
5860 LK(2,IN-1) = IN:NM(IN) = 1
5870 GOTO 5900
5900 LL(LV(IN)) = IN:LL = LV(IN)
5910 RETURN
6000 REM - save one line to file
6010 PRINT#1, STRING$(LV(NM)*1," ");LN$(NM)
6020 RETURN

```

Random Files

Adding Column Totals to Ranprint.Bas

Last issue we corrected some mistakes in the random indexing program, starting with this issue, we're going to start adding more sophisticated features to the print program (RANPRINT.BAS). Some of the features we're going to be working on over the next few issues include:

- 1) Report totals
- 2) Subtotals on break fields (doing subtotals whenever a field value changes)
- 3) Field math to create fields that aren't in the data base
- 4) Output formatting

and much, much more. I've told Irv that we could carry this series for a long time yet just in terms of important features to have.

I'd like to thank all of you that have been writing to say you're making use of the random files programs. I've been hearing about applications people are building for themselves with the program and it's really special. There's no greater kick than knowing that people are using something you designed.

This issue, we're going to make a pretty simple change to the RANPRINT.BAS program which will allow us to get totals at the bottom of columns where we want them. We'll take a simple approach where we'll create a line, just like the report line with the field numbers we want to total. This will format the line just like it does for the values themselves. It's bare bones at the moment, but it will be enough to start with. Later, we'll introduce formatting and other useful features.

To start the change, we have to make an array to hold the values (TOT#0) which is defined in the new line 30:

```
OLD: 30 DIM FP$(20), SC$(24), XY(20,3)
NEW: 30 DIM FP$(20), SC$(24), XY(20,3),
```

```
TOT#(20)
```

We also add a new line 70 which defines a string (TS\$) which will hold the string total pattern string IF it's defined. If TS\$ is "", then we won't bother to print any totals.

A minor correction (the same as we put into the RANIDX.BAS program in the last issue) changes line 250 like this:

```
OLD: 250 IF FP$(1)="DELETED" THEN 270
NEW: 250 IF INSTR(FP$(1),"DELETED")<>0
THEN 270
```

This makes sure that we take out the deleted records correctly no matter how long the field length of the first field (REMEMBER - the first field MUST be at least 7 characters in length in order to hold the DELETED word. Another, shorter word could be used, just so long as it's unique and you change all programs identically.).

Line 275 is added to do a GOSUB to line 4100 where the data will be printed in a total line. Then line 2140 is changed to allow totaling for each selected record:

```
OLD: 2140 GOSUB 2450
NEW: 2140 GOSUB 2450:GOSUB 4000
```

New line 2965 adds a check for a TOTAL line (starts with a capital T) which will show the pattern for the total printout and lines 3300 through 3330 set the TS\$ and zero the totals.

The actual work is done in subroutines 4000 (total the fields at each selected record) and 4100 (print the totals at the end of the printout). Subroutine 4000 takes the simple approach to totals by simply totaling ALL fields without regard to type. At worst, this should create a lot of zeros and waste some time. We'll get better about this as we start to do more sophisticated

things. That's a subject for later articles.

If you look closely at the routine for laying out a printed line, you'll find that the subroutine at 4100 is an EXACT model of that one with only minor differences. We insert totals where we find field numbers just like inserting record values. The special feature is that if there is NO value of TS\$, we simply ignore this and return. Otherwise our sample line (LN\$) is set to TS\$. When the line is filled, we print it in line 4195.

The listing gives the merge file needed to create this version of RANPRINT.BAS from the original available in Issue 14 (Nov/Dec 87) and on the download.

```
=====
30 DIM FP$(20), SC$(24), XY(20,3),
   TOT$(20)
70 TS$=""
250 IF INSTR(FP$(1),"DELETED") <> 0 THEN
   270
275 GOSUB 4100
2140 GOSUB 2450:GOSUB 4000
2965 IF LEFT$(LN$,1)="T" THEN GOSUB
   3300:RETURN
3300 REM - Total Fields
3310 TS$ = LN$
```

Forum, from page 6

magnetic retentivity, and the ability to write and read perfectly. Data in the areas being tested is removed and temporarily stored in your computer's RAM. If testing reveals an error, Disk Technician+ repairs it by writing a single new track using a factory low-level, real format. This new track is then thoroughly retested, and only if it has been perfectly repaired, will the program allow your programs and data to use it again. If the testing shows that any area is not repairable, your data will be relocated to a good area and the bad area will be safely blocked from future DOS use - all automatically without operator intervention. After Disk Technician+ repairs an area, it specially monitors that spot on all subsequent tests to make certain it stays repaired.

The program is copy protected, and is always booted and run from its original diskette and does not get installed on the hard disk. The

```
3320 FOR ZI=0 TO 20:TOT$(ZI)=0:NEXT ZI
3330 RETURN
4000 REM - Add to field totals
4010 FOR ZI=0 TO 20:TOT$(ZI) = TOT$(ZI)
   + VAL(FP$(ZI)):NEXT ZI
4020 RETURN
4100 REM - Print field totals
4105 IF TS$="" THEN RETURN ELSE LN$=TS$
4110 FS$=LN$
4120 IF INSTR(LN$,"#") <> 0 AND
   LEN(LN$) < WD THEN LN$=LN$+STRING$(WD-
   LEN(LN$)," ")
4130 IF INSTR(LN$,"#")=0 THEN 4195
4140 X = INSTR(LN$,"#")
4150 Y = VAL(MID$(LN$,X+1))
4160 MID$(LN$,X) =
   MID$(STR$(TOT$(Y)),2) + " "
4170 IF X>1 THEN MID$(LN$,X-1)=" "
4180 MID$(FS$,X)=" "
4190 GOTO 4130
4195 PRINT LN$:LC = LC + 1:RETURN
=====
```

No part of this change is very difficult, but it's important to be careful. Next issue, we're going to start going more and more into reporting and working out ways to develop more sophisticated reports for the system. Good luck 'til then.

program can be reset to operate on a new machine or hard disk by calling the factory. The first time you run the program it may take up to four hours to check out a 20 meg hard drive. After that it only takes a few minutes per day to run the program. It keeps a history database on the hard drive and makes comparisons every time you run the program.

It's a lot of program for \$99.00, and it provides you with good insurance.

Thanks again for the great letters. Many of you have remarked that I always note the passing of the seasons. Yes, I do. I think the seasons are great. Where we live there really are only two, wet and dry. I very much enjoy the seasons in the midwest, all four of them. You certainly know which one you are in at any time there.

Actually, if it weren't for the seasons on Earth, I think I'd find another planet on which to live.

Irv

Handy Order Form

| | | |
|---|---------|--|
| RENEW Subscription Nov/Dec 88 through Sep/Oct 89 | \$24.95 | |
| All third year issues, Nov 87 through Sep 88 | \$24.95 | |
| All second year issues, Nov 86 through Sep 87 | \$24.95 | |
| All first year issues, Sep 85 through Sep 86 | \$24.95 | |
| 1st Year Program Disk (issues 1 through 7) (Specify computer type below) | \$20.00 | |
| 2nd Year Program Disk (issues 8 through 13) (Specify computer type below) | \$20.00 | |
| 3rd Year Program Disk (issues 14 through 19) (Specify computer type below) <i>Available after 1 Sep 88</i> | \$20.00 | |
| NEW! "Starting with MS DOS" 40-page book explains all | \$7.00 | |

Postage and handling charges already included in all prices.

We can supply program diskettes for PC/MS DOS, TRSDOS 1.3 (Tandy Model 3), TRSDOS 6.x (Tandy Model IV) and most CP/M MBASIC formats, on 5 1/4 inch diskettes. When ordering diskettes, specify your computer type.

COMPUTER TYPE _____

Check/MO enclosed

Charge to my VISA/MC _____ exp _____

Ship to: Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to:
CodeWorks, 3838 S. Warner St. Tacoma WA 98409

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

VISA/Master Card only, we don't take American Express

988

Index & Download

What's happening with both

Here are the updates to bring Cwindex.Dat up to date through the last issue. The entire index for the first three years of CodeWorks, including this issue, will be on the download and on our yearly diskette.

Blo.bas, correction, issue 18, page 4
Poker.bas, reference, issue 18, page 5
Beginning BASIC, user defined functions, issue 18, page 7
Mediator.bas, main program, issue 18, page 9, mediates disputes
Outline.bas, part 1 of 3, issue 18, page 15
Link.bas, main program, issue 18, page 21, part of Outline.bas
List.bas, main program, issue 18, page 22, part of Outline.bas
Random files, article, fixing Ranidx.bas, issue 18, page 23
Ranidx.bas, main program, issue 18, page 25, randemo indexing
Notes, Model I/III memory addresses, issue 18, page 29
Conversions, article, issue 18, page 30, converting to MS DOS

Hard disks, article, issue 18, page 37, questions and answers

Notes, converting WHILE and WEND, issue 18, page 38

Cwindex.dat, updates to this index, issue 18, page 40

The download, which characteristically has gone down in January and July, waited until August this year. It's down as I write this, having suffered from a mild case of high-voltage arcing. A lot of power switching has been going on in our building, which may or may not account for the problems. In any case, it is at the repair center now, and we expect it to be up and running normally by the time you receive this issue.

Once we get it back up, you can renew your subscription via the download. Just leave your name and charge card number in the comments section and we will take it from there.

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
Postage
PAID
Permit # 774
Tacoma, WA

CODEWORKS

Issue 20

Nov/Dec 1988

CONTENTS

| | |
|-----------------------------------|----|
| <i>Editor's Notes</i> | 2 |
| <i>Forum</i> | 3 |
| <i>Beginning BASIC</i> | 7 |
| <i>Playoff.Bas</i> | 14 |
| <i>Notes</i> | 20 |
| <i>Cword.Bas</i> | 21 |
| <i>Randemo Recap</i> | 28 |
| <i>Split.Bas</i> | 37 |
| <i>Renewal/Order Form</i> | 39 |
| <i>Index & Download</i> | 40 |

Issue 20

Nov/Dec 1988

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

(c)1988 80-Northwest Publishing Inc. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, the publisher assumes no responsibility for errors or omissions. Unless otherwise noted, all programs presented in this publication are placed in public domain. Please address all correspondence to CodeWorks, 3838 South Warner Street, 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. **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, WA.

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.

It's almost Thanksgiving time again. It seems like just yesterday that I was writing Poker.Bas during a rare snowstorm around this time of the year, but it was three years ago!

Have you seen the announcement that Steven Jobs has unveiled his new NeXT computer? From what they showed on the Tee Vee, it looked like it had some pretty impressive graphics. I wonder how long it will take for the industry "power users" to start calling it a "NeXt Box" like they now call "386 Boxes" and the like. Or maybe they'll call it the "Jobs Box."

Microsoft is moving their CD Rom affair from Seattle to California this year. It's been here since it first started. Perhaps Seattle is getting too provincial for the mighty MS.

They have also released MS DOS version 4.0 but don't run out and buy it. Wait for the bugs to get fixed first. I've always been a believer in buying 400 horsepower and only using 200 of it. That way, you get long life, good service and dependability. You can call me a "Half Horse Power User."

There have been many requests to us from readers to recommend an MS DOS machine. We like MS DOS and think it will be around for a while. Even OS-2 didn't make the dent they thought it would. It's hard, though, to recommend specific machines. There are a lot of them out there at fantastic prices. Some 286 machines (read PC AT) with hard disks are going for around \$1500.00 and that has to be a buy. We would recommend staying away from off-shore odd-balls however. Who's going to fix it

when it breaks? Is it really compatible? You never know until it's too late. Names? Well, there's Compaq, AT&T, AST, and if you're not a red-neck about the name, Tandy has some very good machines at fair prices. 'Course, you can always be true blue and go for IBM. Check prices, service and compatibility and you won't go wrong.

Speaking of MS DOS, it's interesting to see how many of you who don't have it are buying our booklet on getting started in it. But I guess that makes sense, since you want to see what you may be in for. It's getting good comments from most of you. One person though, called in and said it had a "gross" mistake in it. Seems he took issue with the statement that you can have 115 files in a directory on a hard disk. He says you can have as many as you want in the sub-directory but only 115 in the root directory. Picky, what?

And speaking of books, guess who is the author of a new one put out by Que, called "DOS Programmer's Reference?" Well, no one other than our own Terry Dettmann. It's a monster book, over 800 pages, and technical reading cover to cover. Now, don't get the wrong idea. That's not why he missed the deadline for this issue! The book has been out for some time now, and should be on your local bookstore shelf. Take a look.

Thank you all for your early renewals. It sure helps when we don't have to keep harping on it. We appreciate it.

Irv

Forum

An Open Forum for Questions and Comments

I have enjoyed trying (the program Correl.Bas in Issue 19). The leading/lagging correlation feature is very nice.

You may wish to point out to your readers that the program's accuracy is somewhat limited in calculating the correlation coefficient. I'm using GW BASIC under MS-DOS 3.2. I get accurate results when the values in the input files have no more than two significant figures. There can be a slight error when these values have three significant figures. On a test set of files whose values contain four significant figures, the program gives a correlation coefficient of 0.62 versus the correct value of 0.7853 (determined by using a 10-digit hand calculator)...

...The sources of error appear to be (1) the use of single precision variables in the calculations, and (2) the use of the exponentiation operator (^) to square values instead of multiplication of the value by itself.

In GW BASIC exponentiation appears to return a single precision value even if the variable is double precision. For example:

```
10 DEFDBL A
20 A=98.18
30 PRINT "A=";A
40 PRINT "A*A=";A*A
50 PRINT "A^2=";A^2
Run
A = 98.18000030517578
A*A = 9639.312459924317
A^2 = 9639.312
```

The correct value is 9639.3124. The error in the fourth decimal place may seem trivial, but when differences between squared terms used (such as in program line 1580), one can wind up with only three or four significant figures for subsequent calculations. Thus, the round-off error can be appreciable as in the first example above.

When I added line 142 DEFDBL A,B,C,D,E,F,N and changed the following lines to use multiplication instead of exponentiation,

the revised program gives the correct value for the coefficient of 0.78527 which is rounded to 0.79 by line 1610 and/or line 1090.

```
1030 D=D+(A(I)*A(I))
1040 E=E+(B(I)*B(I))
1200 D=D+(A(I+S)*A(I+S))
1210 E=E+(B(I)*B(I))
1420 D=D+(A(I)*A(I))
1430 E=E+(B(I+T)*B(I+T))
1580 AA=((N3*D)-(B*B))*((N3*E)-(C*C))
```

A better alternative for GW BASIC is to call BASIC with the /D option. This loads BASIC with the double precision transcendental math package and provides double precision values for exponentiation and square root functions. I don't know if other versions of BASIC operate in the same manner.

One further program note: Line 1610 rounds CC to two decimal places. This line seems redundant in view of the Print Using format ##.## which is found in lines 1090, 1270, 1300, 1490 and 1520. The program seems to work fine with line 1610 deleted...

Robert L. Anderson
St. Albans, WV

You are right. Although we knew about the accuracy problem, we simply overlooked it when we wrote the program.

As an experienced BASIC programmer I recently purchased Microsoft's QuickBASIC and fell in love with it the first time I used it. While many typical, small BASIC programs do not really need the speed up from compiling, there are several other major benefits, such as the fine BASIC editor, the pull-down menus, the dialog boxes, and most important, its powerful debugging features. BASICs and GW BASICs editors are primitive by comparison.

How about some articles and examples of the QuickBASIC programs. I am sure many others would appreciate it. Keep up the good work.

Paul G. Delman

Ft. Lauderdale, FL

Although half our readers don't have the machine to run QuickBASIC, we have been thinking about running a regular column on it for those who do. Since we are not experts at it either, it would be a case of learning from scratch, together.

Issue 19 has just arrived, and I agree that switching to desktop publishing has improved the appearance of your magazine; but, unless my copy was an exception, the collating attachment on your photocopier needs adjustment... (It was missing pages, ED.)

Although I still am devoting all of my free time to compiling a family history and genealogy, and my interest in learning to program in BASIC remains "on hold" for the immediate future, I do want my CodeWorks collection to be complete when I get around to it...

Russell Bond
Buffalo, NY

Yours seems to be a common complaint for that issue. Several others have made remarks about desktop publishing and the missing pages problem. The two have absolutely nothing to do with each other. Desktop publishing only produces the camera-ready page masters for the magazine. Those master pages are then photographed, printing plates are made and the plates go onto the printing presses to produce the magazine. The pages must then be collated and sent to the bindery for stitching and trimming. We don't own the presses or the bindery, we send that work out to commercial printers. The problem with Issue 19 was in the bindery, where it seems, things got a little bit out of sync. We will certainly pay more attention to it this issue. And darn, isn't it always like that? Seems to be one thing or another that keeps you from putting out a "perfect" issue - but it's fun trying.

Will Randemo be down-loadable at some future time as a complete file rather than "pieces?"

F. V. Bruch
Troy, NY

It will probably always be in three major pieces. You can tie them all together, however, with a small menu program, as we did with Card.Bas.

Do you have a program in BASIC that I could

run on my Model IV to keep track of payments on a Trust Deed? The payments are interest first, but any amount after that will be applied to the principal. The amount of the payments may vary, but the due date is firm, with a balloon payment at that time. Thanks for your time in this matter.

Richard L. Wright
Buena Park, CA

We do now. It's called Trust.Bas and should appear in these pages within the next few issues. (Actually, we have already sent Mr. Wright an advance copy of the program for check out and suggestions for improvements. - ED)

Is there anything in the future to upgrade "NFL Oracle" where you can enter the play-off games and Superbowl? It is a great predictor and I am very pleased with your magazine. Keep it up. I am over 65 and have learned most of my hobby computing through CodeWorks. It sure helps a novice like me. You should get a super gold star for such detailed explanations.

Ivor J. Fosmo
Sparks, NV

Thanks for the nice comments, and you will find the play-off program in this very issue. It's called Playoff.Bas.

...CodeWorks is almost a "cult" piece, notable for its erudition and clarity. Thus, it is not at all difficult for me to see the jumble in which I sometimes write...

R. J. Richardson
Valencia, CA

And write he did. His program, Budget.Bas, will be appearing in a future issue.

...I have recently moved up from a Model III to a Model 1000 TX and when (Issue 18) arrived I immediately saw listed on the cover a thing called "Conversions, page 30." I raced into the house and in my excitement, nearly tore the cover off. There on pages 34 and 35 I found some "missing Model III commands." Although they are very welcome, I was nearly heartsick to find my beloved CMD"X" is still missing. There are lots of things missing from my dear old "programmer's dream" but that CMD"X" is the one I miss the very most. Can you please help? I know that the Model 1000 does have the "FIND" but that seems to work only for text files, and what

I am primarily looking for is the line numbers of those lines in data statements and in files other than text files which contain my search string. In desperation I once renamed a sequential file, giving it a .TXT extension, but alas, it did not fool my TX at all

I also noticed in Issue 18 a letter from a reader heartily endorsing the genealogy program, CLAN, written by Arthur C. Hurlburt of Davenport, Iowa. I have had his program for almost four years now and have had one book published, using that program. I have over 850 family names in that one with room for more. I am currently working on another book and have over 600 names in it! It is indeed, a wonderful program, I have seen others but I like this one the best...

**Betty Berg
Milan, IL**

The MS DOS FIND command works just fine if you save your programs in ASCII format, i.e., SAVE "filename",A.

...As usual, the articles in Issue 18 are both informative and thought-provoking. I would like to add a few comments to the article by Al Mashburn regarding hard disks. I currently have two hard disks in operation, one on an XT clone (30 meg) and one on a Tandy Model IV (5 meg). In answering the question of recommended disk size, he implies that only the TRSDOS operating system has a "problem" with chunk size. In fact, MS DOS has an identical problem. Using MS DOS 3.1 to format a 20 meg hard disk will give a cluster (chunk) size of 8,192 bytes minimum. This was improved in MS DOS 3.2 to a size of 2,048 bytes. Thus, a 100 byte file under MSDOS will occupy either 8,192 or 2,048 bytes of disk space. Confusion comes from the use of the DIR command, which lists file length for MS DOS and disk usage in TRSDOS. When you issue the DIR command in TRSDOS, the 100 byte file will show up as a 4K file (5 meg disk partitioned into four equal logical drives). Issuing the DIR command from MS DOS will show the file length as 100 bytes, but if you compare the free space on the disk before and after writing the file, you will find that 8,192 or 2,048 (depending on DOS version) have been used. The information about division of the drive into logical drives also needs to be expanded. While

you can break one physical drive into four logical drives, it is not required. You may divide it into two or three or use it as a single logical drive. One arrangement is to create on or two drives of 1.25 meg, with the remainder (3.75 meg or 2.5 meg) used as a single drive for large data files (at one time I had a 1.6 meg inventory file and a 1.2 meg sales files sharing the 3.7 meg second logical drive). He is correct in stating that the drivers supplied by Radio Shack do not support sub-directories and paths, however, the DiskDisk utility available from Misosys will allow a similar type of file organization. In either case, the proper use of file library and archive utilities will greatly enhance the utility of your hard disk. I plan to expand the Model IV hard disk to 20 meg as soon as possible, and feel confident that I won't waste any of it. With used 5 meg drives currently in the \$300 range, no Model IV user should be without one if he can afford it. Note: TRSDOS in the above applies to TRSDOS 6.2 and LS-DOS 6.3.

I hope the above information is useful. Keep up the good work.

**Tom Biggar
Fairview, TN**

...I have a Model III, two Model IV's and now an AT&T 6300 Plus with Unix. There is so much to learn about I may never get the roof on the house reshingled...

**R. C. Chittenden
Amarillo, TX**

If you get into that Unix, you will be setting out drip-buckets for sure.

Qkey.Bas, Issue 10, is a great program! I have a problem making corrections to the data. How do I change typing errors after ENTER?

**R. H. Saunders
Epping, NH**

To correct or delete a line with Qkey, you use the .EDIT command. Let's say you wanted to find something and forgot to put the period before FIND and then pressed ENTER. If you then .EDIT FIND it will find the word "find" and then you can delete it or replace it with something else. Unfortunately, the program is not sophisticated enough to edit within a line, you must retype the entire line to change it.

In Issue 18 you published a letter from me and a letter from Mr. Lawrence J. Carley of Mt. Morris, Michigan.

Mr. Carley's letter advised that a good genealogy program could be purchased from Arthur C. Hurlburt and I immediately purchased this program and find it to be all that he said that it was and I am very well pleased with it. It is easy to run and gives out wonderful reports. Mr. Hurlburt says that he has over 700 names in his program on one disk. I have entered over 400 names on my disk and have room for many more. Mr. Hurlburt is a real nice person to deal with and this is the best \$10 that I have ever spent on a program. The program is for a Tandy Model III.

I want to thank you for taking an interest in my letter and publishing the letter that allowed me to get the genealogy program from Mr. Hurlburt. I also extend my appreciation to Mr. Carley for writing to you about this program...

Walter Evans, Jr.
Waco, TX

And the interest in genealogy goes on and on. We were happy to be of assistance and act as a clearinghouse.

...The MSDOS book is neat. All MSDOS users should buy one and keep it handy for reference before attacking the intimidating, thick, sometimes too technical and unreadable, sometimes incomplete, DOS manual.

I hope Al Mashburn will do more writing for CodeWorks. His work on "Shareware" (Issue 5) and hard disk are welcome. I would like to see more on the subject for hard disks, perhaps the various formats (MFM and RLL and others), causes of a crash, need for a disciplined backup program, good habit of running SHIP with each time you turn off power, etc.

I have noticed that CodeWorks has grown, top to bottom, and left to right. All issues up to 15 were the same size, and all after have been different. The cause?

John R. Miller
Anderson, SC

Three different printers, so far, have printed and bound the magazine. They all insist on doing it their way. We too, would like to see it be a little more consistent.

Thanks once again for the interesting letters and comments. Enjoy the football and the Thanksgiving turkey, and the new administration (whoever it turns out to be), and we'll see you again around the first of the year.

Irv

NOTICE
Due to unavoidable
circumstances
Terry's Outline program
and his
Randemo article will
NOT APPEAR
in this issue.

We will, however, pick them
up again in the next issue.



"Don't worry. In fraudulent tax cases
your computer can't testify against you."

Beginning BASIC

Error Messages for Beginners

If you are new to BASIC, you probably see a lot of error codes and cryptic messages when you program. Wouldn't it be nice if the error message told you a little more about what the error was than "TM error in 300" or some such? And wouldn't it help if you had some idea of where the error line was?

Errmsg.Bas is a program designed to serve as a crutch until you become more familiar with the error codes of your computer's BASIC. It's not a program that will run by itself, but is a program you can merge into your programming during checkout and debugging time. It will then give you a full description of the errors encountered, and will display the line of code where the error has probably occurred.

The full treatment of error trapping is a subject by itself, and is beyond the scope of Beginning BASIC. It will be treated separately in a later article either in this issue or in an upcoming one.

How to use Errmsg.Bas

Type in the program (Errmsg.Bas) or get it from our download. Be sure you have saved it in ASCII (SAVE "ERRMSG.BAS,A"). When you type in a program from a magazine, or write your own from scratch, just go ahead and write as you normally would. You can use line numbers in your program from 11 through 29998. Line 10 and the numbers from 29999 on will be used when you merge Errmsg.Bas. When you get ready to check out your program, load your program and then at the ready prompt type: MERGE"ERRMSG.BAS"

Now, when you run your program, any errors will be displayed with a full description of the error, as well as the line (not just the line number) where the error probably occurred. If the error was a syntax error the line containing

the error will be displayed and you will automatically be in the EDIT mode. All other errors will show the line and give you the BASIC ready prompt. The error messages you now receive will give you enough information so that you can probably fix the error and continue checking out your program.

If the program you are copying from an article contains a line that starts out with "ON ERROR GOTO...", then temporarily remark that line while you use Errmsg.Bas. Don't forget to activate that line again later.

When you are satisfied that your program is checked out, you can delete line 10 and from 29999 to 31130. At the ready prompt, simply type the number 10 and press enter, then type: Delete 29999-31130 and press enter. That will get rid of Errmsg.Bas and leave your program ready to go.

How Errmsg.Bas works

When BASIC encounters an error, it automatically sets two internal variables, ERR and ERL. ERR is the error number and ERL is the line number where the error was detected. In fact, when you get an error, you can tell BASIC to print ERR and ERL and see what they are.

Line 10 of Errmsg.Bas sets the "error trap" so that on any error program flow will jump to line 30000. The END statement in line 29999 is just there to keep your program from crashing into this part of the program. In line 30000 we let X equal the error number that BASIC found.

In lines 30010 through 30055 we use the value of the error number to go to the appropriate lines to display the correct information about the error in question. If the error number encountered was 3 then line 30010 will send us to line 30035, and the third number in the ON X

GOTO line 30035 is 30130. If we go to line 30130, we find that it is the "Return without GOSUB error." The program will print this error and the following lines on the screen for you. Notice line 30145. It ends with a GOTO 31125. Line 31125 lists the line in question on the screen, right under the error message that is already there. Now you have the error message

in full, and the line that probably caused the error displayed right there on the screen. All the other error numbers work the same way. The only exception to this is the syntax error, which automatically puts you into the EDIT mode anyway, so way up in line 30120 we go to line 31130 instead of 31125.

**Errmsg.Bas written for GW BASIC
see note for other machines.**

```
10 ON ERROR GOTO 30000
29999 END
30000 X=ERR
30005 CLS
30010 IF X=<10 THEN 30035
30015 IF X=<20 THEN X=X-10:GOTO 30040
30020 IF X=<30 THEN X=X-20:GOTO 30045
30025 IF X=<60 THEN X=X-49:GOTO 30050
30030 IF X=<71 THEN X=X-60:GOTO 30055
30035 ON X GOTO 30065,30100,30130,30155,30185,30240,30260,30285,
    30300,30330
30040 ON X GOTO 30355,30370,30390,30425,30445,30465,30485,30510,
    30530,30550
30045 ON X GOTO 30570,30590,30610,30625,30645,30660,30685,30570,
    30700,30720
30050 ON X GOTO 30740,30760,30785,30810,30830,30860,30570,30885,
    30905,30570,30570
30055 ON X GOTO 30925,30940,30970,30990,30570,31015,31035,31055,
    31075,31090,31110
30060 ` error 1
30065 PRINT``NEXT without FOR ERROR``
30070 PRINT``BASIC executed a NEXT statement without previously
30075 PRINT``executing a FOR statement, or a variable in a NEXT
30080 PRINT``statement does not correspond to a previously
30085 PRINT``executed FOR statement. The error may or may not be in
30090 PRINT``the line being displayed.``:GOTO 31125
30095 ` error 2
30100 PRINT``Syntax ERROR
30105 PRINT``BASIC encountered a line that contains an incorrect
30110 PRINT``sequence of characters (such as unmatched parentheses,
30115 PRINT``misspelled statement, incorrect punctuation, spacing,
    etc.)
30120 PRINT``The error is in the line being displayed.``:GOTO 31130
```



```
30125 ` error 3
30130 PRINT``Return without GOSUB ERROR
30135 PRINT``BASIC executed a RETURN statement without previously
30140 PRINT``executing a GOSUB statement. The error is NOT
        necessarily in
30145 PRINT``the line being displayed.'':GOTO 31125
30150 ` error 4
30155 PRINT``Out of data ERROR
30160 PRINT``While executing a READ statement, BASIC could not find
30165 PRINT``any DATA statements or un-read data items. This error
30170 PRINT``shows the line that READS as the error line, but the
        error
30175 PRINT``is probably in the DATA line or lines.'':GOTO 31125
30180 ` error 5
30185 PRINT``Illegal function call ERROR
30190 PRINT``A parameter that is out of range was passed to a math
        or
30195 PRINT``string function. This error may also be caused by a
30200 PRINT``negative array subscript or an unreasonably large
30205 PRINT``subscript, a negative or zero argument with LOG, a
30210 PRINT``negative argument with SQR, a negative mantissa with a
30215 PRINT``noninteger exponent, an invalid exponential number, an
30220 PRINT``improper argument to MID$, LEFT$, RIGHT$, etc., or a
30225 PRINT``negative record number with GET or PUT.'':GOTO 31125
30230 ` error 6
30235 PRINT``Overflow ERROR
30240 PRINT``The result of a calculation was too large to be
30245 PRINT``represented in BASIC numeric format.'':GOTO 31125
30250 ` error 7
30255 PRINT``Out of Memory ERROR
30260 PRINT``A program is too large, has too many FOR loops or
30265 PRINT``GOSUBs, has too many variables, or has expressions
30270 PRINT``that are too complicated to untangle.'':GOTO 31125
30275 ` error 8
30280 PRINT``Undefined line number ERROR
30285 PRINT``A nonexistent line was referenced in a GOTO, GOSUB,
30290 PRINT``IF..THEN..ELSE, or DELETE statement.'':GOTO 31125
30295 ` error 9
30300 PRINT``Subscript out of range ERROR
30305 PRINT``An array element is referenced with a subscript out-
30310 PRINT``side the dimensions of the array or with the wrong
30315 PRINT``number of subscripts. Try printing the value of the
30320 PRINT``variable contained in the subscript for clues.'':GOTO
        31125
30325 ` error 10
30330 PRINT``Redimensioned array ERROR
30335 PRINT``BASIC encountered two DIM statements for the same array,
30340 PRINT``or a DIM statement after the default dimension of 10
30345 PRINT``had already been established for that array.'':GOTO
```

```
31125
30350 ` error 11
30355 PRINT''Division by zero ERROR
30360 PRINT''You simply cannot divide by zero. No one can.'':GOTO
31125
30365 ` error 12
30370 PRINT''Illegal direct ERROR
30375 PRINT''A statement that is illegal as a command was
encountered
30380 PRINT''at BASICs prompt.'':GOTO 31125
30385 ` error 13
30390 PRINT''Type mismatch ERROR
30395 PRINT''A string variable name was assigned a numeric value or
30400 PRINT''the other way around. A string function was given a
30405 PRINT''numeric argument or the other way around. You cannot
30410 PRINT''mix strings and integers without converting them
30415 PRINT''first.'':GOTO 31125
30420 ` error 14
30425 PRINT''Out of string space ERROR
30430 PRINT''The amount of memory used by string variables exceeded
30435 PRINT''the amount of free memory.'':GOTO 31125
30440 ` error 15
30445 PRINT''String too long ERROR
30450 PRINT''An attempt was made to create a string more than 255
30455 PRINT''characters long.'':GOTO 31125
30460 ` error 16
30465 PRINT''String formula too complex ERROR
30470 PRINT''The string expression is too long or too complex. The
30475 PRINT''expression should be broken into smaller expressions.'':
GOTO 31125
30480 ` error 17
30485 PRINT''Can't Continue ERROR
30490 PRINT''An attempt was made to continue a program that halted
30495 PRINT''because of an error, was modified during a break in
30500 PRINT''execution or does not exist.'':GOTO 31125
30505 ` error 18
30510 PRINT''Undefined user function ERROR
30515 PRINT''A USR function was called before providing a function
30520 PRINT''definition (DEF USR statement).':GOTO 31125
30525 ` error 19
30530 PRINT''No RESUME ERROR
30535 PRINT''BASIC executed an error-handling routine that did not
30540 PRINT''have a RESUME statement.'':GOTO 31125
30545 ` error 20
30550 PRINT''RESUME without ERROR
30555 PRINT''BASIC executed a RESUME statement when no error
30560 PRINT''had occurred.'':GOTO 31125
30565 ` error 21
30570 PRINT''Unprintable ERROR
```

```

30575 PRINT "An error message is not available for the error
30580 PRINT "that occurred.":GOTO 31125
30585 ` error 22
30590 PRINT "Missing operand ERROR
30595 PRINT "Basic encountered an expression that contained an
30600 PRINT "operator but no operand.":GOTO 31125
30605 ` error 23
30610 PRINT "Line buffer overflow ERROR
30615 PRINT "The line being input is too long.":GOTO 31125
30620 ` error 24
30625 PRINT "Device timeout ERROR
30630 PRINT "Basic did not receive information from an I/O device
30635 PRINT "within a predetermined amount of time.":GOTO 31125
30640 ` error 25
30645 PRINT "Device fault ERROR
30650 PRINT "An incorrect device designation has been entered.":
      GOTO 31125
30655 ` error 26
30660 PRINT "FOR without NEXT ERROR
30665 PRINT "BASIC executed a FOR statement that did not have a
30670 PRINT "matching NEXT. Also check for FOR J= with NEXT I,
30675 PRINT "for example.":GOTO 31125
30680 ` error 27
30685 PRINT "Out of paper ERROR
30690 PRINT "Basic received an out of paper status from the
      printer.":GOTO 31125
30695 ` error 29
30700 PRINT "WHILE without WEND ERROR
30705 PRINT "Basic encountered a WHILE statement that did not have
      a
30710 PRINT "matching WEND.":GOTO 31125
30715 ` error 30
30720 PRINT "WEND without WHILE ERROR
30725 PRINT "Basic executed a WEND statement before executing a
30730 PRINT "WHILE statement.":GOTO 31125
30735 ` error 50
30740 PRINT "Field overflow ERROR
30745 PRINT "A FIELD statement is allocating more bytes than the
30750 PRINT "specified record length of the direct access file.":
      GOTO 31125
30755 ` error 51
30760 PRINT "Internal ERROR
30765 PRINT "An internal malfunction has occurred in BASIC. There
      isn't
30770 PRINT "one heck of a lot you can do about it. BASICs internal
      stack
30775 PRINT "is probably garbled. You lose. Re-load BASIC.":GOTO
      31125
30780 ` error 52
30785 PRINT "Bad file number ERROR

```

```

30790 PRINT''BASIC has encountered a reference to a buffer number
      that
30795 PRINT''is not open or is out of the range of the number of
      files
30800 PRINT''specified when BASIC was first loaded.'':GOTO 31125
30805 ` error 53
30810 PRINT''File not found ERROR
30815 PRINT''A LOAD, KILL, or OPEN statement referenced a file that
30820 PRINT''does not exist on the current disk.'':GOTO 31125
30825 ` error 54
30830 PRINT''Bad file mode ERROR
30835 PRINT''An attempt has been made to use PUT, GET or LOF with a
30840 PRINT''sequential file, to LOAD a direct file, or to execute
30845 PRINT''an OPEN statement with a file mode other than I,O,R,
30850 PRINT''E or D.'':GOTO 31125
30855 ` error 55
30860 PRINT''File already open ERROR
30865 PRINT''BASIC encountered an OPEN statement for sequential
30870 PRINT''output, or a KILL statement, for a file that is
30875 PRINT''already open.'':GOTO 31125
30880 ` error 57
30885 PRINT''Device I/O ERROR
30890 PRINT''An input/output error occurred. This is a fatal error
30895 PRINT''and the operating system cannot recover from it.'':GOTO
      31125
30900 ` error 58
30905 PRINT''File already exists ERROR
30910 PRINT''The filename specified in a NAME statement is
      identical
30915 PRINT''to a file specification in use on the disk.'':GOTO
      31125
30920 ` error 61
30925 PRINT''Disk full ERROR
30930 PRINT''All of the diskette space is already in use.'':GOTO
      31125
30935 ` error 62
30940 PRINT''Input past end ERROR
30945 PRINT''BASIC executed an input statement after all the data
      in
30950 PRINT''the file has already been read, or BASIC executed an
30955 PRINT''input statement to a null (empty) file. To avoid this
30960 PRINT''error, use the EOF function to detect end of file.'':
      GOTO 31125
30965 ` error 63
30970 PRINT''Bad record number ERROR
30975 PRINT''In a GET or PUT statement, the record number is either
30980 PRINT''greater than the max allowed, or is equal to zero.'':
      GOTO 31125
30985 ` error 64
30990 PRINT''Bad file name ERROR

```

```

30995 PRINT''An illegal name was used with a LOAD, SAVE, KILL or
      OPEN
31000 PRINT''statement, for example, a file name with illegal
31005 PRINT''characters in it.'':GOTO 31125
31010 ` error 66
31015 PRINT''Direct statement in file ERROR
31020 PRINT''Information in a non-ASCII format was encountered
      while
31025 PRINT''loading an ASCII format file. The load is terminated.'':
      GOTO 31125
31030 ` error 67
31035 PRINT''Too many files ERROR
31040 PRINT''The disk already contains the max number of files
      allowed.
31045 PRINT''It can also occur with a double extension
      (NAME.DAT.BAS)'':GOTO 31125
31050 ` error 68
31055 PRINT''Device unavailable ERROR
31060 PRINT''An attempt was made to open a file to a non-existent
31065 PRINT''device, or when a device has been disabled.'':GOTO
      31125
31070 ` error 69
31075 PRINT''Communication buffer overflow ERROR
31080 PRINT''Not enough space has been reserved for the comm
      buffer.'':GOTO 31125
31085 ` error 70
31090 PRINT''Disk write protected ERROR
31095 PRINT''Occurs when an attempt is made to write to a diskette
      that
31100 PRINT''is write protected with a write-protect tab.'':GOTO
      31125
31105 ` error 71
31110 PRINT''Disk not ready ERROR
31115 PRINT''Occurs when the drive door is open or there is no
      diskette
31120 PRINT''in the drive.'':GOTO 31125
31125 LIST .
31130 END

```

Note: Some machines will not call all the errors listed in the program because they don't apply. This is especially true for BASICs prior to version 5.0. Also, those with BASIC prior to 5.0 should change line 30000 to $X=(ERR/2)+1$ (Tandy Models I & III particularly will need to do this. Don't forget the space and period in line 31125, it's what lists the offending line for you.

Playoff.Bas

Oracle takes on the post-season play

Staff Project. Here is an additional program that will work with the Stat.Dat data file from NFL88.Bas to help predict the outcome of the post-season games and the Super Bowl. Let's hope it does as well this year as it did on last year's games during checkout.

Well, here it is, football fans: A playoff/Super-Bowl prediction program. You have been asking for this since we first published NFL86.Bas back in 1986. Our thinking on the subject must have been a bit prejudiced because we kept thinking in terms of the original NFL program and worried a lot about how to get the playoff schedule into the program. But after the umpteenth such request, we finally sat back and asked ourselves why not?

The program, of course, had to be based on the season standings. Therefore it would have to use Stat88.Dat as a source of input. One problem came up here. Stat88.Bas only allowed for 15 weeks of data because we projected the 16th week only and didn't need the week 16 stats. This program should use the stats from week 16 as well. So there are two changes to make in Stat88.Bas: Change the DIM in line 190 from 420 to 448 and in line 350, change WN=420 to WN=448. This will allow you to enter the week 16 statistics with Stat88.Bas. Our new program, Playoff.Bas, will use the last four weeks of the regular season as a base upon which to project the playoff winners.

The other mental block we had to overcome was the scheduling of the playoff games. There is no way to know that in advance. So we simply let you enter which teams are playing and let it go at that. This has another advantage in that you can play "what if?" and see what would have happened if Team X were playing Team Y in the playoffs.

Our playoff program works slightly differently than NFL88.Bas does. It uses a different algorithm to calculate the winner, for one thing. For another, it gives you probable scores for the games. It still gives the home team a two point advantage, except when you predict the Super-Bowl, then it removes the home team advantage. You should use a separate "run" of the program when trying to project the Superbowl winner.

The way we figure the winner is not as complicated as in NFL88.Bas, but by trial and error, we found that the defense of the opposing team had to be figured in more prominently (by using points allowed.) This makes the predicted scores rather interesting in that the same team pitted against several other teams will result in different scores for each, depending on the points allowed by the opposing team. In other words, Team X will show different scores when matched against Team Y and Team Z because the defense of Teams Y and Z figures into the score for Team X.

We tried many variations on the calculation routine using last year's data and playoffs. The one that finally worked the best was to look at the last four games of the season. The net result was that it picked seven of nine of last year's playoffs. The two games it missed were the maverick Vikings, who didn't have the stats going in, but upset both the Saints and the 49'ers. And if you could predict an upset it wouldn't be an upset, would it? See figure 1 for how the predictions actually went.

Since the playoff teams are usually matched rather well, we had a problem coming up with tie scores. We use two tie-breakers. The first is the number of first downs, then if there is still a tie score, we use the won/lost record of the teams. This may or may not be valid, but is the best we could come up with, given the stats that we have. It is conceivable that we could still come up with a tie, which we may as well use to signify a game too close to call.

Just because it worked rather well on last year's games is no guarantee that it will perform this year. Given the statistics, however, it should do just about as well - we hope. It will be interesting to see how it stands up.

The Program

The program starts off with code that looks suspiciously like that in Stat88.Bas. This is because we borrowed most of that starting code from that program. In line 160 we dimension the double A array to hold the stats we will be reading in (including week 16 stats). Then we dimension the T\$ array to hold the team names.

The code from 180 to 230 is our familiar locate/print@ subroutine, which you all know about by now. Un-remark the appropriate line for your computer and remark the rest. The team names are held in data statements in lines 260 through 290.

In lines 320 through 410 we read in the existing statistics file and put them into the A(I,J) array, and in lines 440 to 460 we read the data statements into the T\$ array to get the team names.

Lines 480 through 630 print the CodeWorks heading on the screen and prompt for user input as regards printer output, Super Bowl prediction (yes or no) and checks to make sure that the stat file was updated properly.

Something a little different from Stat88.Bas happens from here on. In lines 690 through 750 we print the teams and their team numbers on the screen. They will stay there, at the top of the

screen for the rest of the program. This is necessary so that you can pick the teams by number for the playoff projections. We don't have a schedule for the playoff games, remember? So you need something easy to pick from. Notice the little mechanism we use to get the team number attached to the proper team in line 710. The loop counters alone wouldn't have done this for us. Also note the comma at the end of line 720, used to space the four columns of teams on the screen.

Next we use our locate/print@ scheme to provide for a "dialogue" area just below the teams on the screen. Here (lines 780 through 820) we prompt for the number of the visiting and home teams or zero to terminate the program. The visiting team becomes X1 and the home team X2.

The loop from 850 to 910 reads the statistics file and collects the information we will need. B(1) through B(4) holds information on the visiting team, while C(1) through C(4) holds the same information for the home team. Lines 860 and 870 collect information on won/loss record for the entire season. In line 880, if the week number is less than 13, we simply go on through the file. This results in the following data being collected only for weeks 13 through 16, which is what we want. In lines 890 and 900 we accumulate the data for both teams for weeks 13 through 16.

The calculation of who has the power is made in lines 940 through 1000. Here, we find the average for the values we just accumulated (in lines 940 and 950.) In lines 960 and 970 we take the points scored for the team times two and add the points allowed by the other team. In line 970, if we are not predicting the Super Bowl game, we add two to the home team. In line 980 we add first downs if the two teams are already equal. If they are not equal, we ignore the first downs. In line 990 we take the two values for the two teams and divide them by three and take the integer. This is to get the scores down to something that is more in line with actual scores. After this (in line 1000) if the two scores are still equal, we add the won/loss record to each team.

Lines 1030 to 1090 print the results on the screen and if the printer was selected earlier, on the printer as well. Next, we clear the accumulating arrays in preparation to finding the scores for another set of teams (lines 1120 to 1150.) Then, we clear the input area on the screen (lines 1180 to 1210) and loop back (line 1240) for another pair of teams.

As you can see, you can play any team against

any other team with this setup. Of course, it's water under the bridge, but if your favorite team didn't make it into the playoff schedule, you can at least see how they might have done. Don't forget that to predict the Super Bowl, you should do a separate run for it, otherwise there will be no home team advantage for your other games.

There it is. If it does as well this year as it did in the season just passed, it will please us no end - and we're sure, you too.

| | Predicted | Actual | |
|----------|-----------|--------|--------|
| Seahawks | 21 | 20 | |
| Oilers | 23 | 23 | |
| Vikings | 20 | 44 | |
| Saints | 31 | 10 | missed |
| Broncos | 22 | 34 | |
| Oilers | 20 | 10 | |
| Colts | 14 | 21 | |
| Browns | 19 | 38 | |
| Redskins | 26 | 21 | |
| Bears | 17 | 17 | |
| Vikings | 14 | 36 | |
| Niners | 32 | 24 | missed |
| Broncos | 21 | 38 | |
| Browns | 20 | 33 | |
| Redskins | 24 | 17 | |
| Vikings | 20 | 10 | |
| Redskins | 34 | 42 | |
| Broncos | 33 | 10 | |

Here is how Playoff.Bas predicted last year's post-season games and the Super-Bowl. In both cases where it missed, it was the Vikings upsetting the Saints and the Niners. Some of the point spreads are amiss, but others are close. Anyway, seven out of nine isn't bad.

Figure 1

**Playoff.Bas for MS DOS machines. Changes
for other machines follow this listing.**

```
100 REM * PLAYOFF.BAS * CODEWORKS MAGAZINE * 3838 S. WARNER ST.  
    TACOMA WA.  
110 REM * 98409 (206) 475-2219 VOICE (206) 475-2356 300/1200 MODEM  
120 REM * Projects winners and scores in NFL playoff games  
130 `  
140 PRINT "Loading STAT.DAT data file .."  
150 ` CLEAR 10000: ` Use only if your machine needs to clear  
    string space.  
160 DIM A(448,5),T$(28)  
170 `  
180 REM * General purpose locate/print@ subroutine  
190 GOTO 260  
200 LOCATE X,Y:RETURN `GW-BASIC  
210 `PRINT@((X-1)*64)+(Y-1),,:RETURN `Tandy I/III  
220 `PRINT@((X-1),(Y-1)),,:RETURN `Tandy IV  
230 `PRINT CHR$(27)+"Y"+CHR$(31+X)+CHR$(31+Y);:RETURN ` CP/M  
240 `  
250 REM * Set up the team names in data lines  
260 DATA Redskins,Cowboys,Eagles,Giants,Cards,Bears,Vikings  
270 DATA Packers,Lions,Bucs,Niners,Rams,Saints,Falcons  
280 DATA Dolphins,Patriots,Jets,Bills,Colts,Steelers,Browns  
290 DATA Bengals,Oilers,Seahawks,Raiders,Broncos,Chargers,Chiefs  
300 `  
310 REM ** READ IN THE EXISTING STAT FILE **  
320 WN=448  
330 OPEN "`I",1,"STAT.DAT"  
340 FOR I=1 TO WN  
350 IF EOF(1) THEN 400  
360 FOR J=1 TO 5  
370 INPUT #1,A(I,J)  
380 NEXT J  
390 NEXT I  
400 CLOSE 1  
410 L1=I-1  
420 `  
430 REM * READ IN THE TEAM NAMES  
440 FOR I=1 TO 28  
450 READ T$(I)  
460 NEXT I
```

```

470 `
480 CLS: ` Clear the screen and home the cursor.
490 PRINT STRING$(22,``-``);` The CodeWorks ``;STRING$(23,``-``)
500 PRINT``
           N F L   P L A Y O F F   P R O J E C T I O N S
510 PRINT``
           Projects results of playoff games and superbowl
520 PRINT STRING$(60,``-``)
530 PRINT
540 IF L1 MOD 28 <>0 THEN PRINT``There is extra (or missing) data
      in the file`` ELSE PRINT``The stat file is currently updated
      through week``;L1/28
550 PRINT
560 INPUT``Are you projecting the Super Bowl game (y/n)``;XX$
570 IF XX$=`y` OR XX$=`Y` THEN SB=1 ELSE SB=0
580 INPUT``Do you want hardcopy output too (y/n)``;PR$
590 IF PR$=`y` OR PR$=`Y` THEN PR=1 ELSE PR=0
600 IF PR=0 THEN 650
610 LPRINT``NFL Playoff games, real or what if?``
620 LPRINT STRING$(60,45)
630 LPRINT`` ``
640 `
650 INPUT``Press ENTER to continue``;XX
660 CLS
670 `
680 ` print the teams and their numbers on the screen
690 FOR I=1 TO 7
700   FOR J=1 TO 4
710     TM=TM+1
720     PRINT TM;T$(TM),
730   NEXT J
740   PRINT
750 NEXT I
760 `
770 `input playoff teams routine
780 X=9:Y=1:GOSUB 200:PRINT``Enter playoff teams by number, 0 to
      quit``
790 INPUT``Visiting team number``;X1
800 IF X1=0 THEN IF PR=1 THEN LPRINT CHR$(12)
810 IF X1=0 THEN END
820 INPUT``Home team number   ``;X2
830 `
840 `read the stat file and collect information
850 FOR I=1 TO WN
860   IF A(I,1)=X1 THEN IF A(I,4)>A(I,5) THEN B(4)=B(4)+1
870   IF A(I,1)=X2 THEN IF A(I,4)>A(I,5) THEN C(4)=C(4)+1
880   IF A(I,2)<13 THEN 910
890   IF A(I,1)=X1 THEN B(1)=B(1)+A(I,3):B(2)=B(2)+A(I,4):

```

```

    B(3)=B(3)+A(I,5)
900  IF A(I,1)=X2 THEN C(1)=C(1)+A(I,3):C(2)=C(2)+A(I,4):
    C(3)=C(3)+A(I,5)
910  NEXT I
920  `
930  ` calculate who is gonna win
940  B(1)=B(1)/4:B(2)=B(2)/4:B(3)=B(3)/4
950  C(1)=C(1)/4:C(2)=C(2)/4:C(3)=C(3)/4
960  T1=(2*B(2))+C(3)
970  T2=(2*C(2))+B(3):IF SB=0 THEN T2=T2+2
980  IF T1=T2 THEN T1=INT(T1+B(1)):T2=INT(T2+C(1))
990  T1=INT(T1/3):T2=INT(T2/3)
1000 IF T1=T2 THEN T1=T1+B(4):T2=T2+C(4)
1010 `
1020 ` print the results
1030 PRINT T$(X1),T1
1040 PRINT T$(X2),T2
1050 IF PR=0 THEN 1090
1060 LPRINT T$(X1),T1
1070 LPRINT T$(X2),T2
1080 LPRINT `` ``
1090 INPUT''press ENTER for next pair or to quit'';XX
1100 `
1110 `clear the accumulating arrays
1120 FOR I=1 TO 4
1130   B(I)=0
1140   C(I)=0
1150 NEXT I
1160 `
1170 `clear the input area
1180 Y=1
1190 FOR X=9 TO 14
1200   GOSUB 200:PRINT STRING$(60,32)
1210 NEXT X
1220 `
1230 `loop back for another pair of teams or quit
1240 GOTO 780
1250 `
1260 END `of program

```

Changes in Playoff.Bas for Tandy Models I and III

```
Changed->100 REM * PLAYOFF/BAS * CODEWORKS MAGAZINE * 3838 S. WARNER ST.
TACOMA WA.
Changed->140 PRINT''Loading STAT/DAT data file ..''
Changed->150 CLEAR 10000: ` Use only if your machine needs to clear string
space.
Changed->200 `LOCATE X,Y:RETURN `GW-BASIC
Changed->210 PRINT@((X-1)*64)+(Y-1),,:RETURN `Tandy I/III
Changed->330 OPEN ``I'',1,``STAT/DAT''
Changed->720 PRINT TM;T$(TM),;
Changed->740 ` PRINT
```

Changes in Playoff.Bas for Tandy Models II and IV

```
Changed->100 REM * PLAYOFF/BAS * CODEWORKS MAGAZINE * 3838 S. WARNER ST.
TACOMA WA.
Changed->140 PRINT''Loading STAT/DAT data file ..''
Changed->200 `LOCATE X,Y:RETURN `GW-BASIC
Changed->220 PRINT@((X-1),(Y-1)),,:RETURN `Tandy IV
Changed->330 OPEN ``I'',1,``STAT/DAT''
```

Notes

Where do you check for EOF? We have been writing code like this and it works on all MS DOS machines using GW BASIC:

```
FOR I= 1 TO 5
  IF EOF(1) THEN...
  FOR J=1 TO 6
  etc.,
```

When we run that code on other machines, it gives an error, and we had to change where we checked for EOF to correct the problem:

```
FOR I= 1 TO 5
  FOR J= 1 TO 6
  IF EOF(1) then ...
  etc.
```

It seems that MBASIC and BASIC prior to version 5.0 want it this way.

Cword.Bas

A Chain-Word Game using States & Capitols

Staff Project. Once a year or so we feature a game program. This one is supposed to be educational. With it, you can test your knowledge of states and state capitols.

Previously we had published one or two games of chance. This time, we thought an educational game would be in order. The game is called Cword.Bas (for Chain Word) and roughly follows the game children like to play while on extended automobile trips. In that game, one person picks a name seen on a road sign, another car or truck, or anything along the highway. The next person must then find another such sign whose first letter is the same as the last letter of the first item picked. The game then progresses until one person finds names ending in hard to match letters, and stumps the other person.

Cword.Bas is similar to that just described except that it allows a match of letters at both ends of the word. The words, however, are not just anything, but are the states of the United States and their capitol cities. This, of course, gives a total of 100 names from which to pick. It turns out that there are seven capitol cities which have no possibility of being chained to other names. The computer will not pick from these seven. If you try to pick one of them it will be rejected as an "easy win" and you will be asked to pick another.

If we were only dealing with state names, then the state of Maine would be impossible to match. But since we are including the capitol cities as well, Maine can be matched with Salem, the capital city of Oregon. Some names have only one possibility of being matched. For example,

Juneau will only match with Utah, since no other name starts with the letter U or ends with the letter J. There is only one way to "win" in this game: you must take the first move and connect every name the computer picks. Even then, you only win by one point.

After the chaining of the states and capitols is over, you get the opportunity to test your knowledge of state capitols. In this portion of the program you simply try to correctly pick as many state capitol cities as possible. Be careful with St. Paul. It must be entered exactly as shown in the data statements or you will lose the point. You can, however, enter your names in upper case, lower case or mixed case; the computer will automatically change all entries into upper case so as to find the match with the names in the data statements.

When you tire of states and state capitols, you might want to try to modify the program to use Presidents and Vice-Presidents. We haven't tried it, but it might be an interesting challenge. If you do that, be sure to look for "impossible to connect" names and exclude them by putting them first in the data statements.

The Program

The program starts with the usual CodeWorks heading, followed by the universal print@/locate subroutine. This is followed by the upper case converter subroutine. Anything

we enter into this program will first be filtered through the upper case converter so that the program will see only upper case letters. This is so that you can enter your responses any way you like and the program will still find a match in the data statements.

The initialization section (between lines 320 and 350) follows next. Here, we clear some string space in line 320 if your BASIC is prior to version 5.0 and randomize the random generator using parts of DATE\$ (in line 330). A double subscripted array is dimensioned next in line 340. This array, S\$, will hold all the names and in the second subscript of the array, a flag to indicate whether or not a name has already been chosen. Line 350 contains a time delay variable, TD. If your computer runs faster than 4.7 Mhz you may want to change the value for TD to 3000 or more. On the other hand, if your speed is slower you can change TD to 500 or even 300.

The data statements follow, with the state capitols first and then the state names. The seven state capitol cities that won't chain are listed first, so that later we can easily exclude them. We could have just as well not put them there at all, but then in the second part of the program we would have missed them when we try to match states to their capitol cities.

Lines 570 through 600 read the data statements into the S\$(x,x) array. Note that we are putting a string "1" into the second element of the array with each name we enter. Later, when we use a name, we will null that second position of the array to indicate that that name has already been picked.

Lines 620 to 780 print the heading on the screen and give a little identification and instruction to the user. Lines 810 to 860 set up the playing screen with a scoreboard and ask if you want to go first (if you want to win you should always go first.)

The computer's first move is unique in that it doesn't have to match anything. So, if the computer gets the first move, lines 890 to 920 make that first pick. In line 890 we pick a name at random. Note that in line 900 if the random number picked is seven or less we pick a differ-

ent number. This is where we discriminate between those first seven capitol cities that won't chain. If we make it through line 900 we can assume we have a valid name, and so we null the second element in the array for the name picked. This happens in line 910. The computer is "ME" and "YO" are you. In line 920 the score for ME gets incremented by one, we go to the subroutine at 1760 to update the score on the screen, then to subroutine 1630 to place the name picked properly on the screen.

The computer's moves are relatively simple compared to the user's moves. With the user, we need to check for all sorts of simple things that can go wrong. Is the name a valid name? Has the name been used already? Is the user trying to cause a cheap win by using one of the unchainable names? Is the name valid but won't fit either end of the name the computer picked? Is the user crying for help? Checks for all of these conditions are made between lines 950 and 1190.

The first thing we do between lines 950 and 1190 is position the prompt on the screen to tell the user it's his turn. The response is held in A\$. Next, in line 960, we clear the area where the user entered his response (to show that it was at least conditionally accepted). Then we go to the subroutine at 240 to change A\$ into all upper case letters. Next, in line 980 we check to see if the user asked for help. If he did, we go to line 1260 and let the computer pick for him and then let the computer take the next turn.

Next, we check to see (in lines 990 to 1010) if the name entered is actually a valid name in the B\$(x,x) array. If not, we ask the user to try again. After that, at line 1050 we check for the "cheap win" shot and ask the user to try again if it is. Then we go through the array checking to see that the name picked has not yet been used (lines 1060-1090). If it has been used we ask the user to try another name.

If you got the first move then the score was zero to zero, and your entry doesn't have to match anything (but it still has to be a valid name), so line 1120 takes care of that case and simply prints your choice on the screen and

updates the scoreboard and gives the next move to the computer. If it is not the first move and the user has picked a valid name we check right strings against left strings on the name picked and the name on the screen (lines 1130-1160) and if there is a match we give the next move to the computer (lines 1150-1160) and update the user score. If there is no match (line 1170) we tell the user to try again. The score (when there is a match) is actually updated in lines 1210, just before we give the next pick to the computer. At the same time, in line 1210, we null the second element in the B\$ array to show that the name has been used.

The computer's moves are similar but simpler. They happen in lines 1260 through 1350. The computer simply goes through the B\$ array, looking for a name that has not yet been used and that matches either end of the user's pick. If it finds one, it puts it up on the screen and if there are no names left that will match it ends the game.

The second part of the program is where you match state capitols to states. If you don't want to play this part and answer no to the play question, the program ends in line 1400. Otherwise, if you do want to try your luck, line 1410 clears the screen and line 1420 puts up a screen heading. Since all the states and their capitol cities are still in the B\$(x,x) array, all we have to do is go through the array and show that none of them have been used yet. We do this in lines

1450 to 1470, where we set the second element of the array to a string "1".

Next, the computer picks states at random, from array position 51 through 100. If the name has already been picked, it makes another selection. When it finds an unused state name, it puts it on the screen and nulls the array position so that that name will not be used again. Then checks to see if your answer was correct. If it was your score is incremented by one and it goes on to the next state. If it wasn't, it tells you what the correct answer is and goes on to the next state. When all the states have been used up, line 1590 prints out how many of the 50 you got right. Notice that the order of the data statements is such that the first capitol city will match with the first state, and so on. This way, when we pick K between 51 and 100, then array location B\$(K-50,1) will be the corresponding capitol city.

Lines 1620 through 1780 are prompt positioning subroutines for the various prompts used in both parts of the program. Note that some of them first clear the position and then print there, while others simply print over what was already at that location on the screen.

It's not enough to know the capitol city of a given state. You must also know how to spell it correctly. That holds for the states as well. Have fun with it.

Cword.Bas listing for MS DOS

```
100 REM * Cword.Bas * a state and state capitol game *
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 310
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$(31+Y);:RETURN ' CP/M
220 '
230 'upper case converter subroutine
240 FOR I=1 TO LEN(A$)
250   C$=MID$(A$,I,1)
260   IF C$=>' 'a'' AND C$=<' 'z'' THEN C$=CHR$(ASC(C$)-32)
270   MID$(A$,I,1)=C$
280 NEXT I
290 RETURN
300 '
310 ' Initialization
320 'CLEAR 2000 'only if you need to clear string space
330 RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
340 DIM S$(100,2)
350 TD=1000 ' time delay variable
360 '
370 DATA PHOENIX,BATON ROUGE,JEFFERSON CITY,CARSON CITY,PROVIDENCE
380 DATA PIERRE,CHEYENNE,MONTGOMERY,JUNEAU,LITTLE ROCK,SACRAMENTO
390 DATA DENVER,HARTFORD,DOVER,TALLAHASSEE,ATLANTA,HONOLULU,BOISE
400 DATA SPRINGFIELD,INDIANAPOLIS,DES MOINES,TOPEKA,FRANKFORT
410 DATA AUGUSTA,ANNAPOLIS,BOSTON,LANSING,ST. PAUL,JACKSON,HELENA
420 DATA LINCOLN,CONCORD,TRENTON,SANTA FE,ALBANY,RALEIGH,BISMARCK
430 DATA COLUMBUS,OKLAHOMA CITY,SALEM,HARRISBURG,COLUMBIA
440 DATA NASHVILLE,AUSTIN,SALT LAKE CITY,MONTPELIER,RICHMOND
450 DATA OLYMPIA,CHARLESTON,MADISON
460 DATA ARIZONA,LOUISIANA,MISSOURI,NEVADA,RHODE ISLAND,SOUTH
   DAKOTA
470 DATA WYOMING,ALABAMA,ALASKA,ARKANSAS,CALIFORNIA,COLORADO
480 DATA CONNECTICUT,DELAWARE,FLORIDA,GEORGIA,HAWAII,IDAHO
490 DATA ILLINOIS,INDIANA,IOWA,KANSAS,KENTUCKY,MAINE,MARYLAND
500 DATA MASSACHUSETTS,MICHIGAN,MINNESOTA,MISSISSIPPI,MONTANA
510 DATA NEBRASKA,NEW HAMPSHIRE,NEW JERSEY,NEW MEXICO,NEW YORK
520 DATA NORTH CAROLINA,NORTH DAKOTA,OHIO,OKLAHOMA,OREGON
530 DATA PENNSYLVANIA,SOUTH CAROLINA,TENNESSEE,TEXAS,UTAH,VERMONT
540 DATA VIRGINIA,WASHINGTON,WEST VIRGINIA,WISCONSIN
550 '
560 'read in all the data
570 FOR I=1 TO 100
580   READ S$(I,1)
590   S$(I,2)=' '1"
600 NEXT I
610 '
620 CLS

```



```

630 PRINT STRING$(23,45);'' The CodeWorks '';STRING$(22,45)
640 PRINT''
        C H A I N   W O R D   P R O G R A M
650 PRINT''
        a word game with states and state capitols
660 PRINT STRING$(60,45)
670 PRINT
680 PRINT'' In this game you must enter a state or capitol city
690 PRINT''that starts with the last letter of the one already
700 PRINT''picked or ends with the first letter of the word already
710 PRINT''picked. Seven capitol cities cannot be matched and will
720 PRINT''be rejected as easy wins. You can only win if you start
730 PRINT''and pick every name correctly. If you ask for help the
740 PRINT''computer gets the point.
750 PRINT'' As an example, if the computer picks OLYMPIA, you
760 PRINT''can answer with either IDAHO or ALBANY.
770 PRINT
780 INPUT''Press enter to start'';X
790 `
800 `set up the screen
810 CLS
820 X=2:Y=26:GOSUB 170:PRINT''C H A I N - W O R D''
830 X=3:Y=31:GOSUB 170:PRINT''Scoreboard''
840 X=4:Y=28:GOSUB 170:PRINT ``Me           You''
850 GOSUB 1670:INPUT''Do you want to go first (y/n) '';AN$
860 IF AN$=''y'' OR AN$=''Y'' THEN 950
870 `
880 `if the computer gets the first move
890 R=INT(RND(1)*100)+1
900 B$=S$(R,1):IF R=<7 THEN 890
910 S$(R,2)=''''
920 ME=ME+1:GOSUB 1760:Q$=B$:GOSUB 1630:PRINT Q$
930 `
940 `your moves, including your first move
950 GOSUB 1670:INPUT''Pick a state or capitol city (or help)'';A$
960 X=8:Y=1:GOSUB 170:PRINT STRING$(15,32)
970 GOSUB 240 ` to make all caps
980 IF A$=''HELP'' THEN GOTO 1260
990 FOR I=1 TO 100
1000 IF A$=S$(I,1) THEN 1050
1010 NEXT I
1020 GOSUB 1720:PRINT''That is not a valid state or capitol name.
        Try again.''
1030 FOR T=1 TO TD:NEXT T
1040 GOTO 950
1050 IF I=<7 THEN GOSUB 1720:PRINT''That's a cheap win...try
        another please.'':GOTO 1030
1060 FOR I=1 TO 100

```

```

1070 IF A$=S$(I,1) AND S$(I,2)='1' THEN 1120
1080 NEXT I
1090 GOSUB 1720:PRINT''That name has already been used. Try
      another.''
1100 FOR T=1 TO TD:NEXT T
1110 GOTO 950
1120 IF ME=0 AND YO=0 THEN S$(I,2)='':GOSUB 1630:Q$=A$:PRINT Q$:
      YO=YO+1:GOSUB 1760:GOTO 1260
1130 R$=RIGHT$(Q$,1)
1140 L$=LEFT$(Q$,1)
1150 IF RIGHT$(A$,1)=L$ THEN 1210
1160 IF LEFT$(A$,1)=R$ THEN 1210
1170 GOSUB 1720:PRINT''That name does not fit either end. Try
      again.''
1180 FOR T=1 TO TD:NEXT T
1190 GOTO 950
1200 `
1210 S$(I,2)='':YO=YO+1
1220 GOSUB 1630
1230 Q$=A$:PRINT Q$
1240 `
1250 `the computer picks here
1260 FOR T=1 TO TD:NEXT T ` delay loop
1270 X=8:Y=5:GOSUB 170:PRINT ``I pick...
1280 R$=RIGHT$(Q$,1)
1290 L$=LEFT$(Q$,1)
1300 FOR I=1 TO 100
1310 IF S$(I,2)=''' THEN 1330
1320 IF LEFT$(S$(I,1),1)=R$ OR RIGHT$(S$(I,1),1)=L$ THEN ME=ME+1:
      GOSUB 1760:Q$=S$(I,1):S$(I,2)='':GOSUB 1630:PRINT Q$:GOTO
      950
1330 NEXT I
1340 GOSUB 1670:PRINT''You got me. There is nothing left that will
      match.''
1350 PRINT
1360 `
1370 ` part 2, test your knowledge of capitols
1380 PRINT''Would you like to check your knowledge of
1390 INPUT''state capitols (y/n)'';A$
1400 IF A$='n' OR A$='N' THEN END
1410 CLS
1420 X=8:Y=1:GOSUB 170:PRINT''What is the state capitol of:
1430 `
1440 `reset the array flag in S$(x,2)
1450 FOR I=1 TO 100
1460 S$(I,2)='1"

```

```

1470 NEXT I
1480 `
1490 `pick states at random
1500 K=INT(RND(1)*50)+51
1510 IF S$(K,2)=''' THEN 1500
1520 CT=CT+1:S$(K,2)='''
1530 GOSUB 1670:PRINT S$(K,1),:INPUT A$
1540 GOSUB 230:IF A$=S$(K-50,1) THEN SC=SC+1:GOTO 1560
1550 PRINT TAB(30);'No, it's `';S$(K-50,1)
1560 FOR T=1 TO TD:NEXT T
1570 IF CT<50 THEN 1500
1580 PRINT
1590 PRINT''Your score is `';SC;' out of 50.''
1600 END
1610 `
1620 `prompt locating subroutines
1630 X=8:Y=32:GOSUB 170:PRINT STRING$(15,32)
1640 X=8:Y=32:GOSUB 170
1650 RETURN
1660 `
1670 X=10:Y=1:GOSUB 170:PRINT STRING$(63,32)
1680 X=11:Y=1:GOSUB 170:PRINT STRING$(63,32)
1690 X=10:Y=1:GOSUB 170
1700 RETURN
1710 `
1720 X=11:Y=1:GOSUB 170:PRINT STRING$(63,32)
1730 X=11:Y=1:GOSUB 170
1740 RETURN
1750 `
1760 X=5:Y=27:GOSUB 170:PRINT ME
1770 X=5:Y=39:GOSUB 170:PRINT YO
1780 RETURN

```

Cword.Bas changes for Tandy I/III

```

Changed->100 REM * Cword/Bas * a state and state capitol game *
Changed->140 CLEAR 2000
Changed->170 `LOCATE X,Y:RETURN ` MS-DOS, GW-BASIC
Changed->180 PRINT@((X-1)*64)+(Y-1),,:RETURN ` Tandy Models I/III
Changed->330 `RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->350 TD=400 ` time delay variable
Changed->890 R=RND(50)+50
Changed->1500 K=RND(50)+50

```

Cword.Bas changes for Tandy II/IV

```
Changed->100 REM * Cword/Bas * a state and state capitol game *
Changed->170 'LOCATE X,Y:RETURN ' MS-DOS, GW-BASIC
Changed->190 PRINT@((X-1),(Y-1)),;:RETURN ' Tandy Models II/IV
Changed->330 'RN=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE RN
Changed->350 TD=400 ' time delay variable
Changed->890 R=RND(50)+50
Changed->1500 K=RND(50)+50
```

Randemo Recap

It's that time of year again to review the Randemo series and bring up to date all the changes for the various machines. We'll start by giving the entire listing for Ranprnt2.Bas (last issue we only gave the merge, due to lack of space).

```
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), TOT#(20)
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
70 TS$='''
100 REM -- file setup
110 CLS:PRINT FNCTR$('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 INSTR(FP$(1),"DELETED")<>0 THEN 270
260 GOSUB 2120
270 NEXT RX
275 GOSUB 4100
280 IF LC>0 THEN GOSUB 2670
500 REM -- End of Program
510 CLS:PRINT "All Done":CLOSE:END
600 REM -- input a character
610 C$=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:GOSUB 4000
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 FSS$=LN$
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$,X-1)=' '
2640 MID$(FSS$,X)=' '
2650 GOTO 2590
2660 `
2670 REM -- print footer
2680 FOR I=LC TO PG-BT-NB:LPRINT'' `:NEXT I
2690 FOR I=0 TO NB-1
2700 LN$=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
2965 IF LEFT$(LN$,1)='T' THEN GOSUB 3300: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$,X+1)
3250 RETURN
3300 REM -- Total Fields
3310 TS$ = LN$
3320 FOR ZI=0 TO 20:TOT#(ZI)=0:NEXT ZI
3330 RETURN
4000 REM -- Add to field totals
4010 FOR ZI=0 TO 20:TOT#(ZI) = TOT#(ZI) + VAL(FP$(ZI)):NEXT ZI
4020 RETURN
4100 REM -- Print field totals
4105 IF TS$=' ' THEN RETURN ELSE LN$=TS$
4110 FS$=LN$
4120 IF INSTR(LN$, '#') <> 0 AND LEN(LN$) < WD THEN LN$=LN$+STRING$(WD-LEN(LN$), ' ')
4130 IF INSTR(LN$, '#')=0 THEN 4195
4140 X = INSTR(LN$, '#')
4150 Y = VAL(MID$(LN$,X+1))

```



```

4160 MID$(LN$,X) = MID$(STR$(TOT$(Y)),2)+' ' ''
4170 IF X>1 THEN MID$(LN$,X-1)=' ' ''
4180 MID$(FSS$,X)=' ' ''
4190 GOTO 4130
4195 PRINT LN$:LC = LC + 1: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

```

Ranprnt2.Bas changes for Tandy I/III

```

Changed->30 DIM FP$(20), SC$(16), XY(20,3), TI$(20)
Changed->41 DEF FNLF(X) = LOF(X)
Changed->50 CL=64:RW=16
Changed->125 FD$=FF$+''/dat'':FSS$=FF$+''/stk''
Changed->140 FM$=FF$+''/MAP'':FX$=FF$+''/SCN''
Changed->191 FI$=FF$+''/IDX''
Changed->810 PRINT@((X-1)*64)+(Y-1),,:RETURN
Changed->2790 RF$=RF$+''/PRT''
Changed->3320 FOR ZI=0 TO 20:TI$(ZI)=0:NEXT ZI
Changed->4010 FOR ZI=0 TO 20:TI$(ZI) = TI$(ZI) + VAL(FP$(ZI)):NEXT ZI
Changed->4160 MID$(LN$,X) = MID$(STR$(TI$(Y)),2)+' ' ''

```

Ranprnt2.Bas changes for Tandy II/IV

```
Changed->10 REM - RANPRINT/BAS - GW BASIC Random File Printing
Changed->41 DEF FNLF(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''
```

Next, we'll give the changes for Randemo7.Bas. These changes apply to the original listing in Issue 14 (Randemo5.Bas) and the merge files in Issue 15 that made Randemo7.Bas from Randemo5.

Changes to Randemo7.Bas for Tandy Models I/III

```
Added-->15 CLEAR 2000
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->130 OPEN ``R'',1,FD$:OPEN ``R'',2,FS$,4:FIELD 2, 4 AS SK$
Changed->140 FM$=FF$+''/MAP'' :FX$=FF$+''/SCN''
Changed->810 PRINT@((X-1)*64)+(Y-1),,:RETURN
Changed->1070 IF MR THEN 1010
Changed->1730 GOSUB 1800:IF DUN THEN RETURN
Changed->1810 IN$=```` :DUN=FALSE:MR=FALSE:CF=FALSE
Changed->1831 IF C$=CM$ THEN DUN=TRUE:MR=FALSE:GOTO 1880
Changed->1835 IF C$=NX$ THEN DUN=TRUE:MR=TRUE:GOTO 1880
Changed->2050 IF MR AND MID$(SF$,1,1)<>''#' THEN FR=RN+1:GOTO 2030
Changed->3050 IF MR AND MID$(SF$,1,1)<>''#' THEN FR=RN+1:GOTO 3030
Changed->3250 X=16:Y=1:GOSUB 800
```

Note: For Tandy Model III only. When you enter BASIC and are asked the "How many files?" question, be sure to answer with 3V (for 3 variable length files) or you will get a Bad File Mode error when you run Randemo7.Bas. Yes, we know that the program creates more than three files, but there are never more than three open at any one time. Also (this applies to all machines) **do not** use the number sign (#) in your data input, as in an apartment number on an address. If you do it will cause loads of trouble when you try to print the file with either Ranprnt.Bas or Ranprnt2.Bas.

Changes to Randemo7.Bas for Tandy Models II and IV

```
Changed->10 REM -- RANDEMO7/BAS - Random Files with Screen Control
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->130 OPEN ``R'',1,FD$:OPEN``R'',2,FS$,4:FIELD 2, 4 AS SK$
Changed->140 FM$=FF$+''/MAP'':FX$=FF$+''/SCN''
Changed->810 PRINT@((X-1),(Y-1)),,:RETURN
```

The next set of changes apply to Tandy Models II and IV, and are to be applied to Ranidx.Bas from Issue 18. Ranidx.Bas uses the "System" or "Shell" commands, for which we have found no counterpart for Models I/III nor for some other machines. They should use Ranindex.Bas, changes for which will come later in this article.

Changes to Ranidx.Bas for Models II/IV

```
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$+''/idx''
Changed->560 `SAVE ``ranidx/bas''.
Changed->810 PRINT@((X-1),(Y-1)),,:RETURN
Changed->3210 FT$='`SRT'+MID$(STR$(TF),2)+''/TMP''
Changed->4410 FX$ = ``TMP'+MID$(STR$(N),2)+''/XXX''
Changed->4580 KILL FT$
Changed->4620 SYSTEM ``RENAME ``+FX$+'' ``+''TO''+'' ``+FT$
Changed->4810 KILL FT$
```

The next changes are for Tandy Models I and III and are to be applied to the original listing for Ranindex.Bas in Issue 13. These users, as well as a few others, will need to use this program instead of Ranidx.Bas because of the inability to leave BASIC, perform a system command and return to BASIC.

Changes to Ranindex.Bas for Models I/III

```
Changed->10 REM -- RANINDEX.BAS - 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
```

All these changes were checked out when we prepared our yearly diskettes for the 3rd year. It saddens us not to be able to give more specific details for CP/M machines and the PC Jr. We just don't have those machines to play with.

We have been running various files using the Randemo series programs on a PC. In general, they have worked well. The speed of the programs is more than acceptable, but we did compile them and realized a great improvement in speed. We noted several points while running the programs: Never use a number sign (#) in your data! It will cause grief when you print because the number sign is used for a delimiter. Also, be very careful when you create your .PRT files. Count spaces! Each field must have one more space than the field length. We did have problems printing across 132 spaces with Ranprnt2.Bas until we included a statement in that program to set the width of lprint to 132. Making a .PRT file to print across 132 characters is a bit tricky too, but again, you must count spaces.

The programs could use a little "user" streamlining, and possibly a better name, now that we are past the "demo" stage. Some of the prompts, especially in indexing and printing should be reworded to avoid confusion. Look for more in upcoming issues.

Split.Bas

Breaking up Cwindex.Dat

Staff mini-project. When we ran into this little problem on the Model III and IV disks for this year we had to come up with a way to fix it. Here it is.

So Cwindex.Dat has been growing slowly over the months, and all of a sudden you try to load it with Qkey.Bas to look up something and lo and behold, you get an "out of memory" error! What to do?

The first thing you think of is to eliminate all those help statements in Qkey.Bas, since you probably know them all by heart now anyway. But that's not going to solve the long-range problem, is it?

How about a program to partition the file into smaller files? Nothing elaborate, just something to break that big file up into two, smaller files. You may just have found this problem if you got our disk for year 3. It has the whole 21K of Cwindex.Dat on it. When we tried to load it with Qkey.Bas (from Issue 10) we ran into the memory problem with the Models III and IV. Well, not to worry, here's a fix.

First off, you must enter BASIC with at least three files specified, so tell it 3 when it asks for how many files. Some computers default to three anyway, so it may not matter for them.

The next question is where to break the file? We arbitrarily decided to put all issues up to and including Issue 13, into one file and the rest in another. There were about 388 records in Cwindex.Dat, and that would give the second file some space to grow for a while. The question after that was how would we know if a line contained an issue number that was higher than 13? Well, we could just INSTR to the

number, couldn't we? Not really, there might be other numbers in the record that would throw us off. No, we would have to look specifically for "Issue 13" and that would do it. But, what if we were not consistent and sometimes used a capital I and sometimes a lower-case i? To get around that, we look for "ssue" and add four to it. That puts our INSTR search right after the word "issue."

But we're getting ahead of ourselves. Let's take the little program and break it down to see what and how it does. First, we open the three files that we will need. Line 120 opens buffer 2, through which we will channel data into the file Index1.dat. Line 130 opens buffer 3 to channel data into the file Index2.dat. We have to get our data from the file, so line 140 opens Cwindex.dat through buffer 1 for input. Note the other two files are opened for output.

Now all we need to do is loop through Cwindex.dat and read in one record (line) at a time and examine it. That's what our loop in line 150 does. We set the loop count to more than the number of records in Cwindex.Dat so that we will get them all. In line 160, we check for end of file (EOF) in buffer 1 so that when we get to the end of the file we know we are done and can quit. Now we line input (not just input, or it would stop at the first comma in the record!) from the Cwindex.Dat file and put the record into A\$. We then (in line 180) print A\$ on the screen for all to see. A typical A\$ might look like this:

Conversions, article, issue 18, page 30,

converting to MS DOS

In line 190, we use the INSTR function to find the character combination "ssue" and then we add four to P at that point. That gives P the value of the character position in the line of the space after the word "issue." In line 200, we are going to use the MID\$ function to find the next three characters after the position of P. That will give us the space and the two digits of the issue number. We let B\$ equal that little substring. Yes, it's still in string form, we got it out of the file in string form - it couldn't have been anything but because of the way we put it there in the first place. So in line 210 we let B equal the value of B\$. Well, the space is ignored by the VAL function and so B will end up being just the issue number and it will be in integer form. Just what we always wanted!

Now we can play "greater than" on that integer in B. In line 220 if B is greater than 13 we print to buffer number 3, which is the file Index2.Dat, otherwise we print to buffer number 2, the file Index1.Dat.

So we just go along like that, taking one line (record) out of Cwindex.Dat at at time and finding out if the issue number is greater than

```
100 REM * Split.Bas * partitions
sequential files *
110 `
120 OPEN "O",2,"index1.dat"
130 OPEN "O",3,"index2.dat"
140 OPEN "I",1,"Cwindex.dat"
150 FOR I=1 TO 450
160   IF EOF(1) THEN 240
170   LINE INPUT #1, A$
180   PRINT A$
190   P=INSTR(A$,"ssue")+4
200   B$=MID$(A$,P,3)
210   B=VAL(B$)
220   IF B>13 THEN PRINT #3,A$ ELSE
PRINT #2,A$
230 NEXT I
240 CLOSE
250 END
```

13. If it is, we put it in one file, if not we put it in the other file. When we are done, line 240 will close all files, and we will have all issues up to and including 13 in one file and the rest in another. Cwindex.Dat, itself, will still be intact because all we did was to read it.

Can you see the value of consistency in computing? If we had been indiscriminate about the way we entered our data in the first place it would be quite a chore to pick out those issue numbers. At least it would have complicated the process considerably - if not make it impossible.

Now that you know how it's done, you can make a little program that will break up virtually any sequential file into smaller files. It will have the same form as this one, only the specifics of where you break might be different - but you can handle that now that you know how.

One more thing. INSTR is case sensitive. It won't find "ssue" if we had input ISSUE, for example. Makes another point for consistency. Maybe we should make a bumper sticker that says: "Computers like consistency!"

**When we got done
pasting this issue
this space just
"happened"
to be left over.
Is it a coincidence that
it's right next to the order
form, or what?
If you haven't renewed
yet, would you please?**

Thanks

Handy Order Form

| | | |
|--|---------|--|
| RENEW Subscription Nov/Dec 88 through Sep/Oct 89 | \$24.95 | |
| All third year issues, Nov 87 through Sep 88 | \$24.95 | |
| All second year issues, Nov 86 through Sep 87 | \$24.95 | |
| All first year issues, Sep 85 through Sep 86 | \$24.95 | |
| 1st Year Program Disk (issues 1 through 7) (Specify computer type below) | \$20.00 | |
| 2nd Year Program Disk (issues 8 through 13) (Specify computer type below) | \$20.00 | |
| 3rd Year Program Disk (issues 14 through 19) (Specify computer type below) <i>Available now</i> | \$20.00 | |
| NEW! "Starting with MS DOS" 40-page book explains all | \$7.00 | |

Postage and handling charges already included in all prices.

We can supply program diskettes for PC/MS DOS, TRSDOS 1.3 (Tandy Model 3), TRSDOS 6.x (Tandy Model IV) and most CP/M MBASIC formats, on 5 1/4 inch diskettes. When ordering diskettes, specify your computer type.

COMPUTER TYPE _____

Check/MO enclosed

Charge to my VISA/MC _____ exp _____

Ship to: Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to:
CodeWorks, 3838 S. Warner St. Tacoma WA 98409

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

VISA/Master Card only

1188

Index & Download

What's happening with both

Here are the updates to bring Cwindex.Dat up to date through the last issue. The entire index for the first three years of CodeWorks is on the download and on our yearly diskette.

Notes, using the P option in MS DOS, issue 19, page 3

Card.bas, reference, two level sorting, issue 19, page 3

Notes, using an actual quote mark in print lines, issue 19, page 4

Notes, using disk drive cleaners, issue 19, page 4

Beginning BASIC, exploring PRINT USING, issue 19, page 7

Hard disks, article, setting one up, issue 19, page 9

NFL88.bas, main program, issue 19, page 14, NFL for 1988-89

Stat88.bas, main program, issue 19, page 18, stats for NFL88

Correl.bas, main program, issue 19, page 22, correlation with lead/lag

Outline2.bas, demo program for outline.bas, issue 19, page 30

List.bas, correction, missing lines, issue 19, page 34

Random files, article, adding column totals, issue 19, page 37

Ranprnt.bas, merge program, issue 19, page 38

Download, notes on download, issue 19, page 40

The download has been a little erratic lately, still due to power changes and switching, but mostly it has been up and running.

We have been putting the NFL stats on the download every week, usually by Tuesday noon. Don't forget that the program in this issue for predicting the post-season play will need week 16 stats. We'll be sure and put them up as soon as we have them so that you can get into the playoff picture.

CodeWorks

3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
Postage
PAID
Permit # 774
Tacoma, WA

CODEWORKS

Issue 21

Jan/Feb 1989

CONTENTS

| | |
|---|-----------|
| Editor's Notes | 2 |
| Forum | 3 |
| Beginning BASIC | 6 |
| Writing Filters in BASIC (Fileutil.Bas) | 7 |
| Frame.Bas | 14 |
| Trust.Bas | 32 |
| Notes | 38 |
| Order Form | 39 |
| Index & Download | 40 |

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

(c)1989 80-Northwest Publishing Inc. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, the publisher assumes no responsibility for errors or omissions. Unless otherwise noted, all programs presented in this publication are placed in public domain. Please address all correspondence to CodeWorks, 3838 South Warner Street, 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. 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, WA.

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

As many of you have probably guessed by now, not all the names shown on our masthead are here in the office, working on the magazine. No, some of those names have full-time jobs elsewhere and are kind enough to give a helping hand as "friends" of the magazine. Al Mashburn is our technical advisor, and writes occasional pieces for us. His main occupation is in sales and service for a well-known motorcycle company. Cam Brown is a high school instructor at a local private school. His assistance is in the form of ideas and suggestions on the editorial content and layout of the magazine itself.

You all know Terry Dettmann, since he has been a regular contributor to the magazine from day one. Terry has the title of "Chief Scientist" at a local electronics company. His company recently expanded sales and now covers the entire United States. Lately, Terry has spent more time flying around the country than he usually does. In fact, he told me, in one of his rare weekends at home, that he has spent six of the last seven weeks away from home. He also recently had a catastrophic hard disk crash, and a backup system that apparently failed. Consequently, when he is home, he is busy trying to reconstruct the many programs, articles and projects he had on that system.

All this sounds like a sneaky

way of telling you that his articles are not going to appear in this issue of CodeWorks either. Which is true. We hope to have the finish to Outline.Bas ready for the next issue, but from the way things are looking for Terry, we can't make that a hard promise. Rest assured, that we will get it in eventually. Ditto for more of the Randemo series.

In spite of all that, we appreciate the efforts of all those who actively assist in the production and content of CodeWorks. For the most part their work is *pro bono*, and we appreciate that even more. At this time of the year it is more than appropriate that we say "thanks" to all of you.

If you notice a subtle change in this issue, it is because I have decided to go back to good old New Century Schoolbook as a typeface for the body text. When we first got the laser printer and looked at its available fonts, Bookman stood out and called for attention. It's a nice face, but after a couple of issues it just seemed too robust for the body copy. We are still using it, and it looks very good, when used as headings. Actually, CodeWorks had been in New Century Schoolbook since the very first issue, back in September of 1985. We had decided then that Times Roman was too anemic and that

continues on page 38

Forum

An Open Forum for Questions and Comments

NFL88 got 13 of 14 this week (6-7 Nov). Know any good bookies? Now, if we can only get Lpick.Bas to work as well.

Jim Brandenburg
Lewiston, ID

I intended to write sometime ago, commenting on the interest in genealogy I have seen in CodeWorks.

I have by no stretch of the imagination tried all of the genealogy programs available for the various systems, but have looked at several public domain and shareware programs for the MS DOS machines. The one I finally settled on was Personal Ancestral File. It is \$39.95, including shipping, and well worth it. Contact:

The Church of Jesus Christ of Latter-day
Saints
Genealogical Department
Ancestral File Operations Unit
50 East North Temple Street
Salt Lake City, Utah 85150
(801) 531-2584

PAF consists of several programs which record family data, information on references, report printing and a data communications capability. File size is limited by available storage space. It is well documented and includes tutorials in the manual.

I believe PAF is also available for CP/M machines and Apple.

John M. Gregg
Florence, SC

Several local people we know also have said that this is a very good program.

...Your Beginning BASIC column is the best thing since the wheel. Issue 19, Exploring the Print Using command is terrific. I suppose if all the command possibilities were in the manual, it would turn out to be quite a tome. I have learned quite a bit from that column. Keep up the good

work.

Richard L. Bacon
Tivoli, NY

Yes, it would be quite a tome. It seems that most applications programs today need more explanation. Look at the number of books being written (and bought and read) on WordStar, PageMaker, Lotus 1-2-3 and the like.

...In reply to the letter by M. L. Hall in Issue 19, I agree! As a living, I repair stereo equipment and as a hobby, computers. I have opened my TRS-80 Model III up more than once to install more memory, disk drive controller board and an RS-232 board. So yes, I agree with him in adding to his TRS-80. At this time I do not wish to convert to a 16 or 32 bit machine, although this may appeal to me in the future. What I'm trying to say is that I'd be interested in other technical projects for my computer and I'm sure others would be too...

...Because of the availability of "modern" computers, there seems to be a lot of unused, forgotten, broken, and other TRS-80s waiting to be used. You can get them cheap if you look. And what an easy way to get into computers...

Pat Chong
Las Cruces, NM

They were great computers in their day. Unfortunately, almost no one is writing anything for them these days. In spite of all the advertising hoopla, the new applications are a giant leap ahead of what passed for applications software in days past. And you can't really appreciate it until you actually make the switch yourself.

I enjoy your magazine even though I have an Amiga 1000, and know that most programs are for MS DOS machines. Therefore, I'd like to ask if there are subroutines I can substitute for Shell Commands. In Ranidx.Bas there are several commands: Shell "erase"+FT\$ and Shell "ren"+FX\$"+FT\$. If the first command erases FT\$ then a subroutine would work, however, I

don't know what "ren" does.

**Allan W. Wardell
Providence, RI**

Erase and kill are one and the same, and either can be done from within a BASIC program, as in: 100 KILL "filename". REN is the rename command, and cannot always be done from within a BASIC program as it can be with GW BASIC and the Shell command. We have been looking in vain for a way around the Shell problem and can't seem to find one. Perhaps we have painted ourselves into a corner. My version of MS DOS is 2.11 and it has the SHELL and works fine. We'll keep looking, but in the meantime, you will probably need to use Ranindex.Bas instead of Ranidx.Bas. We know that will restrict you somewhat, but it's the best we can offer for now. But also look into this: some computers use "SYSTEM" instead of "SHELL" and that may work for you. Tandy Models II and IV use System; their Models I and III won't allow it at all.

I have been a subscriber to CodeWorks since Issue 1 and have thoroughly enjoyed it. There must still be a few of us who program in BASIC, so keep up the good work!

I moved up to MS DOS about a year ago (before that I had a Model III TRS-80) and still like to work in BASIC. However, I miss some of the features of the Model III, especially the ability to scroll up or down a program at random (another thing I miss is the machine automatically looking on other drives for a program if it isn't in the default drive.)

Anyway, getting back to the scrolling: is there any program that would allow you to list 330 and then scroll up or down from there, the same way it did on the Model III...

...I own a small business and we took your payroll program (which is pretty good as it stands) and reworked it so that it prints out the weekly, monthly, quarterly, yearly and also fiscal-yearly reports, accumulating all the data as it goes. It also prints out the checks on a Swintec typewriter. We are in the process of converting it into MS DOS with TRSCROSS (a great program!)

**Francis C. Williams
Honolulu, HI**

Scrolling backward and forward in a BASIC

program is one thing that GW BASIC lacks. Look up the PATH command in MS DOS. It can make the computer search any of the drives for a given program. We have incorporated several cosmetic changes to Pay.Bas and will publish them when space permits.

...I currently use a TRS-80 Model IV with 128K of RAM. When running any BASIC program only 64K is usable, to the best of my knowledge. Has anyone come across a way to make the full 128K accessible to a BASIC program?

I am seriously considering switching to an IBM compatible. I note that they have a much larger RAM. If I have an IBM compatible with 512K will that full 512K be accessible to a BASIC program under GW BASIC, or will I still be limited to 64K? Also, if you have 512K initially and later add on more RAM, does the added on RAM automatically get recognized by the CPU?

**Marc Miller
Long Beach, CA**

You can stuff machine language that your BASIC program will call into the upper 64K of your Model IV. Same for MS DOS. GW BASIC still only uses 64K for BASIC, although with the Microsoft QuickBASIC compiler, for example, you can designate a whole 64K chunk for arrays and another 64K chunk for your BASIC program. All that extra memory is used primarily for machine language applications programs, which can span 64K boundaries. Yes, if you have 256K, for example, and add more RAM, MS DOS will recognize that it's there. On most machines you need to move a jumper on one of the boards to tell it the extra memory is there.

I once had a Heath H-89 computer which should be the same as the H-89A. The locate cursor sequence as given in Dmaker.Bas, Issue 17, page 20, program line 250 worked fine on my machine. I suspect the problem B.T. Jeavons (Forum, Issue 19) is having in getting demo programs Cursor 1, 2 and 3 to run correctly lies in the sequence given on page 6 of Issue 17: PRINT CHR\$(27)+"Y"+CHR\$(31+X)+CHR\$(31+Y). This sequence needs a semicolon at the end to avoid a carriage return/line feed. The semicolon

was used in line 250 of Dmaker.Bas...

Robert L. Anderson
St. Albans, WV

...In Issue 20, page 7 there is an Error Message article that is very nice if you don't have a TRS-80 Model III. For this Model there is a simple PATCH that will give you text instead of an error number. It patches the system program on the disk and from then on it will give the text shown on page 90 of the Model III manual.

```
PATCH *4 (ADD=4E28,FIND=20,CHG=18)
```

At the TRSDOS Ready, typing this in *exactly* will modify the disk in the drive so that error message will be in text and not in stupid numbers.

Keep up the good work. I have used the Card.Bas program for a couple of different things. Each one is a modification of the original, to suit the situation.

I'm the roster keeper for our organization of survivors from the USS Abner Read (DD 526). Each name has nine sub items and using this main data base we can print out various lists sorted on Rate/Rank or ZIP or any other way that is needed. It is a very handy program.

W. J. Pottberg
Burlingame, CA

Thanks for the patch, and glad you are getting some use out of Card.Bas.

Back in the dark ages, eons ago, when my TRS-80 Model I was the best and easiest computer around, I obtained from Tandy a program called "Cross Reference."

This was a machine language program which, among other things, would go through a BASIC program and pick out all the variables. It would then provide a printed list of these in alphabetical order and would also make a list of each line in which each variable appeared.

I found this program very handy in program writing when I would lose track of what variables I had already used or for a reference sheet I kept with long programs for use when later modifications were in order.

I'm sure you recall this program. My question

is whether there is something like it available now for use with BASIC in an MS DOS machine such as my Tandy 1400LT. If you know of such a utility, where could I obtain it? If not, how about putting it on your list of programs to work on for publication in CodeWorks? I am sure many of your subscribers would find it as useful as I did.

Charles B. Steele
La Jolla, CA

We published that program (in BASIC, but it could be compiled) in Issue 5, back in May/June 1986. It was called VXREF.Bas. As I recall, that program wasn't the last word in such utilities, and was rather lumpy in spots. If it wasn't compiled, it was also rather slow. Perhaps a re-write and an update of that program would be in order - especially one that could be compiled. Thanks for the nudge.

Thank you again for the interesting and varied input. You would be surprised at how many of our programs are in direct response to your letters, and that's how it should be, after all. So now it's time for us to tell all of you: Keep up the good work!

Irv



Would you believe he's writing a book on how to stop smoking?

Beginning BASIC

A Look at Variable Types

When you first turn on your computer it automatically defaults to single precision accuracy for variables. Single precision gives you six places of accuracy. The exclamation point is used to denote single precision, but since this is the default value anyway, you don't need to use it. Naturally, you would ask why have it then? Well, you can define all variables to some other accuracy at the beginning of the program, and then, if you wanted some selected variable to be single precision, you could use the exclamation mark for just that variable. In this case, variable A and variable A! both refer to the same value, because they are one and the same variable.

The percent sign (%) used after a variable denotes that that variable is an integer variable. The range of integer variables is from -32768 to +32767. If you input 7.7 for variable A% and then print A% you will find that it prints 8. That is because BASIC rounds off integer variables to the nearest whole number. Notice that A and A% are two entirely different variables, not like single precision variables, above. Integer variables take up less memory space than other types and will compute faster. For this reason, it is usually good practice to declare loop counters as integers because it tends to speed up execution of your programs. If you know that you will be using variables I, J and K for loop counters, then at the beginning of your program you can put a statement like this: DEFINT I,J,K (or DEFINT I-K, which is the same thing).

But why should integer variables go from -32768 to +32767? Well, it turns out that one byte of 8 bits can hold up to 256 different values. If we use two bytes, then, to hold a number we have $256 * 256$ different values. That turns out to be 65536 discrete values, which is the number of values between -32768 and +32767.

There is a so-called double precision variable. It is so-called because it is actually almost three times as accurate as single precision. You can use the number sign (#) after a variable to denote double precision, as in A#. Double precision can be accurate up to 16 places. But be careful. If you define one variable as double precision and multiply it by another that is not, and then put the result into another single precision variable, it may print out 16 places but only the first six would be significant. To get the full benefit of double precision, at least one of the variables and the output variable should be in double precision. You can give a "global" definition to your variables to make them double precision: DEFDBL A-C, for example, would make all variables from A to C double precision. Again, A and A# are two entirely different variables. (But if you DEFDBL A then all variables starting with A will be double precision.) A "global" variable designation means that it applies to the entire program.

In addition to the numeric variable types we have just discussed; integer, single precision and double precision, there is also the string variable designator that you all already know about, the \$ after a variable shows that that variable is a string variable. You can also define string variables globally, as in DEFSTR A-F, but we tend to stay away from these types of designations since they can become rather confusing, even to experienced programmers. And again, A and A\$ are two entirely different variables.

So there is a quick look at the various variable types you can use. Play with them and see what they do. Then add them to your arsenal of computing tools. Once you know what they do, you will certainly find uses for them.

Fileutil.Bas

Writing Filters in BASIC

Irene P. Governale, Port Jervis, NY. Here is a collection of useful file utilities which you can use individually or with the menu driven program following this article.

When I tried to define filters, I kept coming up with phrases that would be just as applicable to the word "program." So, I looked in my MS DOS Reference Manual (when all else fails, read the documentation, right?) and this is what I found:

"A filter is a command that reads input, transforms it in some way, and then outputs it, usually to the screen or to a file. Thus, data is said to have been filtered by the command."

Not very helpful, is it? Filters usually work on text files. They do things like removing extraneous control characters or changing text from upper case to lower case. Although some filters work on files a word or a line at a time, the program SORT that comes with the MS DOS operating system for instance, the programs we're going to write here work on one character at a time.

When written in either assembly language or a compiled language, filters get their input from STDIN (the standard input device, usually the keyboard) and send the transformed output to STDOUT (the standard output device, usually the screen.) That doesn't seem very useful at first glance, but both MS DOS and UNIX allow you to redirect either or both of these devices to another device (perhaps the printer?) or file. Redirection won't work with interpreted BASIC, but you can still write filters that will do their jobs.

The pseudocode for a filter written in BASIC would look something like this:

1. Get input and output file names
2. Open the files in random mode with a record length of one
3. Read a byte from the input file
4. Do something to it
5. Write the byte to the output file
6. If we haven't reached the end of file, go back to step 3 and get the next byte
7. Close the files

Now, if you look closely at this pseudocode, you'll see that most of the code will remain the same from program to program. The only place we need to make changes is where we "do something." The following listing provides a skeleton for a BASIC filter, all we'll have to do is fill in the subroutine.

```
10 CLS:CR=13:LF=10:RTN=0: 'cr=
    carriage return, lf=line feed
20 INPUT"Enter input file name: ";FI$
30 INPUT"Enter output file name: ";FO$
40 OPEN "R",1,FI$,1:OPEN "R",2,FO$,1
50 FIELD #1,1 AS IN$:FIELD #2,
    1 AS O$
60 I=1:J=1
70 GET #1,I
80 A=ASC(IN$)
90 GOSUB 500
100 LSET O$=CHR$(A)
110 PUT #2,J
120 I=I+1:J=J+1
130 IF NOT (EOF(1)) AND A<>26
    THEN 70
```

```
140 CLOSE
150 END
```

It's always nice to clear the screen at the beginning of a program, and that's the first thing we do. (You will need to change the CLS to whatever command your computer uses to clear the screen.) Then we set up some standard variables: CR is the ASCII code for carriage return, LF is the code for a line feed, and RTN will be used by programs that need to count carriage returns.

The next two lines get the input and output file names from the user. These can be standard file names or, on MS DOS computers you can use CON for the keyboard or screen, or PRN for the printer.

In line 40 we open the files in random mode with a record length of one. I thought about saving some processing time by reading more characters at once and then processing them, but if the last record is shorter than the record length defined by the program, you end up padding your output file with unnecessary characters.

Line 50 sets up the buffer variables for the two files and line 60 initializes the two variables we'll be using for file pointers. Now that we've done all the set-up, we can get down to business. Lines 70 and 80 get a character from the input file and convert it to an ASCII code in the variable A.

Line 90 then calls the subroutine that will do the actual processing. The only reason for putting this code in a subroutine is to make it easy to write new filters simply by replacing the subroutine code. If you save this skeleton in ASCII format, you can simply merge it into your subroutines as you write them.

Once we've returned from the subroutine, we move the character code into the output buffer and send it to the file. Then we increment our two pointers, and if we haven't reached the end of file, we loop back to line 70 to get the next character. For a more detailed explanation of

line 130, see the sidebar "Finding EOF."

Now for the first filter. If you've ever used the TYPE command to examine a WordStar file on the screen, you've seen a lot of weird characters. WordStar turns on bit seven of some characters, for instance line feeds within paragraphs. Turning on bit seven adds 128 to the character's ASCII code. Since characters with ASCII codes above 127 are graphics characters, this makes for the weird display. To convert these characters back to standard ASCII, all you have to do is turn off bit seven. The lines of code in the next listing (WSCONVRT) will do that.

```
490 ' WSCONVRT - convert WordStar files to
Standard ASCII
500 A=A AND 127
510 IF A<>LF THEN PRINT CHR$(A);
520 RETURN
```

Line 500 ANDs the value of A with 127. Since the binary value of 127 is 0 1 1 1 1 1 1, ANDing 127 with a number like 130 (1 0 0 0 0 1 0), would result in (0 0 0 0 0 1 0). AND performs a logical AND on the two numbers on a bit by bit basis. The only bits in the resulting number that will be set to 1 are those which were set to one in both numbers. You can visualize it this way:

```
1 0 0 0 0 1 0 (decimal 130) and
0 1 1 1 1 1 1 (decimal 127)
-----
0 0 0 0 0 1 0 (decimal 2)
```

Line 510 is not really necessary, but I can't stand to sit there looking at a blank screen. I *know* the computer is working, but I'd rather have the program run a bit slower and show me what it's doing. It's up to you whether you want that line. If you'd rather speed up the processing of the file, leave it out.

The reason that line 510 checks the ASCII code and doesn't print line feeds is that GW-BASIC treats carriage returns and line feeds the same way. It returns the carriage to the left side of the screen and advances a line. If you printed a carriage return/line feed pair, the document would appear to be double spaced.

Now for two filters that do a similar job. The first is called UCASE. It converts all lower case letters in the input file into upper case. Take a look at the listing which follows.

```
490 ' UCASE - convert file to
    all upper case
500 IF A>=97 AND A<=122 THEN A=
    A-32
510 IF A<>10 THEN PRINT CHR$(A);
520 RETURN
```

The ASCII codes for lower case letters fall between 97 and 123. Since these are the only characters we want to work on, we test the value of the variable A for codes in that range. The difference between the code for 'A' and the code for 'a' is 32, but if you simply subtracted 32 from the ASCII value of all the characters in the file, you'd end up with a pretty strange file! Now for the other filter, which I'm sure you have already figured out, which is called LCASE. To convert upper case letters to lower case, use this code.

```
490 ' LCASE - convert file to
    lower case
500 IF A>=65 AND A<=90 THEN A=A
    +32
510 IF A<>10 THEN PRINT CHR$(A);
520 RETURN
```

The only difference between this subroutine and the last one is that the letters we want to work with here have ASCII values in the range 65 to 90. Now for a pair of programs that change the line spacing of a file.

Listing SSPACE, changes a file from double space to single space. The first thing this subroutine does is check to see whether or not we've found a carriage return. If we haven't, we simply print the character and return. When we find the first carriage return, we set RTN to one and return. This way, the first carriage return/line feed pair will be printed to the output file. The next time we find a carriage return, we increment I and get the next character from the input file.

```
490 ' SSPACE convert file to single space
500 IF A<>CR THEN PRINT CHR$(A)
    ;;RETURN
510 IF A=CR AND RTN=0 THEN RTN=
    1:RETURN
520 IF A=CR AND RTN>0 THEN I=I+
    1:GET #1,I:A=ASC(IN$):IF A=
    LF THEN I=I+1:GET #1,I:A=AS
    C(IN$):GOTO 520
530 PRINT CHR$(A);:RTN=0:RETURN
```

If the next character is a line feed, which it should be on a standard text file, the variable I is incremented again, then we get the next character and execute line 520 again. As long as we keep finding carriage return/line feed pairs, we'll keep executing line 520. As soon as we find something other than a carriage return, we drop down to the next line which sets RTN to zero, prints the character on the screen and returns.

This is probably the most complicated example we'll have, so now you can breathe a sigh of relief and go on to the next filter, DSPACE, which will double space a single spaced file.

```
490 ' DSPACE - convert single
    spaced file to double space
500 IF A<>CR THEN PRINT CHR$(A)
    ;;RETURN
510 GET #1,(I+1):A=ASC(IN$)
520 IF A=LF THEN LSET O$=CHR$(C
    R):PUT #2,J:J=J+1:LSET O$=C
    HR$(LF):PUT #2,J:J=J+1
530 GET #1,I:A=ASC(IN$):RETURN
```

Once again we check to see if we've got a carriage return. If we don't, we print the character and return. When we find a carriage return, we check to see if the next character is a line feed. If it is, we print the carriage return/line feed pair to the output file, incrementing the output file pointer as we do so, then we get the current character (the carriage return) again, print it, and return.

If you'd like to display a file on the screen with-

out having to leave BASIC, the next listing is for you. Since we know that the output will be on the screen, we can remove any code that deals with an output file. This leaves only the request for the input file name, and the code to open and read from the file.

```
10 CLS:CR=13:LF=10:RTN=0
20 INPUT"Enter input file name: ";FI$
30 OPEN "R",1,FI$,1
40 FIELD #1,1 AS IN$
50 I=1
60 GET #1,I
70 A=ASC(IN$)
80 GOSUB 500
90 I=I+1
100 IF NOT (EOF(1)) AND A<>26
    THEN 60
110 CLOSE
120 END
497 '-----
498 'BTYPE - displays a file on
    the screen
499 '-----
500 IF A<>LF AND A<>26 THEN PRINT CHR$(A);
510 IF A=CR THEN RTN=RTN+1 ELSE
    RETURN
520 IF RTN=21 THEN PRINT"<more>"
    "; ELSE RETURN
530 IF INKEY$="" THEN 530
540 PRINT:RTN=0:RETURN
```

When we get to the subroutine, the first thing we do is check to see if the character is either a line feed or the end of the file. If it is neither, we print it. Then, since we don't want the text to scroll off the screen before we can read it, we check for a carriage return. If we do have a carriage return, we increment RTN. If the character is anything other than a carriage return, we're done processing this character.

The next thing we need to do each time we find a carriage return is check to see if we've printed a screenfull of lines. If RTN=21, then we've filled the screen and we print "<more>" and wait for a key press at line 530. The last thing we need to do is PRINT, so that the next character will be

printed on a blank line, set RTN back to zero so we can start counting again and return.

I saved the easiest filter of all for last. It doesn't do anything to the input file. It just copies it to the output file. The subroutine in Listing BCOPY is only there to cover the case where the input file might be "CON" (the keyboard in MS DOS). Its sole reason for existence is to let you see what you're typing.

```
480 'BCOPY - simply copies input to output
500 IF FI$="CON" OR FI$="con"
    THEN PRINT CHR$(A);IF A=CR
    THEN LSET O$=CHR$(A):PUT
    #2,J:J=J+1:A=LF
510 RETURN
```

What possible use would we have for this program? Well, depending on the input and output files we choose, we can use it to create simple text files, print a file on the printer, or copy files. How's that for simplicity? Of course, you'd better be a pretty good typist to use this program to create files because a CHR\$(8) (backspace) would be printed to the file just like any other character.

One final word of warning. If you want to use BCOPY to copy program files, you'll have to remove the check for CHR\$(26) from line 130. Remember, the occurrence of CHR\$(26) in a program file might not necessarily signal end of file. If we stopped processing a program file the first time we encounter CHR\$(26), we might end up copying only part of the file.

There's still room for improvement in our program skeleton and even in the subroutines. We could add error checking for the read and write process, or we could add code to check for the existence of the input file, or code to prevent over-writing existing output files. Those elements were left out so we could concentrate on the filters themselves. But, as they say, no program is ever really completed.

Now, why write filters in BASIC? Well, for one thing it's easy. Besides, if you don't have a compiler or assembler, what other way is there?

These short examples have gotten you started, now here are a couple of ideas you can work on yourself. How about a filter to expand tab characters to the correct number of spaces to carry you to the next tab stop? You could add a variable for the size of the tab and request it from the user at run time. Or how about a word counting program? Or . . . now it's your turn. Have fun with BASIC filters.

(Editor's Note: We have compiled these fragments of code into one menu-driven program (FileUtil.Bas) for your convenience. Listing follows. This is the version which will appear on the CodeWorks download and on the yearly diskette.)

Finding EOF

Line 130 of our program skeleton says:

```
130 IF NOT (EOF(1)) AND A <> 26 THEN 60
```

Why the double check? Well, normally finding

the end of a file is easy. The BASIC function EOF(filename) will tell us when to stop processing. Actually though, under MS DOS there are two ways of determining the end of a text file. You can use the actual length of the file in bytes, or you can look for a CHR\$(26) which is Control-Z.

Most programs that produce text files end them with Control-Z, but programs like WordStar may pad the file with more than one Control-Z in order to make the size of the file evenly divisible by 128. This is one more place where MS DOS's CP/M ancestry shows. CP/M creates files in blocks of 128 bytes which means that the only way to determine the actual end of the file is that terminal Control-Z.

Since we're dealing with our files on a byte by byte basis, those of us using MS DOS have a choice of processing all the way to the end of the file or stopping at the first Control-Z. Only one Control-Z is necessary to terminate a text file, so there's no reason to pass along any padding we might find. We still check for EOF though, just in case we come across a program that doesn't use the Control-Z.

Fileutil.Bas for all models

```
100 REM * FileUtil.Bas * A file utility program by
110 REM * Irene P. Governale, Port Jervis, NY for
120 REM * CodeWorks Magazine 3838 S. Warner St. Tacoma WA 98409
130 REM * (206) 475-2219 voice (206) 475-2356 300/1200 modem
140 REM * (c)1988 80-NW Publishing Inc. & placed in public domain.
150 'CLEAR 2000 ' use only if you need to clear string space
160 CLS
170 PRINT STRING$(22,45);'' The CodeWorks '';STRING$(23,45)
180 PRINT''          F I L E   U T I L I T Y   C O L L E C T I O N
190 PRINT''          a group of 7 file utilities
```

```

200 PRINT STRING$(60,45)
210 PRINT
220 PRINT TAB(10);''1 - convert WordStar files
230 PRINT TAB(10);''2 - convert file to all UPPER CASE
240 PRINT TAB(10);''3 - convert file to all lower case
250 PRINT TAB(10);''4 - convert file to single space
260 PRINT TAB(10);''5 - convert file to double space
270 PRINT TAB(10);''6 - display a file on the screen
280 PRINT TAB(10);''7 - copy a file
290 PRINT
300 INPUT''Number of your choice'';XX
310 IF XX=6 THEN 850
320 `-----
330 ` Mainline of program
340 `-----
350 CLS:CR=13:LF=10:RTN=0: `cr=carriage return, lf=line feed
360 INPUT''Enter input file name: ``;FI$
370 INPUT''Enter output file name: ``;FO$
380 OPEN ``R'',1,FI$,1:OPEN ``R'',2,FO$,1
390 FIELD #1,1 AS IN$:FIELD #2,1 AS O$
400 I=1:J=1
410 GET #1,I
420 A=ASC(IN$)
430 ON XX GOSUB 530,590,650,710,780,850,1050
440 LSET O$=CHR$(A)
450 PUT #2,J
460 I=I+1:J=J+1
470 IF NOT (EOF(1)) AND A<>26 THEN 410
480 CLOSE
490 END
500 `-----
510 `convert WordStar files to standard ASCII
520 `-----
530 A=A AND 127
540 IF A<>LF THEN PRINT CHR$(A);
550 RETURN
560 `-----
570 `convert file to all upper case
580 `-----
590 IF A>=97 AND A<=122 THEN A=A-32
600 IF A<>10 THEN PRINT CHR$(A);
610 RETURN
620 `-----
630 `convert file to all lower case
640 `-----
650 IF A>=65 AND A<=90 THEN A=A+32

```

```

660 IF A<>10 THEN PRINT CHR$(A);
670 RETURN
680 '-----
690 'convert file to single space
700 '-----
710 IF A<>CR THEN PRINT CHR$(A);:RETURN
720 IF A=CR AND RTN=0 THEN RTN=1:RETURN
730 IF A=CR AND RTN>0 THEN I=I+1:GET #1,I:A=ASC(IN$):IF A=LF THEN
    I=I+1:GET #1,I:A=ASC(IN$):GOTO 730
740 PRINT CHR$(A);:RTN=0:RETURN
750 '-----
760 'convert single spaced file to double space
770 '-----
780 IF A<>CR THEN PRINT CHR$(A);:RETURN
790 GET #1,(I+1):A=ASC(IN$)
800 IF A=LF THEN LSET O$=CHR$(CR):PUT #2,J:J=J+1:LSET O$=CHR$(LF):
    PUT #2,J:J=J+1
810 GET #1,I:A=ASC(IN$):RETURN
820 '-----
830 'display a file on the screen
840 '-----
850 CLS:CR=13:LF=10:RTN=0
860 INPUT''Enter input file name: '';FI$
870 OPEN ``R'',1,FI$,1
880 FIELD #1, 1 AS IN$
890 I=1
900 GET #1,I
910 A=ASC(IN$)
920 GOSUB 970
930 I=I+1
940 IF NOT (EOF(1)) AND A<>26 THEN 900
950 CLOSE
960 END
970 IF A<>LF AND A<>26 THEN PRINT CHR$(A);
980 IF A=CR THEN RTN=RTN+1 ELSE RETURN
990 IF RTN=21 THEN PRINT''<more>''; ELSE RETURN
1000 IF INKEY$=''' THEN 1000
1010 PRINT:RTN=0:RETURN
1020 '-----
1030 'simply copies input to output
1040 '-----
1050 IF FI$='''CON'' OR FI$='''con'' THEN PRINT CHR$(A);:IF A=CR THEN
    LSET O$=CHR$(A):PUT #2,J:J=J+1:A=LF
1060 RETURN

```

Frame.Bas

Cost & Materials to Frame and Cover a Building

Staff Project. This is another of those long winter night projects where you can plan what you will be doing in the spring. With it, you can define a building (or part of one) and get a materials and cost list. Trim, paint and nails are extra.

Long winter nights are an excellent time to plan spring and summer projects. Our major project for this issue is Frame.Bas. It is a long and involved program that will determine quantity and prices for a wide variety of building projects. It takes into account the many choices available in framing and covering a building. The program covers the basic requirements only, and does not include such things as trim, nails, paint or stain or the cost of windows or doors.

Frame.Bas does take into account such things as studs, siding, plywood sheathing, drywall, felt paper, insulation, preformed rafters and both wood and composition shingles. It has the ability to include interior walls, and if you are adding to an existing structure, the ability to exclude one or more walls. Further, it can be used to calculate interior walls only, as when you already have an unfinished structure in which you wish to add partitions.

Programs of this sort have the tendency to be input intensive. There are so many different items that need to be entered, and several choices for each item. There seems to be no other way to do it than to swallow hard and dig in. But there is a neat way around doing that for every project. Once a basic structure has been entered, you can call it up and edit in changes to it and save it back as an entirely new project. The

program allows you to have as many different structures as your disk will hold. You can call any project in, change it and save it back under its original name or give it a new name and save it.

The program uses one other file, that being the unit price file called "Prices." A menu option allows you to create this file and another option lets you call it up and make changes to it. There is only one "Prices" file, and it is used for all projects.

There are two outputs to this program. One is a list of specifications for any project, the other is a material and cost list for a project. The material and cost output will combine similar items, so that you get a total, for example, of all 12 foot 2x4's used in the structure. It also calculates the total dollar amount of the project. See the accompanying figures for examples of the output for a hypothetical garage.

We won't go into an excruciating detail on the program because that would take an entire issue, but we will go through it and note those things that should be of interest to you. Let's start at the top. The defined function in line 180 is going to assist us in getting studs to their next largest two foot increment. That's because that is the way they are sold. Line 190 sets our error trap. Let's go to line 5710 and see what happens

there.

The error trap at the end of the program (line 5710 and on) checks for the error "file not found." But there are two files involved with this program. In one case we may be asking for a project file that doesn't exist. In the other case the program may have tried to read the "Prices" file and didn't find one. If we asked for a project file from the main menu, then the variable XX would be 3. In that case the error trap simply tells us that there is no file like that and sends us back to the main menu. If the "Prices" file is missing then when we get to this error trap line 5730 will come into play. All we do in that case is set H\$ equal to "Do this First" and return to the main menu. Back at the main menu, you will note that H\$ figures into line 640. It will be printed right there on the screen along with the other menu items. When we first run the program, H\$ will be a null string. If the "Prices" file is not there, H\$ will be set to our little reminder and when we come back to the menu, the menu itself will tell us to input a "Prices" file first. After that, line 680 will reset H\$ to a null string and we won't see it in the menu again. If there is a "Prices" file when we run the program, the error trap will not be sprung and H\$ will remain a null string. This is another example of "dynamic" menu items (see Issue 19, Correl.Bas, for other examples of this.)

Having gotten that all out of the way, we can go back and try to continue our top-down approach to the description of the program. The construction trade is rife with its own terminology. Wood shingles are sold by the pack, for example, while composition shingles are sold by the "Square." There are actually three "packs" of composition shingles to one square. Felt paper comes in 36-inch wide rolls, 160 feet long; insulation comes in rolls too, but is measured by the square foot. Two by fours come in even two foot lengths, so if you need 8 feet, 1 inch, you must buy 10 footers and waste the difference. That's just the way it is.

Because we are going to give options when you input specifications, we will need to keep all the possible names and lengths someplace. And

since we are going to be using those names and lengths many times, we may as well put them into their own separate arrays, so we can always come back and find what we need. Lines 210 through 490 are data statements containing all the names and lengths we will need for the program, as well as the little loops that will read the data into arrays for us. The arrays are all string arrays, and are A\$() through E\$().

In line 510 we go to the subroutine at line 2250 to read in the "Prices" file. We have already talked about what happens if that file does not exist. Line 520 checks to see if your computer uses DATE\$ and if it does, it sets DT\$ equal to DATE\$. If you have no date in your computer, the prompt in line 520 comes into play and asks you to enter a date - which becomes DT\$.

The menu follows, from line 530 to 710. Standard stuff here, but note that the "Prices" file will be automatically loaded (and saved that way too), but your projects must be explicitly saved or loaded (menu options 2 and 3, respectively.) If all we ever wanted to do was work with just one project, we could make it automatic. But we want the option of loading a file, editing it, and saving it as a different file.

Our input routine goes all the way from line 730 to line 1960. The first section of that input routine (line 750 to 870) asks for dimensions. Here, we have a problem with feet and inches. To make things simple for the program, we input feet and inches like a decimal number, that is, 10.6 would be ten feet, six inches. It may be a little unorthodox, but it's effective and does simplify things. Obviously, with this method, we don't want to try and specify half-inch increments; they aren't needed and would only complicate things.

In the next part of the input routine, from 880 to 1960, the names and numbers of the choices are printed on the screen for you. You select the one you want and the program goes on to the next. Most choices also have a "no choice" entry, so that you can ignore whatever part of the project (like ignoring the floor joists because the

floor is a concrete slab, for example.) After a tedious session of input, line 1960 takes us back to the main menu in line 530.

The section of code from 2000 to 2080 is where we save the specifications file to disk. The next section, from 2120 to 2210 lets us load any specifications file from disk.

In lines 2250 through 2300 we open and read the "Prices" file. The program automatically comes here when we first run the program. Note that it is a subroutine, with a RETURN at its end.

The option to enter new unit prices goes from 2340 to 2440. Again, note that when we have entered new unit prices the program keeps right on going and writes the "Prices" file back out.

The "Edit Specifications" routine comes next, from lines 2480 to 2700. If we have not input specifications or loaded a specifications file, then R(7,4) will be 0, otherwise, it will always contain a 4. Line 2490 tests the condition of R(7,4) and if it is zero the following two lines come into play. They inform you that you must have input a file or new data in order to edit, and send you back to the main menu to make another selection. If R(7,4) contains other than a zero, the next two lines are skipped and we get to the editing portion of this routine. Since the first six items of the specifications are slightly different than the rest, we handle them in their own loop. Then we deal with items seven through 29 in the following section of code. It may be of interest here to note that the R(x,x) array is a 29 item array, five deep. R(X,1) always contains the number of your selection, while R(X,2) through R(X,5) contain the possible choices you could have made. As an example, let's take R(25,X). R(25,X) is "Exterior sheathing" and R(25,2) contains the choice "1/2 inch ply", R(25,3) has the choice "3/4 inch ply", R(25,4) has the choice "1x6 board" and R(25,5) has the "none" choice. R(25,1) will contain the number of the choice you made when you first input (or edited) the specifications. Further, the number in R(25,2), for example, will indicate the array number in the B\$() array that says, "1/2 inch ply" if R(25,2) contained a 5. And, incidentally,

tally, the "25" in R(25,X) indicates the 25th item in the A\$() array, which is "Exterior sheathing."

In lines 2740 through 2790 we clear out the M(x,x) array. This array will be used to carry all of our information until print-out time. Again, this array will carry integer numbers that will refer to the A\$() array through the E\$() array. There are six positions in the M(x,x) array. They are: item name, size, length, quantity, price and price per. In the final output, the total price will be calculated from the quantity and price per, or in some cases, the length times the quantity times the price per. We clear this array now in preparation to filling it with the calculations for our project.

In lines 2810 through 3040 we set up a lot of calculations that will simplify later calculations. Notice, in line 2810 for example, that we are taking our feet and inches figure (remember that it was input as a decimal?) and taking it apart to find the total number of inches. Let's say that we input the length of the building as 20 feet, six inches (20.6). In line 2810 we take the integer part of that number (20) and multiply it by 12 to get inches. Then we subtract the integer part of the number from itself and have .6 left over. Now we multiply the .6 by 10 and get 6, which we add to the number of inches we obtained earlier. We now have the length of the wall in inches. In lines 2980 and following we use some trigonometry to determine the area of the gable ends of the building and also the size of the roof. Roof overhang, both on the gable ends and on the sides, is figured in as a fixed amount in lines 3000 and 3020. You can adjust these amounts to whatever you think is right.

From 3060 to 3210 we find the number and length of exterior studs needed. Exterior studs are those that form the outside walls of the structure. Here we need to figure in the thickness of the stud plates that will be used on top and bottom of the studs. There can be one or two such plates, and if the wall height was specified at eight feet, 6 inches, for example, you can still get by with eight foot studs if you have double plates top and bottom. Line 3080 sets C1 to the value for studs on 16 or 24 inch centers. In line

3090 we subtract one half of the excluded wall length (we'll be multiplying by two later to get the right wall length.) The "P" variables set in lines 3100 to 3130 will be used later, in the plate section of the code. In line 3150 we figure out how many studs we need by dividing the length of the wall by either the 16 or 24 (centers) we found earlier. We then add three studs to each dimension. We do that because you need at least two studs at the ends of the wall (corner, if you will) to form a nice corner to nail drywall to. That, plus you may not have a wall evenly divisible by the 16 or 24 inch centers, so you need one extra one there to fill out. In line 3170 we use our defined function (way up in line 180) to get our studs to the next nearest two-foot length. It does it by adding one inch at a time to the length of the stud (H1) until the defined function is not true, at which time H1 divided by 12 will always be a length in feet evenly divisible by two. Lines 3200 and 3210 then stuff the values we just found into the M(x,x) array for later use.

The next sections of code, all the way to line 4690, find all the values we will need and fill the remainder of the M(x,x) array. Ceiling joist sheathing, in line 4030, is that sheathing that you may want to put on top of your ceiling joists to form a storage space, for example.

The program only considers preformed roof rafters. This seems to be the way everyone is going these days. Also, if the roof will be covered with composition shingles then a sheathing is required, whereas if it will be covered with wood shingles or shakes, then only 1 x 3 inch nailing strips will be needed (so the roof can "breathe.")

When we get to line 4710 we have completed our list of needed items for the structure. Now we go through the M(x,x) array and combine similar items (you might be using 10-foot 2x4's for both the exterior wall studs and the ceiling joists, for example.) Then, starting at line 4810, we sort the list into ascending order by item name.

In line 4940 we finally get around to getting some output from this whole affair. All the information we need (except for one item) is now in

the M(x,x) array. All we need to do is print it out and extend the unit price to the quantity for a total price per item. Note that on the studs, we need to take the length times the number of studs times the unit price in line 5110 to get the total stud price. While printing out individual items total prices, we accumulate that amount in variable TL, and this becomes the total price of the project in line 5150.

If we don't want printed output line 5170 takes us back to the main menu, otherwise, flow of the program goes right on to the following code and repeats what we just did on the screen to the printer.

The "Edit unit prices" routine follows in line 5430. This little routine lets you display the whole price list on the screen and then pick whatever line you want to change and change it. When you are done editing prices, the program sends control back to line 2390, which first saves the updated file and then sends control back to the main menu.

The last section of code, from line 5570 to 5680, prints the list of specifications for any project you have currently loaded on the line printer. See the accompanying figure for an example of our garage project.

The program will let you do any part of a larger project. Don't forget, if you are doing only interior walls, to specify wall height - it's still needed and will give unpredictable results if you leave it out. You can leave out the floor entirely in case you are going to have a concrete slab floor. Actually, the floor was the most difficult part of this program because there are so many different ways to cover a floor. So we took it to sub-floor sheathing and insulation and left it at that. Keep in mind also the things that the program does not cover, like paint, nails, trim, doors and windows. Unless you are considering a large garage door, the number of studs and cover material will not be very much affected by the addition of windows and doors.

So there it is. Now you can play "what if?" and

find comparative prices for your next backyard (or front yard) project. A trip to your local lumber yard should get you all the unit prices you will

need. The ones we have shown in our garage sample figures are probably not valid in your area.

Frame.Bas for MS DOS Machines and Tandy Models II/IV

```
100 REM * Frame.Bas * Building estimate program written for
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 `CLEAR 5000 ` use only if you need to clear string space.
170 DIM R(30,5),A$(30),B$(22),C$(20),D$(20),E$(20),L(20),M(20,6),
    P(20)
180 DEF FNM(M)=M MOD 24 <> 0 ` nearest two ft. length function
190 ON ERROR GOTO 5710
200 `
210 DATA length ft.in,width ft.in,wall hgt ft.in
220 DATA length excluded wall ft.in,total length interior walls
230 DATA roof pitch degrees,floor joists,floor joist centers
240 DATA sub-floor sheathing,floor felt paper,floor insulation
250 DATA exterior studs,ext. studs on center,# top plates
260 DATA # bottom plates,interior studs,int.studs on center
270 DATA ceiling joists,ceiling joists on center,ceiling joist
    sheathing
280 DATA roof spans,rafters on center,roof felt paper,roof finish
    cover
290 DATA ext sheathing type,exterior felt paper
300 DATA siding,insulate outside walls,drywall
310 FOR I=1 TO 29:READ A$(I):NEXT I
320 `
330 DATA None,ft.in,2x4,2x6,2x8,1/2 in.ply,3/4 in.ply,16 in.,24
    in.
340 DATA 1x6 board,yes,no,one,two,degrees,wood shingle,composition
    shingle
350 DATA 1x6 siding,1/2 in.drywall,5/8 in.drywall,width,length
360 FOR I=0 TO 21:READ B$(I):NEXT I
370 `
380 DATA stud,stud,stud,board,board,siding,plywood,plywood
390 DATA drywall,drywall,felt paper,insulation,insulation
400 DATA preformed rafter,wood shingle,composition shingle
410 FOR I=1 TO 16:READ C$(I):NEXT I
420 `
430 DATA 2x4,2x6,2x8,1x3,1x6,1x6,3/4"x4'x8',1/2"x4'x8',1/2"x4'x8'
```

```

440 DATA 5/8"x4'x8',36"x160',15",23",-,-,-
450 FOR I=1 TO 16:READ D$(I):NEXT I
460 `
470 DATA ft.,ft.,ft.,ft.,ft.,ft.,panel,panel,panel,panel,roll
480 DATA sq.ft.,sq.ft.,each,pack,square
490 FOR I=1 TO 16:READ E$(I):NEXT I
500 `
510 GOSUB 2250 ` read in the prices file
520 IF DATE$="" THEN INPUT "What is today's date";DT$ ELSE
    DT$=DATE$
530 CLS
540 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
550 PRINT"      B U I L D I N G   E S T I M A T E   P R O G R A M
560 PRINT"      cost and materials to frame and cover
570 PRINT STRING$(60,45)
580 PRINT
590 PRINT TAB(10)"1- Create new specifications
600 PRINT TAB(10)"2- SAVE specifications to disk
610 PRINT TAB(10)"3- LOAD specifications from disk
620 PRINT TAB(10)"4- EDIT specifications
630 PRINT TAB(10)"5- Calculate and display material & costs
640 PRINT TAB(10)"6- Create new unit prices ";H$
650 PRINT TAB(10)"7- EDIT unit prices
660 PRINT TAB(10)"8- Print out the specifications
670 PRINT TAB(10)"9- Quit
680 H$=""
690 INPUT "The number of your choice ";XX
700 ON XX GOTO 730,2000,2120,2480,2740,2340,5450,5590,720
710 GOTO 690
720 CLS:END
730 CLS:I=1
740 PRINT"Please use a period to separate feet and inches (ft.in)":
    PRINT
750 PRINT "Dimensions":PRINT
760 INPUT"overall length (enter ft.in)";N
770 R(I,1)=N:R(I,2)=1:I=I+1
780 INPUT"overall width (enter ft.in)";N
790 R(I,1)=N:R(I,2)=1:I=I+1
800 INPUT"wall height (enter ft.in)";N
810 R(I,1)=N:R(I,2)=1:I=I+1
820 INPUT"Length of excluded wall (ft.in)";N
830 R(I,1)=N:R(I,2)=1:I=I+1
840 INPUT"Total length of interior walls (ft.in)";N
850 R(I,1)=N:R(I,2)=1:I=I+1
860 INPUT "How many degrees pitch to roof (20 to 45) ";N
870 R(I,1)=N:R(I,2)=1:I=I+1
880 CLS
890 PRINT"Floor":PRINT
900 PRINT A$(7), "choices are:"
910 PRINT "0 ";B$(0),"3 ";B$(3),"4 ";B$(4)

```

```

920 INPUT "Number of your choice ";N
930 R(I,1)=N:R(I,2)=0:R(I,3)=3:R(I,4)=4:I=I+1
940 PRINT A$(8), "choices are:"
950 PRINT "0 ";B$(0),"7 ";B$(7),"8 ";B$(8)
960 INPUT "Number of your choice ";N
970 R(I,1)=N:R(I,2)=0:R(I,3)=7:R(I,4)=8:I=I+1
980 PRINT A$(9), "choices are:"
990 PRINT "0 ";B$(0),"6 ";B$(6),"9 ";B$(9)
1000 INPUT "Number of your choice ";N
1010 R(I,1)=N:R(I,2)=0:R(I,3)=6:R(I,4)=9:I=I+1
1020 PRINT A$(10), "choices are:"
1030 PRINT "10 ";B$(10),"11 ";B$(11)
1040 INPUT "Number of your choice ";N
1050 R(I,1)=N:R(I,2)=10:R(I,3)=11:I=I+1
1060 PRINT A$(11), "choices are:"
1070 PRINT "10 ";B$(10),"11 ";B$(11)
1080 INPUT "Number of your choice ";N
1090 R(I,1)=N:R(I,2)=10:R(I,3)=11:I=I+1
1100 `
1110 CLS:PRINT "Exterior walls":PRINT
1120 PRINT A$(12), "choices are:"
1130 PRINT "0 ";B$(0),"2 ";B$(2),"3 ";B$(3)
1140 INPUT "Number of your choice ";N
1150 R(I,1)=N:R(I,2)=0:R(I,3)=2:R(I,4)=3:I=I+1
1160 PRINT A$(13), "choices are:"
1170 PRINT "0 ";B$(0),"7 ";B$(7),"8 ";B$(8)
1180 INPUT "Number of your choice ";N
1190 R(I,1)=N:R(I,2)=0:R(I,3)=7:R(I,4)=8:I=I+1
1200 PRINT A$(14), "choices are:"
1210 PRINT "12 ";B$(12),"13 ";B$(13)
1220 INPUT "Number of your choice ";N
1230 IF N=0 THEN N=12
1240 R(I,1)=N:R(I,2)=12:R(I,3)=13:I=I+1
1250 PRINT A$(15), "choices are:"
1260 PRINT "12 ";B$(12),"13 ";B$(13)
1270 INPUT "Number of your choice ";N
1280 IF N=0 THEN N=12
1290 R(I,1)=N:R(I,2)=12:R(I,3)=13:I=I+1
1300 `
1310 CLS:PRINT "Interior walls":PRINT
1320 PRINT A$(16), "choices are:"
1330 PRINT "0 ";B$(0),"2 ";B$(2),"3 ";B$(3)
1340 INPUT "Number of your choice ";N
1350 R(I,1)=N:R(I,2)=0:R(I,3)=2:R(I,4)=3:I=I+1
1360 PRINT A$(17), "choices are:"
1370 PRINT "7 ";B$(7),"8 ";B$(8)
1380 INPUT "Number of your choice ";N
1390 R(I,1)=N:R(I,2)=7:R(I,3)=8:I=I+1
1400 `

```

```

1410 CLS:PRINT''Ceiling joists'':PRINT
1420 PRINT A$(18),'choices are:''
1430 PRINT ``0 ``;B$(0),'2 ``;B$(2),'3 ``;B$(3)
1440 INPUT''Number of your choice ``;N
1450 R(I,1)=N:R(I,2)=0:R(I,3)=2:R(I,4)=3:I=I+1
1460 PRINT A$(19),'choices are:''
1470 PRINT ``7 ``;B$(7),'8 ``;B$(8)
1480 INPUT''Number of your choice ``;N
1490 R(I,1)=N:R(I,2)=7:R(I,3)=8:I=I+1
1500 PRINT A$(20),'choices are:''
1510 PRINT ``0 ``;B$(0),'5 ``;B$(5),'6 ``;B$(6),'9 ``;B$(9)
1520 INPUT''Number of your choice ``;N
1530 R(I,1)=N:R(I,2)=0:R(I,3)=5:R(I,4)=6:R(I,5)=9:I=I+1
1540 `
1550 CLS:PRINT''The Roof'':PRINT
1560 PRINT A$(21),'choices are:''
1570 PRINT ``20 ``;B$(20),'21 ``;B$(21)
1580 INPUT''Number of your choice ``;N
1590 R(I,1)=N:R(I,2)=20:R(I,3)=21:I=I+1
1600 PRINT A$(22),'choices are:''
1610 PRINT ``7 ``;B$(7),'8 ``;B$(8)
1620 INPUT''Number of your choice ``;N
1630 R(I,1)=N:R(I,2)=7:R(I,3)=8:I=I+1
1640 PRINT A$(23),'choices are:''
1650 PRINT ``10 ``;B$(10),'11 ``;B$(11)
1660 INPUT''Number of your choice ``;N
1670 R(I,1)=N:R(I,2)=10:R(I,3)=11:I=I+1
1680 PRINT A$(24),'choices are:''
1690 PRINT ``15 ``;B$(15),'16 ``;B$(16)
1700 INPUT''Number of your choice ``;N
1710 R(I,1)=N:R(I,2)=15:R(I,3)=16:I=I+1
1720 `
1730 CLS:PRINT ``Exterior sheathing'':PRINT
1740 PRINT A$(25),'choices are:''
1750 PRINT ``5 ``;B$(5),'6 ``;B$(6),'9 ``;B$(9)
1760 INPUT''Number of your choice ``;N
1770 R(I,1)=N:R(I,2)=5:R(I,3)=6:R(I,4)=9:I=I+1
1780 PRINT A$(26),'choices are:''
1790 PRINT ``10 ``;B$(10),'11 ``;B$(11)
1800 INPUT''Number of your choice ``;N
1810 R(I,1)=N:R(I,2)=10:R(I,3)=11:I=I+1
1820 PRINT A$(27),'choices are:''
1830 PRINT ``15 ``;B$(15),'17 ``;B$(17)
1840 INPUT''Number of your choice ``;N
1850 R(I,1)=N:R(I,2)=15:R(I,3)=17:I=I+1
1860 `
1870 CLS:PRINT''Interior covering'':PRINT
1880 PRINT A$(28),'choices are:''
1890 PRINT ``10 ``;B$(10),'11 ``;B$(11)

```

```

1900 INPUT''Number of your choice '';N
1910 R(I,1)=N:R(I,2)=10:R(I,3)=11:I=I+1
1920 PRINT A$(29),''choices are:''
1930 PRINT ``18 ``;B$(18),''19 ``;B$(19)
1940 INPUT''Number of your choice '';N
1950 R(I,1)=N:R(I,2)=18:R(I,3)=19:I=I+1
1960 GOTO 530
1970 `
1980 `save specifications to disk routine
1990 `
2000 INPUT''Please name the disk file (8 chars or less)'';FF$
2010 OPEN ``O'',1,FF$
2020 FOR I=1 TO 29
2030     FOR J=1 TO 5
2040         PRINT #1, R(I,J)
2050     NEXT J
2060 NEXT I
2070 CLOSE 1
2080 GOTO 530
2090 `
2100 `load specifications from file routine
2110 `
2120 INPUT''What file name do you want to load '';FF$
2130 OPEN ``I'',1,FF$
2140 FOR I=1 TO 29
2150     IF EOF(1) THEN 2200
2160     FOR J=1 TO 5
2170         INPUT #1, R(I,J)
2180     NEXT J
2190 NEXT I
2200 CLOSE 1
2210 GOTO 530
2220 `
2230 `read the unit price file
2240 `
2250 OPEN ``I'',1,``PRICES''
2260 FOR I=1 TO 16
2270     INPUT #1,P(I)
2280 NEXT I
2290 CLOSE 1
2300 RETURN
2310 `
2320 `Enter new unit prices routine
2330 `
2340 CLS
2350 FOR I=1 TO 16
2360     PRINT C$(I);'' ``;D$(I);:PRINT'' Enter price per ``;E$(I);:
        INPUT P(I)
2370 NEXT I
2380 `

```

```

2390 OPEN ``O'',1,``PRICES''
2400 FOR I=1 TO 16
2410   PRINT #1,P(I)
2420 NEXT I
2430 CLOSE 1
2440 GOTO 530
2450 `
2460 `edit the specifications routine
2470 `
2480 CLS
2490 IF R(7,4)<>0 THEN 2520
2500 PRINT``You must have input a file or new data to edit.``
2510 PRINT``press any key for main menu``;:INPUT X:GOTO 530
2520 FOR I=1 TO 6
2530   PRINT A$(I);`` ``;R(I,1)
2540   INPUT``Change to, or enter to keep ``;N$
2550   IF N$=```` THEN ELSE R(I,1)=VAL(N$)
2560 NEXT I
2570 FOR I=7 TO 29
2580   CLS
2590   PRINT TAB(10);A$(I)
2600   PRINT ``(``R(I,2)``) ``;B$(R(I,2))
2610   PRINT ``(``R(I,3)``) ``;B$(R(I,3))
2620   PRINT ``(``R(I,4)``) ``;B$(R(I,4))
2630   PRINT ``(``R(I,5)``) ``;B$(R(I,5))
2640   PRINT``You chose ``;``(``R(I,1)``) ``;B$(R(I,1))
2650   PRINT
2660   INPUT``Change to #, or enter to keep, or Q to quit``;XX$
2670   IF XX$=``Q`` OR XX$=``q`` THEN 2700
2680   IF XX$<>```` THEN R(I,1)=VAL(XX$)
2690 NEXT I
2700 GOTO 530
2710 `
2720 `first, clear the m array then convert ft to inches
2730 `
2740 PRINT``Calculating....``
2750 FOR I=1 TO 20
2760   FOR J=1 TO 6
2770     M(I,J)=0
2780   NEXT J
2790 NEXT I
2800 `
2810 L=(INT(R(1,1))*12)+(R(1,1)-INT(R(1,1)))*10 `length in inches
2820 W=(INT(R(2,1))*12)+(R(2,1)-INT(R(2,1)))*10 `width in inches
2830 H=(INT(R(3,1))*12)+(R(3,1)-INT(R(3,1)))*10 `wall height in
inches
2840 L1=(INT(R(4,1))*12)+(R(4,1)-INT(R(4,1)))*10 `excluded wall
length in inches
2850 L2=(INT(R(5,1))*12)+(R(5,1)-INT(R(5,1)))*10 `interior wall

```

```

length in inches
2860 IF R(1,1)=0 THEN 2910
2870 XA=((L1/12)*(H/12)) `area in sq ft of excluded wall
2880 FA=((L/12)*(W/12)) `floor (or ceiling) area in sq ft.
2890 LA=((L/12)*(H/12)) `area in sq ft of one long wall
2900 WA=((W/12)*(H/12)) `area in sq ft of one short wall
2910 IA=((L2/12)*(H/12)) `area of interior walls in sq ft.
2920 `
2930 `figure the roof angles and the roof and gable area
2940 `
2950 IF R(21,1)=0 THEN 3080
2960 IF R(21,1)=20 THEN B=.5*W
2970 IF R(21,1)=21 THEN B=.5*L
2980 Q=TAN(R(6,1)/(180/3.1416))*B `height of roof triangle in
inches
2990 HY=SQR(Q^2+B^2) `hypotenuse, or 1/2 roof in
inches
3000 RF=HY+18:RF=INT(RF/12)+1 `rf is roof in ft. allow for 18 in
overhang
3010 GA=INT((Q/12)*(B/12))*2 `total gable area on both ends in
sq.ft.
3020 RR=RF*((L+24)/12) `half the roof area in sq ft allow for
extra ft at ends
3030 RT=RR*2 `total roof area in sq ft.
3040 RX=2*(LA+WA)+GA-XA ` total outside area in sq ft excluding
roof
3050 `
3060 `find number and length of exterior studs
3070 `
3080 IF R(13,1)=7 THEN C1=16 ELSE IF R(13,1)=8 THEN C1=24
3090 L=L-(L1*.5)
3100 IF R(14,1)=12 THEN H1=INT(H-1.5):P=P+1
3110 IF R(14,1)=13 THEN H1=INT(H-3):P=P+2
3120 IF R(15,1)=12 THEN H1=INT(H-1.5):P=P+4
3130 IF R(15,1)=13 THEN H1=INT(H-3):P=P+8
3140 IF R(12,1)=0 THEN 3250
3150 SA=((L/C1)+3)*2:SB=((W/C1)+3)*2
3160 TS=INT(SA+SB) `total studs
3170 FOR I=1 TO 25:IF FNM(H1) THEN H1=H1+1:NEXT I
3180 LS=H1/12 `length of studs in even 2 ft.
3190 H1=0
3200 IF R(12,1)=2 THEN M(1,2)=1:M(1,5)=1 ELSE IF R(12,1)=3 THEN
M(1,2)=2:M(1,5)=2
3210 M(1,1)=1:M(1,3)=LS:M(1,4)=TS:M(1,6)=1
3220 `
3230 `find number and length of interior studs
3240 `
3250 IF R(16,1)=0 THEN 3390
3260 IF R(14,1)=12 THEN H1=INT(H-1.5)
3270 IF R(14,1)=13 THEN H1=INT(H-3)

```



```

3280 IF R(15,1)=12 THEN H1=INT(H-1.5)
3290 IF R(15,1)=13 THEN H1=INT(H-3)
3300 IF R(17,1)=7 THEN C1=16 ELSE IF R(17,1)=8 THEN C1=24
3310 SA=((L2/C1)+3):TS=INT(SA) ' total studs
3320 FOR I=1 TO 25:IF FNM(H1) THEN H1=H1+1:NEXT I
3330 LS=H1/12 'length of studs in even 2 ft.
3340 IF R(16,1)=2 THEN M(2,2)=1:M(2,5)=1 ELSE IF R(16,1)=3 THEN
M(2,5)=2:M(2,5)=2
3350 M(2,1)=1:M(2,3)=LS:M(2,4)=TS:M(2,6)=1
3360 '
3370 'find number and length of floor joists
3380 '
3390 IF R(7,1)=0 THEN 3510
3400 IF R(8,1)=7 THEN C1=16 ELSE IF R(8,1)=8 THEN C1=24
3410 SA=(L/C1)+3:TS=INT(SA) ' total joists
3420 W1=W
3430 FOR I=1 TO 25:IF FNM(W1) THEN W1=W1+1:NEXT I
3440 LS=W1/12 'length of joists in even 2 ft.
3450 W1=0
3460 IF R(7,1)=3 THEN M(3,2)=2:M(3,5)=2 ELSE IF R(7,1)=4 THEN M(3,
2)=3:M(3,5)=3
3470 M(3,1)=1:M(3,3)=LS:M(3,4)=TS:M(3,6)=1
3480 '
3490 'find number and length of ceiling joists
3500 '
3510 IF R(18,1)=0 THEN 3620
3520 IF R(18,1)=7 THEN C1=16 ELSE IF R(18,1)=8 THEN C1=24
3530 SA=(L/C1)+3:TS=INT(SA) ' total ceiling joists
3540 W1=W
3550 FOR I=1 TO 25:IF FNM(W1) THEN W1=W1+1:NEXT I
3560 LS=W1/12 'length of ceiling joists in even 2 ft.
3570 IF R(18,1)=2 THEN M(4,2)=1:M(4,5)=1 ELSE IF R(18,1)=3 THEN
M(4,2)=2:M(4,5)=2
3580 M(4,1)=1:M(4,3)=LS:M(4,4)=TS:M(4,6)=1
3590 '
3600 'find number and length of plate studs for exterior walls
3610 '
3620 IF R(1,1)=0 OR R(2,1)=0 THEN 3740
3630 P1=((L+W)*2)-L1 'one perimeter less excluded walls
3640 IF P=5 THEN P1=P1*2
3650 IF P=9 OR P=6 THEN P1=P1*3
3660 IF P=10 THEN P1=P1*4
3670 TS=INT((P1/144)+1)
3680 LS=12
3690 M(5,1)=1:M(5,2)=M(1,2):M(5,3)=LS:M(5,4)=TS:M(5,6)=1
3700 M(5,5)=M(1,2)
3710 '
3720 'find number and length of plate studs for interior walls
3730 '

```

```

3740 IF R(16,1)=0 THEN 3850
3750 IF P=5 THEN P1=L2*2
3760 IF P=6 OR P=9 THEN P1=L2*3
3770 IF P=10 THEN P1=L2*4
3780 TS=INT((P1/144)+1)
3790 LS=12
3800 M(6,1)=1:M(6,2)=M(2,2):M(6,3)=LS:M(6,4)=TS:M(6,6)=1
3810 M(6,5)=M(2,2)
3820 `
3830 `find sub-floor sheathing
3840 `
3850 IF R(9,1)=0 THEN 4050
3860 IF R(9,1)=6 THEN TS=INT(FA/32)+1:M(7,1)=7:M(7,2)=7:M(7,4)=TS:
M(7,5)=7:M(7,6)=7
3870 IF R(9,1)=9 THEN TS=INT(FA*2.3):M(7,1)=5:M(7,2)=5:M(7,4)=TS:
M(7,5)=5:M(7,6)=5
3880 `
3890 `find floor felt paper needed
3900 `
3910 IF R(10,1)=0 OR R(10,1)=11 THEN 3970
3920 TS=INT(FA/500)+1
3930 M(8,1)=11:M(8,2)=11:M(8,4)=TS:M(8,5)=11:M(8,6)=11
3940 `
3950 `find floor insulation needed
3960 `
3970 IF R(11,1)=0 OR R(11,1)=11 THEN 4050
3980 TS=INT(FA*.9)
3990 IF R(8,1)=7 THEN M(9,2)=12:M(9,5)=12:M(9,6)=12
4000 IF R(8,1)=8 THEN M(9,2)=13:M(9,5)=13:M(9,6)=13
4010 M(9,1)=12:M(9,4)=TS
4020 `
4030 `find ceiling joist sheathing (top of joists)
4040 `
4050 IF R(20,1)=0 THEN 4120
4060 IF R(20,1)=5 OR R(20,1)=6 THEN TS=INT(FA/32)+1:M(10,4)=TS:
M(10,6)=8
4070 IF R(20,1)=9 THEN TS=INT(FA*2.3):M(10,1)=5:M(10,2)=5:M(10,
4)=TS:M(10,5)=5:M(10,6)=5
4080 IF R(20,1)=5 THEN M(10,1)=8:M(10,2)=8:M(10,5)=8:M(10,6)=8
ELSE IF R(20,1)=6 THEN M(10,1)=7:M(10,2)=7:M(10,5)=7:M(10,
6)=7
4090 `
4100 `find number of preformed rafters needed
4110 `
4120 IF R(22,1)=0 THEN 4200
4130 IF R(22,1)=7 THEN C1=16 ELSE IF R(22,1)=8 THEN C1=24
4140 IF R(21,1)=20 THEN M(11,4)=INT(((L+24)/C1)+2):M(11,
3)=INT((W/12)+3)
4150 IF R(21,1)=21 THEN M(11,4)=INT(((W+24)/C1)+2):M(11,

```

```

3)=INT((L/12)+3)
4160 M(11,1)=14:M(11,5)=14:M(11,6)=14
4170 '
4180 'find amount of roof felt paper needed
4190 '
4200 IF R(23,1)=0 OR R(23,1)=11 THEN 4260
4210 TS=INT(RT/500)+1
4220 M(12,1)=11:M(12,2)=11:M(12,4)=TS:M(12,5)=11:M(12,6)=11
4230 '
4240 'find the amount of finish roof cover
4250 '
4260 IF R(24,1)=0 THEN 4380
4270 IF R(24,1)=16 THEN 4300
4280 IF R(24,1)=15 THEN M(13,1)=4:M(13,2)=4:M(13,4)=INT(RT*1.2):
M(13,5)=4:M(13,6)=4
4290 M(14,1)=15:M(14,4)=INT(RT/25)+1:M(14,5)=15:M(14,6)=15:GOTO
4380
4300 IF R(24,1)=16 THEN M(14,1)=16:M(14,4)=INT(RT/100):M(14,5)=16:
M(14,6)=16
4310 IF R(25,1)=5 OR R(25,1)=6 THEN M(13,4)=INT(RT/32)+1:M(13,1)=7:
M(13,6)=7
4320 IF R(25,1)=9 THEN M(13,4)=INT(RT*2.3):M(13,1)=5:M(13,5)=5:
M(13,6)=5
4330 IF R(25,1)=5 THEN M(13,1)=8:M(13,2)=8:M(13,5)=8
4340 IF R(25,1)=6 THEN M(13,1)=7:M(13,2)=7:M(13,5)=7
4350 '
4360 'find exterior sheathing needed
4370 '
4380 IF R(25,1)=0 THEN 4450
4390 IF R(25,1)=5 THEN M(15,1)=8:M(15,2)=8:M(15,4)=INT(RX/32)+1:
M(15,5)=8:M(15,6)=8
4400 IF R(25,1)=6 THEN M(15,1)=7:M(15,2)=7:M(15,4)=INT(RX/32)+1:
M(15,5)=7:M(15,6)=7
4410 IF R(25,1)=9 THEN M(15,1)=5:M(15,2)=5:M(15,4)=INT(RX*2.3):
M(15,5)=5:M(15,6)=5
4420 '
4430 'find exterior felt paper needed
4440 '
4450 IF R(26,1)=0 OR R(26,1)=11 THEN 4500
4460 M(16,1)=11:M(16,2)=11:M(16,4)=INT(RX/500)+1:M(16,5)=11:M(16,
6)=11
4470 '
4480 'find exterior siding needed
4490 '
4500 IF R(27,1)=0 THEN 4560
4510 IF R(27,1)=15 THEN M(17,1)=15:M(17,4)=INT(RX/25):M(17,5)=15:
M(17,6)=15
4520 IF R(27,1)=17 THEN M(17,1)=6:M(17,2)=6:M(17,4)=INT(RX*2.3):
M(17,5)=6:M(17,6)=6

```

```

4530 `
4540 `find insulation for ext walls and ceiling
4550 `
4560 IF R(28,1)=0 OR R(28,1)=11 THEN 4640
4570 M(18,1)=13:M(18,2)=13:M(18,4)=INT(2*(LA+WA)-XA):M(18,5)=13:
    M(18,6)=13
4580 IF R(12,1)=7 THEN M(18,1)=12:M(18,2)=12:M(18,5)=12:M(18,6)=12
4590 M(19,1)=13:M(19,2)=13:M(19,4)=INT(FA*.9):M(19,5)=13:M(19,
    6)=13
4600 IF R(19,1)=7 THEN M(19,1)=12:M(19,2)=12:M(19,5)=12:M(19,6)=12
4610 `
4620 `find the amount of drywall needed
4630 `
4640 IF R(29,1)=0 THEN 4730
4650 T1=(2*(LA+WA)-XA)
4660 T2=2*IA
4670 TS=T1+T2+FA
4680 IF R(29,1)=18 THEN M(20,1)=9:M(20,2)=9:M(20,4)=INT(TS/32)+1:
    M(20,5)=9:M(20,6)=9
4690 IF R(29,1)=19 THEN M(20,1)=10:M(20,2)=10:M(20,4)=INT(TS/32)+1:
    M(20,5)=10:M(20,6)=10
4700 `
4710 `combine identical items in the list
4720 `
4730 FOR I=1 TO 20
4740   FOR J=I+1 TO 19
4750     IF M(I,1)=M(J,1) AND M(I,2)=M(J,2) AND M(I,3)=M(J,3) THEN
        M(J,4)=M(J,4)+M(I,4):M(I,1)=0
4760   NEXT J
4770 NEXT I
4780 `
4790 `sort the list into ascending order by item
4800 `
4810 FOR I=1 TO 19
4820   L=I+1
4830   IF C$(M(I,1))=<C$(M(L,1)) THEN 4910
4840   SWAP M(I,1),M(L,1)
4850   SWAP M(I,2),M(L,2)
4860   SWAP M(I,3),M(L,3)
4870   SWAP M(I,4),M(L,4)
4880   SWAP M(I,5),M(L,5)
4890   SWAP M(I,6),M(L,6)
4900   F=1
4910 NEXT I
4920 IF F=1 THEN F=0:GOTO 4810
4930 `
4940 ` print out the results on the screen
4950 `
4960 CLS

```

```

4970 TL=0
4980 PRINT''Materials and prices for Project: '';FF$;TAB(60);DT$
4990 PRINT STRING$(70,45)
5000 PRINT''Item'';TAB(22);''Size'';TAB(34);''Length'';TAB(42);''Qty'';
    TAB(48);''Price'';TAB(56);''Per'';TAB(64);''Total''
5010 PRINT STRING$(70,45)
5020 FOR I=1 TO 20
5030     IF C$(M(I,1))=''' THEN 5140
5040     PRINT C$(M(I,1));TAB(22);
5050     PRINT D$(M(I,2));TAB(34);
5060     PRINT M(I,3);TAB(42);
5070     PRINT M(I,4);TAB(48);
5080     PRINT P(M(I,5));TAB(56);
5090     PRINT E$(M(I,6));TAB(62);
5100     ST=M(I,4)*P(M(I,5))
5110     IF C$(M(I,1))='''stud'' THEN ST=M(I,3)*M(I,4)*P(M(I,5))
5120     PRINT USING ''##,###.##'';ST
5130     TL=TL+ST
5140 NEXT I
5150 PRINT:PRINT TAB(51); ''Total is '';USING ''$###,###.##'';TL
5160 INPUT''Do you want hardcopy of this (y/n)'';XX$
5170 IF XX$='''N'' OR XX$='''n'' THEN 530
5180 '
5190 'printer output goes here
5200 '
5210 TL=0
5220 LPRINT''Prices and materials for Project: '';FF$;TAB(60);DT$
5230 LPRINT STRING$(70,45)
5240 LPRINT''Item'';TAB(22);''Size'';TAB(34);''Length'';TAB(42);''Qty'';
    TAB(48);''Price'';TAB(56);''Per'';TAB(64);''Total''
5250 LPRINT STRING$(70,45)
5260 FOR I=1 TO 20
5270     IF C$(M(I,1))=''' THEN 5380
5280     LPRINT C$(M(I,1));TAB(22);
5290     LPRINT D$(M(I,2));TAB(34);
5300     LPRINT M(I,3);TAB(42);
5310     LPRINT M(I,4);TAB(48);
5320     LPRINT P(M(I,5));TAB(56);
5330     LPRINT E$(M(I,6));TAB(62);
5340     ST=M(I,4)*P(M(I,5))
5350     IF C$(M(I,1))='''stud'' THEN ST=M(I,3)*M(I,4)*P(M(I,5))
5360     LPRINT USING ''##,###.##'';ST
5370     TL=TL+ST
5380 NEXT I
5390 LPRINT'' '';LPRINT TAB(51); ''Total is '';USING ''$###,###.##'';TL
5400 LPRINT CHR$(12) ' give a printer page eject
5410 GOTO 530
5420 '
5430 'edit the prices routine
5440 '

```

Changes for Handy Model III

```

5450 CLS:PRINT TAB(15);''Change Unit Prices
5460 FOR I=1 TO 16
5470   PRINT I;''  '';C$(I);TAB(26);D$(I);TAB(38);''price per '';
      TAB(48);E$(I);TAB(56);P(I)
5480 NEXT I
5490 PRINT
5500 INPUT''Number of item to change is (or 0 for none) '';XX
5510 IF XX=0 THEN 2390
5520 INPUT''Change price to '';XY
5530 P(XX)=XY
5540 INPUT''Change more prices (y/n)'';XX$
5550 IF XX$='Y' OR XX$='y' THEN 5500 ELSE 2390
5560 `
5570 `print out the specifications on the printer routine
5580 `
5590 LPRINT ``Project: ``;FF$ TAB(30);''Specifications ``;TAB(60);DT$
5600 LPRINT STRING$(70,45)
5610 FOR I=1 TO 6
5620   LPRINT A$(I);TAB(30);R(I,1)
5630 NEXT I
5640 FOR I=7 TO 29
5650   LPRINT A$(I);TAB(30);B$(R(I,1))
5660 NEXT I
5670 LPRINT CHR$(12)
5680 GOTO 530
5690 `
5700 `error trap
5710 IF ERR <>53 THEN ON ERROR GOTO 0
5720 `IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0 `BASIC prior to 5.0
5730 IF XX<>3 THEN H$=' ' <-----DO THIS FIRST'' :GOTO 530
5740 IF XX=3 THEN PRINT''There is no file called ``;FF$
5750 INPUT''Press ENTER to continue ``;XX:GOTO 530
5760 END `of program

```

Changes for Tandy Model I/III

Changed->160 CLEAR 5000 ` use only if you need to clear string space.

Changed->170

R(30,5),A\$(30),B\$(22),C\$(20),D\$(20),E\$(20),L(20),M(21,6),P(20)

DIM

Changed->180 DEF FNM(M)=(INT(M-24)*INT(M/24))<>0

Changed->2990 HY=SQR(Q[2]+(B[2])

Changed->4840 T1=M(I,1):M(I,1)=M(L,1):M(L,1)=T1

Changed->4850 T1=M(I,2):M(I,2)=M(L,2):M(L,2)=T1

Changed->4860 T1=M(I,3):M(I,3)=M(L,3):M(L,3)=T1

Changed->4870 T1=M(I,4):M(I,4)=M(L,4):M(L,4)=T1

Changed->4880 T1=M(I,5):M(I,5)=M(L,5):M(L,5)=T1

Changed->4890 T1=M(I,6):M(I,6)=M(L,6):M(L,6)=T1

Changed->5710 `IF ERR <>53 THEN ON ERROR GOTO 0

Changed->5720 IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0 `BASIC prior to 5.0

Materials and prices for Project: garage

01-01-1980

| Item | Size | Length | Qty | Price | Per | Total |
|------------------|------------|--------|------|-------|--------|--------|
| board | 1x3 | 0 | 936 | .17 | ft. | 159.12 |
| board | 1x6 | 0 | 2001 | .21 | ft. | 420.21 |
| drywall | 5/8"x4'x8' | 0 | 48 | 3.05 | panel | 146.40 |
| felt paper | 36"x160' | 0 | 3 | 8.95 | roll | 26.85 |
| insulation | 15" | 0 | 432 | .19 | sq.ft. | 82.08 |
| insulation | 23" | 0 | 1136 | .22 | sq.ft. | 249.92 |
| plywood | 3/4"x4'x8' | 0 | 16 | 23.5 | panel | 376.00 |
| plywood | 1/2"x4'x8' | 0 | 16 | 21.5 | panel | 344.00 |
| preformed rafter | | 23 | 15 | 32.8 | each | 492.00 |
| siding | 1x6 | 0 | 2001 | .29 | ft. | 580.29 |
| stud | 2x4 | 8 | 91 | .21 | ft. | 152.88 |
| stud | 2x8 | 20 | 21 | .38 | ft. | 159.60 |
| stud | 2x4 | 20 | 21 | .21 | ft. | 88.20 |
| stud | 2x4 | 12 | 37 | .21 | ft. | 93.24 |
| wood shingle | | 0 | 32 | 13.4 | pack | 428.80 |

Total is \$ 3,799.59

Project: garage Specifications

| | | |
|-----------------------------|----------------|-------|
| length ft.in | 24 | |
| width ft.in | 20 | |
| wall hgt ft.in | 8 | |
| length excluded wall ft.in | 0 | |
| total length interior walls | 20 | |
| roof pitch degrees | 40 | |
| floor joists | 2x8 | |
| floor joist centers | 16 in. | |
| sub-floor sheathing | 3/4 in.ply | |
| floor felt paper | yes | |
| floor insulation | yes | |
| exterior studs | 2x4 | |
| ext. studs on center | 16 in. | .21 |
| # top plates | two | .32 |
| # bottom plates | two | .38 |
| interior studs | 2x4 | .17 |
| int.studs on center | 24 in. | .21 |
| ceiling joists | 2x4 | .29 |
| ceiling joists on center | 24 in. | 23.5 |
| ceiling joist sheathing | 1/2 in.ply | 21.5 |
| roof spans | width | 2.86 |
| rafters on center | 24 in. | 3.05 |
| roof felt paper | no | 8.95 |
| roof finish cover | wood shingle | .19 |
| ext sheathing type | 1x6 board | .22 |
| exterior felt paper | yes | 32.8 |
| siding | 1x6 siding | 13.4 |
| insulate outside walls | yes | 24.49 |
| drywall | 5/8 in.drywall | |

Below is a dump of the file called "Prices." To the left are the specifications for an imaginary garage. The figure above is the final output for the garage. You can use these same figures to check that you have entered the program correctly. You should get the same answers as shown here.

Trust.Bas

computerize your loan payments booklet

Staff Project. This program was written at the request of Mr. Richard L. Wright, of Buena Park, California. It takes the place of that little book that is sent back and forth between lender and borrower. In addition, it gives you selective printouts for any period of time, with totals.

Trust.Bas is another reader-requested program. It takes the place of that little green (or blue) booklet that borrowers and lenders send back and forth to each other with the payments on a loan. So, if you have the booklet already, what's the use of this program?

The program makes the calculations necessary to determine how much of a payment is going to interest and how much to the principal. It also lets you pick a portion of the payment history and gives totals only for that portion. This is handy at the end of the year if you are the lender, because you can then give the borrower a statement of what he paid that year. It's something he will need for his income tax preparation.

The program allows any amount of payment on a loan and figures it correctly. And, it even uses negative amortization if the payment is insufficient to cover the interest. That is, it adds the difference to the principal! This is the type of thing that was going on back in the late '70s, when so many farms were going belly-up. It's enough to warm the cold heart of any slum lord.

One of the problems that many computer programs will be running into shortly is making the gap between the year 1999 and the year 2000. To get around this problem, the program uses a date format that is slightly unconventional. The date is entered as YYYYMMDD. It is

also a double precision number in the program. This lets you find dates greater than and less than a given date.

The program will work with as many files (loans) as you wish. The files are sequential, and contain the initial principal amount and the interest rate, along with the record of payments made. Provision is made to print the payment history on the screen or on a line printer. When on the line printer, provision is made for paging the output into 36 payments per page (three years worth if payments are made monthly.)

When the program asks for starting and ending dates, those dates are inclusive. If you don't remember what dates are in the file and want to see them all you can enter 00000000 for a starting date and 99999999 for an ending date. The size of the file is adjusted so that each file can hold 30 years of monthly payments. This should cover most loans.

Program Details

The program starts with initialization in line 150. In line 160 we clear some string space for those machines using BASIC prior to version 5.0. The rest of us can leave this line remarked. In line 170 we dimension the variables that will hold the information on each payment. Here is where we adjust the size of the file to 30 years

(360 monthly payments). Line 180 is just a "calibration" line, so that you can get the spacing of the following lines properly. Since F1\$ through F4\$ will be used several times in the program, and at various places in the program, we set them up as formatted strings here, in lines 190 to 220. Putting them all in one nice chunk like this makes it easy to adjust the spacing should you want to change it. The last line of the initialization section, line 230, sets our error trap. The only error we will be trapping for is the "file not found" error.

The section of code from 250 to 340 loads the particular file we wish to work with. If the file does not yet exist, the error trap will be sprung and we will go to line 1150 to create the file. Let's go there now and see how a file is initialized.

If your BASIC is prior to version 5.0 you should remark line 1150 and un-remark line 1160. These lines "reset" the error trap so that normal errors will still show properly. Line 1180 tells you that the file you requested does not yet exist and line 1190 asks if you want to create it. If no, we just end the program so you can start all over. Otherwise, we open the file in line 1210 and then, in line 1220, ask what the beginning balance is. Then we ask for the interest rate in line 1240. Line 1250 prints these values to the file as BA and IR, respectively. We then close the file in 1260 and go to line 270. Let's go back there now.

In line 270 we open the file for input and immediately input the two values, BA and IR. Then the loop between 290 and 320 reads in (if there is any) the data. D#(I) is the date, P(I) is the payment amount, IA(I) is the interest portion of the payment, PI(I) is the principal portion of the payment and B(I) is the new balance amount. In line 330 we let N1 equal the number of array items in the file. We will need to know that number later on when we read the array. Line 340 simply closes the file, because we are now done with the file and all the data is in the appropriate arrays.

Lines 360 to 490 are the heading and the menu options that will print on the screen. We

use a simple ON XX GOTO statement in line 480 to take us to the appropriate sections of the program depending on our menu choice. Line 490 is there so that if the number we choose is less than one or more than four, the question will automatically be asked again.

The "quit" routine is at line 520. It simply closes any open files, clears the screen and ends the program, returning us to the BASIC ready prompt.

The "enter payments" routine is from 540 to 680. This is also where we figure out how much of the payment is interest and how much goes off the principal. First of all, line 550 asks for the date of the payment, and line 560 asks for the amount. These become D# and P, respectively. Now we have to worry about the very first payment on a loan because all the information we have at that point is the balance and the interest rate. We can check array element B(1) to see if it is zero to tell us if this is the very first payment. We do just that in line 580. If it is then T1 (the amount of interest in the payment) is calculated by taking the balance times the interest rate and dividing that by 12 to get the monthly amount. If it is not the first payment we have to use the last ending balance, so the ELSE portion of line 580 takes the last balance times the interest rate and divides by 12.

In line 590 we take care of the case where the payment is not enough to cover the interest portion. Here, we apply the whole payment amount to the interest portion, nothing to the principal payment portion, and add the difference between the interest amount and the payment to the principal balance! Awful, no?

In line 610 we again look to see if this is the very first payment and again make adjustments appropriately. Next we print the format strings we established early in the program, F1\$ and F2\$. Then, using the format string F3\$, we print the date, the payment amount, the amount going to interest, the amount going off the principal and the new balance amount. Now we have to add these values to the array (up till now they

were just printed on the screen). So, in line 650 we "bump" the array count by one to make a new place for the latest information and then in line 660 we put the information into the new array position (N1 plus one).

If there are more entries to make, line 680 will take us back to line 550 where we can enter more. Otherwise, normal program flow will take us to line 700, where we automatically save the freshly updated array to the disk file. Having done this, line 770 takes us back to line 360 to display the heading and menu options again. One interesting thing to note in saving the file (or reading it, for that matter) is that the file does not have to be identical items. Note that the first two items in the file are BA and IR, and then the array items follow. If you do this, you must always be sure to read in those two items first and when you save, save them first. You can actually put a variety of different sized items in a file this way.

The code from line 790 to 960 lets you select a range of entries to display on the screen and get totals for that portion. This is where you can get totals for a given year, for example. Whatever portion of the file you specify, by dates, will be totaled by this section of code. You are asked for the starting and ending dates. Remember that these dates are inclusive. We again use our format strings, F1\$ and F2\$ to print the heading on the screen, and then loop through the array pulling out dates that fit our specifications. In line 860 we set S1, S2 and S3 to zero, since they will be used as accumulating registers to keep a running track of the totals. Those totals will be for payment, interest amount and payment on principal amount. We print the entries selected on the screen using another of our format strings, F3\$, in line 890. When all selected entries have been printed, another format string, F4\$, will print our totals. The input statement in line 950 simply stops the program at this point so we can inspect our results before we continue. When we do continue, program flow takes us back to the main menu.

The last section of code from 980 to 1120 is almost a repeat of the section we just went

through. This time, however, it sends output to the printer. In addition, it provides for paging the output, 36 entries per page with a heading on each page. Like the previous section, running totals are kept, again using S1, S2 and S3, which are set to zero again in line 1000. Also in line 1000, we set the page counter, PA, to zero and the line counter, CT, as well. These counters will be used to tell us when a page is full (CT) and the page number we are on (PA). Note that we are not using a For...Next loop here as we did in the screen print section. Sometimes it is easier to use a home made loop in situations like this one. Jumping into and out of it is a little more graceful. The LPRINT CHR\$(12) in line 1110 is a page eject command to the printer, so that we can index to the next page. It appears again in 1120. The page eject in line 1110 is there so that the page will eject when we are done printing the report. The one in line 1120 is there to advance us to the next page during the printing of the report. Our loop is incremented in line 1110, where I=I+1 appears as the first item in that line. Our loop actually operates between lines 1080 and 1120. Note in line 1120 that if the line count is equal or less than 36 (the number of items to print on one page) we go back to line 1080 to get more items. Once we exceed 36 lines (items) on the page we advance the page counter by one, set the line counter (CT) back to one, issue a page eject and go back to line 1030 so that we can print the heading on the new page. This is why a For...Next would have been a bit cumbersome in this case.

Operating Notes

No provision has been made to prevent scrolling on the screen. The assumption being that you probably don't want more than one screen-full at a time there anyway. The printed output will provide neatly paged entries, however. The program does not have an edit mode. You only make two entries, the date and the payment amount. The rest of the information is calculated by the program itself. If you make a mistake in the payment amount, it will reflect errors in all the following entries. For that reason, enter your amounts carefully. The data file created by this

program can be called up and edited with any text editor, since it is nothing more than a pure ASCII file. At any time during the life of the loan that a balloon payment is called for, the "balance unpaid" amount will be the balloon payment amount.

You can have as many data files (loans) as you wish with this program. To get from one to another, you must exit and type "RUN" and then

give the name of the file you wish to use. This is possible because we have included the interest rate and beginning balance in the data file itself, instead of hard coding them into the program structure.

You can use this program if you are a borrower or a lender. We use it to check up on the lender of our own loans to make sure that each payment is being credited properly by the lender.

Trust.Bas listing for MS DOS and Tandy Models II and IV. Tandy Model I and III changes follow listing.

```

100 REM * Trust.Bas * payments on a deed of trust
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)1989 80-NW Publishing Inc. & placed in public domain.
140 '
150 'Initialization
160 'CLEAR 2000 ' use only with BASIC prior to ver. 5.0
170 DIM D#(360),P(360),IA(360),PI(360),B(360)
180 '012345678901234567890123456789012345678901234567890123456789
190 F1$='Date of      Amount      Credited on      Balance
200 F2$='Payment      Paid      Interest      Principal      Unpaid
210 F3$='#####      ####.##      ####.##      ####.##      #####.##
220 F4$='          $$#,#####.## $$#,#####.## $$#,#####.##
230 ON ERROR GOTO 1150
240 '
250 'load the data file, if it doesn't exist, make it
260 INPUT'What filename do you wish to work with `';FF$
270 OPEN `I',1,FF$
280 INPUT #1, BA,IR
290 FOR I=1 TO 361
300 IF EOF(1) THEN 330
310 INPUT #1, D#(I),P(I),IA(I),PI(I),B(I)
320 NEXT I
330 N1=I-1
340 CLOSE 1
350 '
360 CLS
370 PRINT STRING$(22,45);' The CodeWorks `';STRING$(23,45)
380 PRINT' RECORD OF PAYMENTS PROGRAM
390 PRINT' Records payment on a Deed of Trust

```

```

400 PRINT STRING$(60,45)
410 PRINT
420 PRINT TAB(15);''1 - Enter payments
430 PRINT TAB(15);''2 - Print history of payments
440 PRINT TAB(15);''3 - Hardcopy history of payments
450 PRINT TAB(15);''4 - Quit
460 PRINT
470 INPUT''      Number of your choice  '';XX
480 ON XX GOTO 540,800,990,520
490 GOTO 470
500 `
510 `quit routine
520 CLOSE:CLS:END
530 `
540 CLS
550 INPUT''Date of payment (YYYYMMDD)  '';D#
560 INPUT''Amount of payment          '';P
570 PRINT
580 IF B(1)=0 THEN T1=(BA*IR)/12 ELSE T1=(B(N1)*IR)/12
590 IF P<T1 THEN T3=B(N1)+(T1-P):T2=0:T1=P:GOTO 620
600 T2=P-T1
610 IF B(1)=0 THEN T3=BA-T2 ELSE T3=B(N1)-T2
620 PRINT F1$
630 PRINT F2$
640 PRINT USING F3$;D#;P;T1;T2;T3
650 N1=N1+1
660 D#(N1)=D#;P(N1)=P;IA(N1)=T1;PI(N1)=T2;B(N1)=T3
670 INPUT''Any more payments to record (y/n)'';XX$
680 IF XX$=''Y'' OR XX$=''y'' THEN 550
690 `
700 `save the updated file
710 OPEN''O'',1,FF$
720   PRINT #1,BA,IR
730   FOR I=1 TO N1
740     PRINT #1,D#(I);P(I);IA(I);PI(I);B(I)
750   NEXT I
760 CLOSE 1
770 GOTO 360
780 `
790 `print payment history on screen
800 CLS
810 INPUT''Starting date (YYYYMMDD)'';D1#
820 INPUT''to ending date (YYYYMMDD)'';D2#
830 PRINT F1$
840 PRINT F2$
850 PRINT
860 S1=0:S2=0:S3=0
870 FOR I=1 TO N1
880   IF D#(I)<D1# OR D#(I)>D2# THEN 910

```

```

890 PRINT USING F3$;D#(I);P(I);IA(I);PI(I);B(I)
900 S1=S1+P(I):S2=S2+IA(I):S3=S3+PI(I)
910 NEXT I
920 PRINT
930 PRINT USING F4$;S1;S2;S3
940 PRINT
950 INPUT "press enter to continue";XX
960 GOTO 360
970 '
980 'make hardcopy of payment history and page it
990 CLS
1000 PA=1:I=1:S1=0:S2=0:S3=0:CT=1
1010 INPUT "Starting date (YYYYMMDD)";D1#
1020 INPUT "to ending date (YYYYMMDD)";D2#
1030 LPRINT FF$;TAB(60);"Page ";PA
1040 LPRINT " "
1050 LPRINT F1$
1060 LPRINT F2$
1070 LPRINT " "
1080 IF D#(I)<D1# OR D#(I)>D2# THEN 1110
1090 LPRINT USING F3$;D#(I);P(I);IA(I);PI(I);B(I)
1100 S1=S1+P(I):S2=S2+IA(I):S3=S3+PI(I):CT=CT+1
1110 I=I+1:IF I>N1 THEN LPRINT " ":LPRINT USING F4$;S1;S2;S3:LPRINT
CHR$(12):GOTO 360
1120 IF CT<=36 THEN 1080 ELSE PA=PA+1:CT=1:LPRINT CHR$(12):GOTO
1030
1130 '
1140 'error trap for file not found
1150 IF ERR <>53 THEN ON ERROR GOTO 0
1160 'IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0
1170 CLS
1180 PRINT "That file does not exist yet."
1190 INPUT "Do you wish to create it (y/n)";XX$
1200 IF XX$="N" OR XX$="n" THEN 520 ' to close, clear and END
1210 OPEN "O",1,FF$
1220 INPUT "What is the beginning balance ";BA
1230 PRINT "What is the interest rate."
1240 INPUT "enter as a decimal, like .085";IR
1250 PRINT #1, BA,IR
1260 CLOSE 1
1270 GOTO 270
1280 END 'of program

```

Change lines for Tandy I and III

Changed->160 CLEAR 2000 ' use only with BASIC prior to ver. 5.0
 Changed->1150 'IF ERR <>53 THEN ON ERROR GOTO 0
 Changed->1160 IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0

Editor's Notes, from page 2

Souvenir was too clutzy. It turns out that Schoolbook is just a little heavier than Times Roman, but not so overpowering as Bookman. I think we'll keep it for a while. Aside from that, Schoolbook's italics stand out a lot better than in any of the other typefaces we have available. And just in case you are interested in the difference, our listings are all done in 11 point Courier. We use that because Courier is a mono-spaced face, which means that it prints 10 characters to the inch like a typewriter. If we didn't do that, we would lose the spacing in the listing between quotes, for example. As you can see, there's more going on in publishing a magazine than selecting the editorial content.

Ever since we started with the NFL Oracle, back in 1986, we have been keeping track of how well it does. So far, this year, it is way ahead of what it did in previous years. Even though there are still two regular season games to be played as I write this, NFL88 is picking at 66 percent! Now I suppose after that boasting, it will die on us for the last two weeks of the season. What it really says, though, is that the teams are playing at just about their calculated strengths, in spite of an occasional upset. That's good, because it will up our averages for the year and also give us a better chance at picking the post season games and the Super Bowl with Playoff.Bas. This year, we will be putting the week 16 statistics on the download so you can use Playoff.Bas. In prior years, of course, we only needed the stats for the first 15 weeks. It's interesting to note that with only two weeks of regular season play left, there are several divisional championships still up for

Notes

As you know, MS DOS and GW BASIC have on-screen editing capability. This makes it easy to duplicate lines of code because you can edit line numbers. Something not so obvious was that you can re-execute a direct statement (statement without a line number), simply by moving the cursor under the first letter of the command and pressing ENTER. We had this

grabs (ours included.) Of course, by the time you read this, we will all know a little more about it all. But right now, I'd say that the Super Bowl will be played between the Vikings and the Bills. Oh well, I've been wrong many times before.

Did you know that anything you can print on the screen or printer can also be printed to a disk file? Including tabs, print USING and all the rest? We found that out when we tried to capture an output screen so that we could import it into PageMaker and put it directly into the issue. We couldn't think of a better way to do it, so we just went into the program and wherever it prints out, we added an identical line that instead of just saying PRINT, said PRINT #1,. Naturally, you open a disk file first so that you can input all that information. After the program has printed all its stuff on the screen and into the disk file, you close the file. Then, you can load the file with a text editor and clean it up if necessary, and then import it directly into the desktop publisher. If you choose 10 point Courier for a type face, it will even space out correctly in the typesetting program. Well, it isn't 100 percent yet, but you can see some of the results we got on page 31 of this issue. Our efforts are to make this a totally desktop published issue. We even scan in the cartoon (you can tell by the little jagged edges on curved lines) and hope the author of those cartoons doesn't mind too much.

Another new year is upon us, and we wish to take this opportunity to thank each and every one of you for supporting CodeWorks and our efforts to bring good BASIC programming to you. We hope your new year will be peaceful and prosperous. Happy New Year to you all. Irv

demonstrated quite by accident lately when we entered the following line to check out a random number sequence:

```
FOR I=1 to 100:PRINT INT(RND(1)*15)+1;
NEXT I
```

We were checking to see that the numbers were in the range we asked for. By putting the cursor under the F in FOR, and pressing ENTER, the whole sequence executes again. You can do the same with RUN or LIST.

Handy Order Form

| | | |
|--|---------|--|
| RENEW Subscription Nov/Dec 88 through Sep/Oct 89 | \$24.95 | |
| All third year issues, Nov 87 through Sep 88 | \$24.95 | |
| All second year issues, Nov 86 through Sep 87 | \$24.95 | |
| All first year issues, Sep 85 through Sep 86 | \$24.95 | |
| 1st Year Program Disk (issues 1 through 7) (Specify computer type below) | \$20.00 | |
| 2nd Year Program Disk (issues 8 through 13) (Specify computer type below) | \$20.00 | |
| 3rd Year Program Disk (issues 14 through 19) (Specify computer type below) <i>Available now</i> | \$20.00 | |
| NEW! "Starting with MS DOS" 40-page book explains all | \$7.00 | |

Postage and handling charges already included in all prices.

We can supply program diskettes for PC/MS DOS, TRSDOS 1.3 (Tandy Model 3), TRSDOS 6.x (Tandy Model IV) and most CP/M MBASIC formats, on 5 1/4 inch diskettes. When ordering diskettes, specify your computer type.

COMPUTER TYPE _____

Check/MO enclosed

Charge to my VISA/MC _____ exp _____

Ship to: Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to:
CodeWorks, 3838 S. Warner St. Tacoma WA 98409

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

VISA/Master Card only

189

Index & Download

What's happening with both

Here are the updates to bring Cwindex.Dat up to date through the last issue. The entire index for the first three years of CodeWorks is on the download and on our yearly diskette.

Correl.bas, reference, issue 20, page 3

Notes, MS DOS FIND command, issue 20, page 4

Notes, on hard disks, issue 20, page 5

Qkey.bas, reference, how to edit errors, Issue 20, page 5

Errmsg.bas, main program, issue 20, page 8, expanded error messages

Beginning BASIC, error messages for beginners, issue 20, page 7

Playoff.bas, main program, issue 20, page 14, post season predictions

Cword.bas, main program, issue 20, page 21, a chain word program

Random files, yearly recap of changes, issue 20, page 28

Ranprnt2.bas, main program, issue 20, page 29, column totals

Split.bas, main program, issue 20, page 37, split ASCII files

Download, notes on the download, issue 20, page 40

Notes, where to check for EOF, issue 20, page 20

The download has been running rather smoothly for the past two months. Our power shifting has finally come to an end and things have settled down nicely.

Keep in mind (if you are a football fan) that this year we will be utilizing the statistics for week 16 of regular play. This will be needed for the program Playoff.Bas (from the last issue) to predict playoff games and the Super Bowl. We will have the week 16 stats on the download by Tuesday noon of the week following the last week of regular season play. The file size by then will be at least 10 to 12K.

CodeWorks

3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
U.S. Postage
PAID
Permit 774
Tacoma, WA

2910
ERICKSON, MIKE / KRJB
BOX 250
MONTE RIO CA 95462

CODEWORKS

Issue 22

Mar/Apr 1989

CONTENTS

| | |
|-------------------------------------|-----------|
| Editor's Notes | 2 |
| Forum | 3 |
| Beginning BASIC | 6 |
| Budget.Bas | 9 |
| Notes | 18 |
| Sort on Input | 19 |
| Flow.Bas | 24 |
| Pay2.Bas | 30 |
| CWindex & Download | 40 |

Issue 22

Mar/Apr 1989

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

(c)1989 80-Northwest Publishing Inc. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, the publisher assumes no responsibility for errors or omissions. Unless otherwise noted, all programs presented in this publication are placed in public domain. Please address all correspondence to CodeWorks, 3838 South Warner Street, 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. 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, WA.

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.

As many of you found out, we weren't answering our phone for a couple of days in early February. This area, in spite of being more than 47 degrees North, hardly ever gets to see snow. Hardly ever happened, in fact, on the 1st of February. Not only did we get snow, but it stayed cold for over two weeks, keeping the stuff right there on the roads. In an area like this, where it snows so seldom, people just don't know how to drive in it and so there were lots of interesting accidents and fender benders. Anyway, we all did the prudent thing and stayed home during the worst couple of days. Now that it has thawed out, the auto body people and the plumbers are having a field day.

During the cold snap I took the opportunity to start researching and writing a feature article for the magazine on Artificial Intelligence. The search of available material led me down some surprising paths. In trying to answer the questions that came up I was led into the realm of quantum physics, of all things. I found two books on that subject that were totally absorbing - I couldn't put them down, even though they had strayed from the primary focus of what I was trying to write about. Still, there were several questions that are frequently asked by both the proponents of AI and quantum physics. One of them asks "what's real?" and the other is "just what is intelligence?" Both are interesting and intriguing questions when you stop to think about them. From what I could gather, physics is now down to the level where they don't know if it's really physics or philosophy. The physicist Heisenberg (uncertainty principle) says that the mere act of measuring anything at the quantum level forever changes it into something else and excludes all possibility of performing a different measurement on that same thing.

There are several schools of thought on these subjects. Niels Bohr

and Albert Einstein had several interesting exchanges in regards to quantum mechanics and reality. One theory has it that if you can't see something it really doesn't exist, or that reality is in the eyes of the beholder. Yet another says that when one course of action is chosen, all the other possible courses of action take place too, in another reality. In all there were about six or seven different theories like this. Amazing stuff - and these guys got paid for sitting around and thinking up these things. It's interesting to note that Bohr and Co. occupied facilities in Copenhagen donated by a beer company. Then there is the Austrian physicist Erwin Schrodinger's cat - but by now I was so far off the course of AI that I reluctantly put away the quantum books and got back to the subject.

If what I have just said piques anyone's imagination, the two books on quantum physics (very readable, by the way) were: *Quantum Reality: Beyond the New Physics*, Nick Herbert, Anchor Press/Doubleday, Garden City NY 1985 and *Beyond the Quantum*, Michael Talbot, Bantam Books, 1988.

Meanwhile, back in the real world, I began to wonder if Dettmann had deserted us. I got a phone call from him just after midnight one night. He was calling from the airport and was on his way again on yet another of his seemingly endless trips. He told me that he had the Outline program in his portable and would send it to me via modem from wherever he happened to be. A few days later I found a few of the opening lines of Outline on my computer and nothing else. It was just about one screenful. There's still a little time before we close this issue, and he just might show up yet - but I'm not holding my breath. Have faith, good people, we will get that program published yet!

Irv

Forum

An Open Forum for Questions and Comments

I am trying to get information on how Dollar (packed) or "packed decimal" works, the basic code used and how to read it in a hexadecimal file. I have a program that uses it and would like to be able to change error amounts by using a file editor. I have not found any books or articles on it under these names. Do you have any recommendations?

E. A. Hamer
Lighthouse Point, FL

Although I'm not quite sure what you are asking, it sounds a little like "string packing." String packing is a technique whereby you assign a string variable with, say, ten spaces between quotes. Then you put the data which will be packed into that string into data statements. Then, with a loop, you read the data statements and poke the data into the empty string. Then you can delete the data statements themselves and the loop which poked the data into the string. You can put machine code into such a string, and when it executes in Basic, it will execute the machine code. It also takes up less memory space than conventional methods. On some earlier machines, it was the only way to get fast graphics on the screen. This scheme was never documented in any manufacturer's literature, but was the subject of various articles in some computer magazines (80-Micro comes to mind as one of them.)

I have a correction you can make in Issue 20, page 35, under changes to Ranidx.Bas for Models II/IV.

```
Line 4620 SYSTEM "RENAME "+FX$+"
"+TO"+" "+FT$
```

change to

```
Line 4620 NAME FX$ AS FT$
```

Loyd G. Orr
Bellevue, NE

Now that you mention it, you certainly are right.

... is there any reason that Bio.Bas won't

"VXREF?" I have not been able to get a printout of it.

S. A. Langell
North Canton, OH

Can't think of any reason Bio will not VXREF, except, if it is not saved in ASCII format first. Try this: SAVE "BIO.BAS",A

... I have been reading and entering some programs from the book "Modems and Communication on IBM PCs" by W. David Schwaderer. There is some fun stuff here. Much of the simple material is covered in your Beginning BASIC series. Some of the more complicated material on communications software could be useful. The author writes strictly for BASICA. If this were converted to generic BASIC, those of us who use other machines could benefit.

I like your magazine. There isn't much left for the casual programmer, except you. May you live a thousand years.

Tom Witt
Rochester, NY

This is something we'll have to put on our to-do list. Thanks, and may you too, live long and prosper.

... I too, have been with you since issue number one. Your editorial in number 19 was very interesting. I do believe that we will see as many improvements in the next twenty years.

... Phil Brown wrote me a letter and thanked me for the plug he got in Issue 17 when I wrote you about the Family History System genealogical program. My Long Family History file now has 967 records and the disk is only 3/4 full. I may have to get a hard card if I find many more cousins.

Seymour E. Long
Margate, FL

As a charter subscriber to your magazine you are to be congratulated on publishing the very

best computer programming publication I have been able to find in over 10 years of work and play with computers. Thanks, and keep it up.

For some time I have been searching for a program which would compare paying for a loan with any selected interest rate versus paying with cash from a bank account, and repaying yourself over the period of the loan with deposits equal to the monthly loan payments, compounding the interest as the bank account is replenished to the amount of the withdrawal.

To clarify, you take a \$10,000 loan, with a 12.5% APR, which would carry an interest amount of \$1295, and a monthly payment of roughly \$450. Against this you wish a comparison should you have a \$10,000 bank account which you can withdraw and pay cash, but each month you deposit \$450 back into the bank account, and assuming certain selectable bank interest rates, with compounding at variable selectable periods, you get a gain or loss over and above the interest you would receive if the money remained in the savings account? . . .

Otto Kinbacher
Babylon, NY

Back when they started the world, there was a little footnote in the charter that said, "You will always pay more interest than you receive," and apparently, no one has ever changed it. But getting back to reality, we have several different types of loan programs floating around here. We'll work on putting something together that will make the comparison.

. . . I have a lot of fun with your programs and the feature articles are very helpful. I am taking a local Basic programming course and doing some home study, so your magazine helps everything fit together.

My problem with line 1000 in the NFL88 program coming up "Subscript out of range" finally cleared up. Although I could find no discernible error, when I retyped the line completely the problem disappeared. I have had this happen occasionally before. . .

There are two errors, however, in the NFL88 program that I found and had to clear up to get the program to run properly. Lines 1240 and 1250 each have terms TO which should read SO.

Donald E. Williams

Tucson, AZ

Retyping entire lines has happened to us too on occasion. Sometimes, it's hard to explain it. As to the errors in lines 1240 and 1250: Those variables are T-zero and S-zero, not TO and SO. In fact, there is already an S-zero in the previous lines. If you changed TO to SO, you probably have gotten yourself into even more trouble. Check line numbers to see what a zero looks like and the word "FOR" to see what an "OH" looks like. We used to slash our zero, but with our new way of putting the magazine together it doesn't work anymore.

. . . Thank you very much for the many hours of information and pleasure that have been gained from your magazine. After reading The Forum, I want to add my dittos to Mr. Jeavons' comments about hanging in there with this great work of education and enjoyment - don't give up the ship. Concerning Mr. Kelley (\$194), I also am very busy (travel for a living), but you can always make time for some pleasures in this life considering the time spent is both rewarding, educational and affords you many time-saving concepts and ideas that are most rewarding.

Bill Dahlstrom
Cliffs Notes, Inc
Lincoln, NE

Thanks for the nice comments. We'll try.

I am virtually a charter subscriber to CodeWorks and after all this time, I still don't know why. I have 1-2-3, DBase, and WordPerfect on a clone and lack for little in the way of software that I need. Your magazine is well-written, by computer standards, and informative. What I would like to see is material directed more towards applications that are not likely to be provided by the major software houses. There are already terrific database and modeling tools available. Why not explore new areas, graphics, for example.

Attached is an article from Scientific American describing a set of curves that can be developed, iteratively, with truly impressive results, at least as reported by the author.

(Following) is a short program which I had hoped to be the basis of an integrated set of programs described in the article. I started with

a simple exercise, draw a circle, the hard way, point by point. Only with the most egregious of finagling constants could I get a round circle.

```

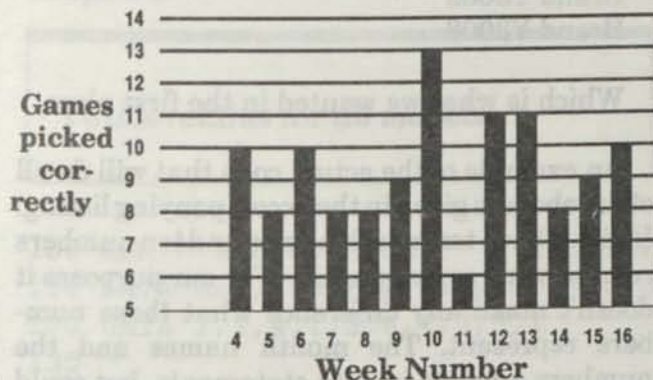
10 SCREEN 2
20 WINDOW (0,0)-(330,100)
30 INPUT "ENTER SCALE <50";C
40 CLS
50 FOR T=0 TO 360
60 R=T*3.1415/180
70 X=(C*600/240*COS(R)+165)
80 Y=(C*SIN(R)+50)
90 A=PMAP(X,2)
100 B=PMAP(Y,3)
110 PSET (X,Y)
120 PSET (A,B)
130 NEXT T
140 END

```

Why do I have to finagle the X coordinate to get a round circle? Shouldn't the WINDOW correct for the differences in the physical and logical coordinates? . . .

**David Charlton
Corning, NY**

(We tried (the above) program to plot circles and found it to be rather impressive. Getting round circles is always a problem. This is because even though the digital circuits in your computer may define a perfect circle, the video amplifiers in your video display are analog. Both vertical and horizontal deflection amplifiers have gain and linearity problems. You don't see it, usually, with just text on the screen. But with graphics these



This is how NFL88 picked them for the past season. We were over 50% for every week but one, for an over-all performance of 66.48%
Playoff.Bas did worse, picking only 55%.

come into apparent display. Even when you fiddle with the gain to make a circle round, it still may not be, because the amplification is not constant over the entire sweep of the amplifier. Higher quality video display units exhibit less of these tendencies, but still have them.

Well, if you take a look at the lower left of this page, you will see the results of NFL88 for this year. Not too bad - actually, the best we have ever done with it in the three years it's been around. Not so with our Playoff.Bas program. Out of the nine games (including the Super-Bowl) we only picked five correctly. Not so very good, and we can't even blame it all on the Vikings this year! And what happened to my prediction in the last issue? You win some and you lose some.

Thanks once again for the good input, and we'll see you next in the Spring. **Irv**



An old man, a computer and the sea.

Beginning BASIC

Two-Level Sorting

Sorting on two levels is not really that big a deal when you get into it. Let's take an example where we have two arrays, with the information in the first array tied somehow to the information in the second. (Like name and age, or item and price.) The steps necessary to do a two level sort on these arrays are:

1. Sort the first array in the normal manner, making sure that when a switch is required that you also switch the corresponding item in the second array.
2. Sort again, this time on the second array - but make switches only when the corresponding item in the first array is the same as the item following it.

Let's say we have two makes of television, Brand X and Brand Y, and that we have three different models of each. We want to sort by make, and within make, we want the model numbers to be in ascending order. The unsorted list might look like this:

```
Brand X4008
Brand Y3003
Brand Y3002
Brand X4000
Brand X4002
Brand Y3001
```

We first sort on the Brand, carrying the model number along when we must make a switch. In pseudo-code, it would look like this:

```
FOR I=1 TO 5
  nextbrand = I+1
  if brand is equal or less than nextbrand then
goto NEXT I
  else switch brand with nextbrand, and...
  also switch the corresponding model numbers
NEXT I
```

After doing the above, our two arrays would look like this:

```
Brand X4008
Brand X4000
Brand X4002
Brand Y3003
Brand Y3002
Brand Y3001
```

Now we can do the second level sort. Again, in pseudo-code, it would look like this:

```
FOR I=1 TO 5
  nextmodel = I+1
  if brand is NOT equal to nextbrand then goto
NEXT I
  if model# is equal or less than nextmodel#
then goto NEXT I
  else switch model# with nextmodel#
NEXT I
```

Now we are done and the list looks like this:

```
Brand X4000
Brand X4002
Brand X4008
Brand Y3001
Brand Y3002
Brand Y3003
```

Which is what we wanted in the first place.

An example of the actual code that will do all of the above is given in the accompanying listing. In it, we have ten month names and ten numbers representing some quantity. For our purposes it doesn't make any difference what those numbers represent. The month names and the numbers are held in data statements, but could just as well have been brought in from a disk file. Since we have them in data statements, we need to read them into their respective arrays. Note

that one array is a string array and the other is integer. This doesn't have to be like that, but is there to show that you can sort either or both.

Our arrays are set up from the data statements in lines 140-170 and 190-220. For purposes of illustration only, we next go to the subroutine at line 530 to print the arrays out so you can see them.

Our first level sort takes place between lines 280 and 340. Notice that our loop count goes to one less than the number of items to sort. This is because in line 290 we are looking ahead one, and so that last item will be taken into account. In line 300 if the item we are looking at is equal or less than the next item in the list, we leave it alone and go on to the next item in the list. If the item we are looking at is larger (larger ASCII value) than the next item in the list, we switch the two in line 310. Then we switch the corresponding items in the second array in line 320. Also in line 320 we set the flag F, equal to one. This flag is used to tell us that we have just made a swap. In line 340 we will check to see if that flag is set, and if it is we reset it to zero and go through the list again, looking for more things to swap. When we can get through the entire list without making a swap (F does not get set to one) it means that the list is now sorted and we drop through line 340 to the next section of code.

In line 370 we again print out the list to see what it looks like after the first level sort is completed.

TwoLevel.Bas for all models

```
100 REM * Twolevel.bas * a two level sort demo
110 DATA Mar,Jan,Jan,Jan,Jan,Jan,Jan,Jan,Jan,Jan
120 DATA 2,2,4,1,3,6,4,3,5,1
130 `
140 `Read in the 1st 10 data items
150 FOR I=1 TO 10
160   READ A$(I)
```

The second level sort is similar, but not exactly the same as the first level sort was. The flag works the same way. So does the swap, except this time we switch items in the second array. The trick here is in line 420. It works this way: We don't even look at the second array if the item and next item in the first array are not equal. And we never move items in the first array, since they are already sorted into the proper order. It is only when the item and next item in the first array are equal that we look at the second array locations. And then we only make a switch if the array two item is larger (in ASCII value) than the next array two item. This is how we can get a "sort within a sort."

In line 490 we again print out the two arrays to see how they look. In the subroutine to print out the arrays, at line 560 we increment a C count, and at line 570 we print a prompt on the screen to press ENTER. The last time we print the arrays C will equal 3 and we bypass the prompt and just quit because we are done.

We used bubble sorts in our example. In actual practice you might want to use a faster sort, especially for the first sort. In fact, you could use any sort routine in either place. Since the second level usually does not take as many items into account a bubble sort there will usually suffice.

The idea of two level sorting could just as well be extended to three level or more sorting. It just takes more time and more code, but the idea is the same.

The program Flow.Bas, in this issue, shows a good example of how two-level sorting works.

```

170 NEXT
180 `
190 ` Read in the 2nd set of 10 data items
200 FOR I=1 TO 10
210   READ B(I)
220 NEXT I
230 `
240 `print out the unsorted arrays
250 GOSUB 530
260 `
270 `Do the first level sort - swap both arrays
280 FOR I=1 TO 9
290   L=I+1
300   IF A$(I) <= A$(L) THEN 330
310   T$(I) = A$(I) : A$(I) = A$(L) : A$(L) = T$(I)
320   T(I) = B(I) : B(I) = B(L) : B(L) = T(I) : F=1
330 NEXT I
340 IF F=1 THEN F=0 : GOTO 280
350 `
360 `Print out the 1st level sorted list
370 GOSUB 530
380 `
390 ` Do the second level sort
400 FOR I=1 TO 9
410   L=I+1
420   IF A$(I) <> A$(L) THEN 450
430   IF B(I) < B(L) THEN 450
440   T(I) = B(I) : B(I) = B(L) : B(L) = T(I) : F=1
450 NEXT I
460 IF F=1 THEN F=0 : GOTO 400
470 `
480 `print out 2nd level sorted list
490 GOSUB 530
500 END
510 `
520 `subroutine to print out the list
530 FOR I=1 TO 10
540   PRINT A$(I), B(I)
550 NEXT I
560 C=C+1 : PRINT
570 IF C < 3 THEN INPUT "Press Enter to continue"; XX
580 RETURN

```


Budget.Bas

A Personal Budget Program

R. J. Richardson, Valencia, California.

If money is the root of all evil, then surely the lack of it is all the rest of the forest. While a paucity of pennies is cold and horrid, a glut of gelt conveys the warmth and goodness of Mom, apple pie, and the American Way. Love may conquer all, but the guy who said that never tried it when he was broke.

Budget.Bas provides a means to be as thorough as you wish in recording how much money comes in, and why, where and when it goes out. The program is designed to be run monthly, but the interval is up to you. It is dimensioned for 20 categories, although only 19 are used, since #20 does a little sweeping and dusting (internal housework) for the program itself. Each of the 19 categories is dimensioned for 20 entries, and has its own "page," selectable from the main menu. This construct allows the program to operate exactly like a ledger, providing instant access to any page. Category 20 contains summary, print, and disk access functions.

To personalize the program, you should install your own Data statements (category names) in lines 40 through 74. However, the second statement in line 74 (Summaries and Disk Access) must remain as it is. This group of data statements must total 20, lest it cause an "out of data" error, something which produces brain fatigue by being mis-diagnosed by the error trapping subroutine. By the way, deleting line 2900 will prevent this mis-diagnosis, but at the risk of having a loaded program dumped before the data is saved.

Line 140 contains the amount budgeted for each category, and its data entries should total

18. The "Reconcile:" label which appears in each page of the budget indicates by how much you are over, or under, budget. If you went over, the figure will be negative.

In the interest of clarity, a data read loop is placed directly above its own data.

Operation of the program requires a few extras. A small spiral notebook is used to record daily spending *as it occurs*. (It will not work if you wait until evening, and then try to recall everything you spent during the day.) A calendar of the month is photo-copied (19 copies), three-hole punched, and copies placed in a three-ring binder. Each copy is numbered (one page for each category) with a marking pen, and you may "screen print" the program's main menu to obtain your index.

Expenditures are noted as they are made in the small notebook, and copied into the three-ring binder (proper date and category) at your convenience. Don't forget to include expenses paid by check. The program is run from the three-ring binder at the end of each month, and the print-out can provide data for income tax preparation or other financial review.

You should expect to revise your category names after using the program for a time, since some categories will prove redundant, and other (new) ones will be needed. However, remember to change the three-ring binder (calendar) pages, and the index at the same time.

It is important that you remember the follow-

ing: There are four entries to be made for each expense on a category "page," the date (08/88), the description of the expenditure, the amount (rounded up to integers, and cash or check number.) Please remember that you must PRESS ENTER after each entry. You cannot go across the page with either the right arrow, or the space bar. OK?

Variables are identified in lines 600 to 740, and the entire program (excepting the lines below 100) was renumbered using CodeWorks' own Renum.Bas (Issue 9, Jan/Feb 1987), a mega-algorithm for the serious programmer.

And so, what's next?

Budget.Bas for GW BASIC, with minor modifications will run on TRS-80 IV

```

5 `Budget.Bas * R J Richardson *
10 `----- Read Data - Dimension Arrays -----
20 KEY OFF
30 ON ERROR GOTO 2870
40 DIM CT$(20):FOR N=1 TO 20:READ CT$(N):NEXT N
50 DATA ``Income Tax:`,`House & Assn. Payment:``
52 DATA ``House: Ins-Tax-Maint:`,`Gas-Electric-Telephone:``
54 DATA ``Food and Sustenance:`,`Trash-H2O-Paper-Cable:``
60 DATA ``Auto: Gas-Repair-Misc:`,`Auto: Insurance-License:``
62 DATA ``Aircraft Fuel:`,`Aircraft Expense: Gen'1:``
64 DATA ``Toni: Necessaries:`,`Bobby: Necessaries:``
66 DATA ``Medical: Pres-Ins-Mis:`,`Toni: Beauty-Grooming:``
70 DATA ``Charge Cards:`,`TONI-Personal Acct:``
72 DATA ``BOBBY-Personal Acct:`,`Misc.-(Other Things):``
74 DATA ``Monthly Income:`,`-- Summaries & Disk Access ---``
80 FOR I=1 TO 6:READ W$(I):NEXT I
82 DATA ``(P)rint existing file to screen:``
84 DATA ``(L)ine print file to printer:``
86 DATA ``(A)dd to existing file:``
88 DATA ``(E)dit existing file:``
90 DATA ``(Q)uit category:``
92 DATA ``(Enter . for Date to Terminate Entry:``
100 HD$=`File #:      Date:      Description:
      Amount $:      Check #:``

```

Well, can you

1. Write a subroutine which figures what percentage of the total expenditure is spent in each category?

2. Compare your percentages with those recommended by business advisors?

3. Re-write the program into an appointment scheduler which prints out an appointment listing either daily, or by the month. (The mechanisms - the big guys always say algorithms - are all here. You only have to change some of the fluff.)

Thanks for your time.

```

110 ZW$='#####.##'
120 DIM DT$(20,20),DS$(20,20),AM(20,20),CK$(20,20),ST(20),PT$(20),
    TT(20),BD(20)
130 FOR I=1 TO 18:READ BD(I):NEXT I
140 DATA 175,440,145,185,400,75,175,150,300,100,60,100,60,75,250,
    200,400,100
150 GOTO 240
160 '
170 '---- Universal Print@ / Locate Subroutine: UNremark as
    needed.
180 LOCATE X,Y:RETURN ' MS-DOS GW-BASIC
190 ' PRINT@((X-1*64)+(Y-1),,;:RETURN ' TANDY MODELS I / III
200 ' PRINT@(X-1),(Y-1),,;:RETURN ' TANDY MODEL IV
210 ' PRINT@(X,Y),,;:RETURN ' SOME MBASIC MACHINES
220 ' PRINT CHR$(27)+'Y'+CHR$(31+X)+CHR$(31+Y);:RETURN ' CP/M
    ADJUST TO SUIT
230 '
240 CLS:X=5:Y=10:GOSUB 170;
250 INPUT''Please Enter Month and Year (jan88):'';MN$
260 '----- Print main menu -----
270 CLS
280 X=2:Y=18:GOSUB 170:PRINT''Your Ever-lovely Hot-dog Financial
    Report!'' :PRINT
290 FOR N=1 TO 20:IF ST(N)=0 THEN PT$(N)=''' ELSE PT$(N)=''*''
300 NEXT N
310 FOR N=1 TO 19:PRINT N;'' . '' :PRINT PT$(N);'' '' :PRINT CT$(N):
    NEXT N
320 N=20:PRINT N;'' . '' :PRINT PT$(N);'' '' :PRINT CT$(N):X=4:Y=63:
    GOSUB 170:PRINT''* = Data''
330 X=25:Y=62:GOSUB 170:PRINT''(999 To Quit)''
340 X=25:Y=30:GOSUB 170:INPUT''Select Category:'';CT:IF CT=20 THEN
    2050 ELSE IF CT=0 THEN 340
350 IF CT=999 THEN 2910 ELSE IF CT>20 THEN 340
360 GOSUB 370:GOTO 430
370 '----- Print Individual Page Heading -----
380 CLS
390 X=1:Y=1:GOSUB 170:PRINT STRING$(80,205)
400 X=2:Y=1:GOSUB 170:PRINT''Page'';CT;'' : '' ;MN$
410 X=2:Y=27:GOSUB 170:PRINT''Category'';CT;'' : '' ;CT$(CT);'' (''' ;BD(CT);
    '' )''
420 PRINT STRING$(80,205):RETURN
430 '-----Draw Box for Internal Menu -----
440 X=5:Y=15:GOSUB 170:PRINT STRING$(50,223)
450 X=14:Y=15:GOSUB 170:PRINT STRING$(50,220)
460 FOR P=5 TO 14:X=P:Y=15:GOSUB 170:PRINT CHR$(222):NEXT P
470 FOR P=5 TO 14:X=P:Y=65:GOSUB 170:PRINT CHR$(221):NEXT P

```

```

480 \ -----Print Menu Inside Box-----
490 W=1
500 FOR X=7 TO 11
510 Y=20:GOSUB 170:PRINT W$(W)
520 W=W+1:NEXT X
530 X=13:Y=45:GOSUB 170:PRINT'' _____?''
540 G$=INKEY$:IF G$=''' THEN 530
550 IF G$='''P'' OR G$='''p'' THEN ZQ=1:GOTO 750
560 IF G$='''L'' OR G$='''l'' THEN ZQ=2:GOTO 880
570 IF G$='''A'' OR G$='''a'' THEN ZQ=3:GOTO 1620
580 IF G$='''E'' OR G$='''e'' THEN ZQ=4:GOTO 1730
590 IF G$='''Q'' OR G$='''q'' THEN 260
600 \-----Identification of variables:-----
610 \ CT$ - Name of Category
620 \ CT - Category Number
630 \ MN$ - Report Month
640 \ TT - Total spent in a category
650 \ BD - Amt budgeted for a category
660 \ AM - Amt of money
670 \ DS$ - Description of expenditure
680 \ CK$ - Check number - or cash
690 \ TB - Total budgets for all categories
700 \ GT - Total of all money spent - all categories
710 \ TR - Total reconciliation: amt over or under budget
720 \ ST - 'Stop' number for each file: i. e. EOF
730 \ Note: 'Reconcile' does not have an assigned variable.
740 \ It is figured (BD-TT) as needed. (Don't ask.)
750 \ ----- List File to Screen -----
760 GOSUB 3240
770 IF ST(CT)=0 THEN PRINT:PRINT;CT$(CT);'' EMPTY:'' :GOSUB
3020:GOSUB 370:GOTO 430
780 TT(CT)=0
790 FOR I=1 TO ST(CT)
800 GOSUB 3290
810 TT(CT)=TT(CT)+AM(CT,I)
820 NEXT I
830 PRINT:PRINT TAB(25);''Budget: $ '';BD(CT);:PRINT TAB(58);
840 PRINT USING ZW$;TT(CT)
850 PRINT:PRINT TAB(40);''Reconcile: $ '';:PRINT TAB(58);
860 PRINT USING ZW$;BD(CT)-TT(CT)
870 INPUT''<Enter> for Menu:'';ZQ$:CLS:GOSUB 370:GOTO 430
880 \ ----- Line Print -----
890 GOSUB 370:X=4:Y=1:GOSUB 170:PRINT W$(ZQ)
900 X=5:Y=15:GOSUB 170:PRINT STRING$(50,223)
910 X=14:Y=15:GOSUB 170:PRINT STRING$(50,220)
920 FOR X=5 TO 14:Y=15:GOSUB 170:PRINT CHR$(222):NEXT X

```

```

930 FOR X=5 TO 14:Y=65:GOSUB 170:PRINT CHR$(221):NEXT X
940 X=7:Y=20:GOSUB 170:PRINT''Select letter:''
950 X=9:Y=25:GOSUB 170:PRINT''(L)print this category only.''
960 X=10:Y=25:GOSUB 170:PRINT''(P)rint entire expense report.''
970 X=11:Y=25:GOSUB 170:PRINT''(R)eturn to main menu.''
980 X=13:Y=48:GOSUB 170:PRINT''      ?'' :G$=INKEY$
990 IF G$=''' THEN 980 ELSE IF G$='''P'' OR G$='''p'' THEN 1210
1000 IF G$='''R'' OR G$='''r'' THEN 260
1010 `----- Lprint category only -----
1020 X=18:Y=20:GOSUB 170:PRINT''Printer - - OnLine:''
1030 X=19:Y=20:GOSUB 170:PRINT''Paper - Top of Form:''
1040 X=21:Y=20:GOSUB 170:INPUT''<Enter> - Starts Printer:'';ZQ$
1050 X=1:Y=1:GOSUB 170:LPRINT STRING$(80,205)
1060 X=2:Y=1:GOSUB 170:LPRINT''Page'';CT;'' : ``;MN$
1070 X=2:Y=30:GOSUB 170:LPRINT''Category'';CT;'' :'';CT$(CT)
1080 LPRINT STRING$(80,205):TT(CT)=0
1090 IF ST(CT)=0 THEN PRINT:PRINT CT$(CT);'' is empty.''
1100 IF ST(CT)=0 THEN LPRINT:LPRINT CT$(CT);'' is empty.'':GOTO 1170
1110 X=15:Y=15:GOSUB 170:PRINT'' ---- Printing ----'' :LPRINT HD$
1120 FOR I=1 TO ST(CT)
1130 LPRINT I;TAB(12);DT$(CT,I);TAB(26);DS$(CT,I);TAB(58);
1140 LPRINT USING ZW$;AM(CT,I);:LPRINT TAB(73);CK$(CT,I)
1150 TT(CT)=TT(CT)+AM(CT,I)
1160 NEXT I
1170 LPRINT:LPRINT TAB(58);:LPRINT USING ZW$;TT(CT)
1180 LPRINT:LPRINT TAB(20);''Budgeted:'';BD(CT);TAB(46);''Reconcile:'';

1190 LPRINT TAB(58);:LPRINT USING ZW$;BD(CT)-TT(CT)
1200 LPRINT:LPRINT''End of File:'' :GOSUB 370:GOTO 430
1210 `----- Lprint Entire Expense Report --
1220 X=18:Y=20:GOSUB 170:PRINT''Adjust paper to top of form:''
1230 X=20:Y=20:GOSUB 170:INPUT''<Enter> starts printer:'';ZQ$
1240 X=16:Y=17:GOSUB 170:PRINT''Printing Complete Report - - :''
1250 FOR K=1 TO 19
1260 TT(K)=0:LPRINT STRING$(80,254):LPRINT:LPRINT''Page:'';K;'' :``;
    MN$
1270 LPRINT''Category'';K;'' :'';CT$(K):LPRINT STRING$(80,250)
1280 IF ST(K)=0 THEN LPRINT CT$(K);'' -- Empty.'':LPRINT:GOTO 1380
1290 LPRINT HD$
1300 FOR I=1 TO ST(K)
1310 LPRINT I;TAB(12);DT$(K,I);TAB(26);DS$(K,I);TAB(58);
1320 LPRINT USING ZW$;AM(K,I);:LPRINT TAB(73);CK$(K,I)
1330 TT(K)=TT(K)+AM(K,I)
1340 NEXT I
1350 LPRINT:LPRINT TAB(58);:LPRINT USING ZW$;TT(K)

```

```

1360 LPRINT:LPRINT TAB(20);''Budget:'';BD(K);TAB(40);''Reconcile:'';
1370 LPRINT TAB(58);:LPRINT USING ZW$;BD(K)-TT(K)
1380 NEXT K
1390 `----- Line Print Budget Summary -----
1400 LPRINT:LPRINT''Category:'';TAB(30);''Budgeted:'';
1410 LPRINT TAB(46);''Amt. Spent:'';TAB(65);''Reconcile'' :LPRINT
1420 FOR K=1 TO 18
1430 LPRINT K;''.'';CT$(K);TAB(30);:LPRINT USING ZW$;BD(K);:LPRINT
TAB(46);
1440 LPRINT USING ZW$;TT(K);:LPRINT TAB(65);:LPRINT USING ZW$;
BD(K)-TT(K)
1450 NEXT K
1460 `----- Prepare & Print 'Bottom Line' -----
1470 TB=0:GT=0:TR=0
1480 FOR K=1 TO 18
1490 TB=TB+BD(K):GT=GT+TT(K):TR=TR+(BD(K)-TT(K))
1500 NEXT K
1510 LPRINT:LPRINT TAB(25);''The Bottom Line:'' :LPRINT
1520 LPRINT TAB(10);''Total Budget:'';TB
1530 LPRINT TAB(30);''Total Cash Spent:'';
1540 LPRINT TAB(55);:LPRINT USING ZW$;GT
1550 LPRINT TAB(30);''Less Cash Income: -'';
1560 LPRINT TAB(55);:LPRINT USING ZW$;TT(19)
1570 LPRINT:LPRINT TAB(30);''Reconciliation:'';TAB(55);
1580 LPRINT USING ZW$;TT(19)-GT:LPRINT
1590 IF SGN(TT(19)-GT)=-1 THEN LPRINT''Congratulations! You have a
Negative Cash Flow!''
1600 IF SGN(TT(19)-GT)=1 THEN LPRINT''Congratulations! You took in
More than you Spent!''
1610 GOSUB 370:GOTO 430
1620 `----- Add to Existing File -----
1630 GOSUB 370:X=4:Y=1:GOSUB 170:PRINT W$(ZQ);'' '' ;W$(6)
1640 X=6:Y=1:GOSUB 170:PRINT HD$
1650 X=7:Y=1:GOSUB 170:PRINT STRING$(80,196):R=8:M=ST(CT)
1660 M=M+1:X=R:Y=1:GOSUB 170:PRINT M:Y=11:GOSUB 170:LINE INPUT
DT$(CT,M)
1670 IF DT$(CT,M)=''.''' THEN M=M-1:ST(CT)=M:GOSUB 370:GOTO 430
1680 Y=26:GOSUB 170:LINE INPUT DS$(CT,M):Y=58:GOSUB 170;
1690 INPUT AM(CT,M):Y=72:GOSUB 170:LINE INPUT CK$(CT,M)
1700 R=R+1
1710 IF M=>20 THEN GOSUB 3160:GOTO 430
1720 GOTO 1660
1730 `----- Edit Existing File -----
1740 `
1750 GOSUB 370:X=4:Y=1:GOSUB 170:PRINT W$(ZQ)

```

```

1760 X=6:Y=1:GOSUB 170:PRINT HD$
1770 X=7:Y=1:GOSUB 170:PRINT STRING$(80,196)
1780 IF ST(CT)=0 THEN GOSUB 3200:GOTO 430
1790 ` print file as is
1800 FOR I=1 TO ST(CT)
1810 GOSUB 3290
1820 NEXT I
1830 PRINT:INPUT"Enter File # of Entry to be Edited:";I
1840 CLS:GOSUB 3240:GOSUB 3290
1850 X=11:Y=20:GOSUB 170:PRINT"Do you wish to?"
1860 X=13:Y=25:GOSUB 170:PRINT"(C)hange the line?"
1870 X=14:Y=25:GOSUB 170:PRINT"(R)emove the line?"
1880 X=16:Y=40:GOSUB 170:PRINT"_____?":PRINT
1890 G$=INKEY$:IF G$="" THEN 1890
1900 IF G$="C" OR G$="c" THEN 1980
1910 ` Remove the line
1920 IF ST(CT)=1 THEN ST(CT)=0:CLS:X=12:Y=35:GOSUB 170:
PRINT"Gone!":GOSUB 3020:GOTO 750
1930 XX=ST(CT):DT$(CT,I)=DT$(CT,XX):DS$(CT,I)=DS$(CT,XX)
1940 AM(CT,I)=AM(CT,XX):CK$(CT,I)=CK$(CT,XX):ST(CT)=ST(CT)-1
1950 CLS:X=12:Y=35:GOSUB 170:PRINT"Done!"
1960 FOR Z=1 TO 3000:NEXT Z:GOTO 750
1970 ` Change the line
1980 X=14:Y=1:GOSUB 170:PRINT STRING$(80," ` `"):PRINT STRING$(80,
` ` ` `)
1990 PRINT"Please Enter Correct Line Now: ` `":PRINT:
X=18
2000 PRINT I:Y=10:GOSUB 170:LINE INPUT DT$(CT,I)
2010 Y=24:GOSUB 170:LINE INPUT DS$(CT,I)
2020 Y=59:GOSUB 170:INPUT AM(CT,I)
2030 Y=72:GOSUB 170:LINE INPUT CK$(CT,I)
2040 GOSUB 3020:CLS:GOTO 430
2050 `----- Summary and Disk Access -----
2060 CLS:GOSUB 370
2070 X=5:Y=15:GOSUB 170:PRINT STRING$(50,223)
2080 X=14:GOSUB 170:PRINT STRING$(50,220)
2090 FOR P=5 TO 14:X=P:GOSUB 170:PRINT CHR$(222)
2100 NEXT P
2110 FOR P=5 TO 14:X=P:Y=65:GOSUB 170:PRINT CHR$(221)
2120 NEXT P
2130 X=6:Y=20:GOSUB 170:PRINT"Do you wish to?"
2140 X=8:Y=25:GOSUB 170:PRINT"(P)rint Results on This Screen?"
2150 X=9:GOSUB 170:PRINT"(L)ine Print Entire Report?"
2160 X=10:GOSUB 170:PRINT"(S)ave All Files to Disk?"
2170 X=11:GOSUB 170:PRINT"(R)etrieve Files from Disk?"
2180 X=12:GOSUB 170:PRINT"(G)o Back to Main Menu?"

```

```

2190 X=13:Y=52:GOSUB 170:PRINT''_____?'
2200 G$=INKEY$:IF G$=''' THEN 2200
2210 IF G$='S' OR G$='s' THEN 2520
2220 IF G$='L' OR G$='l' THEN 1210
2230 IF G$='R' OR G$='r' THEN 2680
2240 IF G$='G' OR G$='g' THEN 260
2250 ` Screen print Budget Summary
2260 CLS:PRINT''Category:'';TAB(30);''Budgeted:'';TAB(46);
    ``Amt. Spent:'';TAB(65);''Reconcile:''
2270 GOSUB 3090
2280 FOR K=1 TO 18
2290 PRINT K;''.'';CT$(K);:PRINT TAB(30);:PRINT USING ZW$;BD(K);
2300 PRINT TAB(46);:PRINT USING ZW$;TT(K);
2310 PRINT TAB(65);:PRINT USING ZW$;BD(K)-TT(K)
2320 NEXT K
2330 ` Last minute total of Category 19 (Income)
2340 TT(19)=0
2350 FOR J=1 TO ST(19)
2360 TT(19)=TT(19)+AM(19,J)
2370 NEXT J
2380 ` Calculate totals - bottom of page
2390 TB=0:GT=0:TR=0
2400 FOR K=1 TO 18
2410 TB=TB+BD(K):GT=GT+TT(K)
2420 TR=TR+(BD(K)-TT(K))
2430 NEXT K
2440 PRINT:PRINT TAB(30);:PRINT USING ZW$;TB;:PRINT TAB(46);
2450 PRINT USING ZW$;GT;:PRINT TAB(65);
2460 PRINT USING ZW$;TR:PRINT''Less Income: (Cat. 19): -----'';
2470 PRINT TAB(46);:PRINT USING ZW$;TT(19)
2480 PRINT TAB(31);''Reconcile: ``':PRINT TAB(46);
2490 PRINT USING ZW$;GT-TT(19)
2500 INPUT''<Enter>:'';ZQ$:GOTO 260
2510 INPUT''Enter:'';ZQ$:GOTO 260
2520 ` ----- Disk Access ----- Save Data -----
2530 CLS:M1$=MN$+'' .dat''
2540 X=5:Y=5:GOSUB 170:PRINT''CAUTION
2550 PRINT''Be sure that program data and file name are correct.''
2560 PRINT''An incorrect file name can erase an existing file.''
2570 PRINT''If you wish to double check, enter M or m''
2580 PRINT''at the prompt to return to the main menu.'':PRINT
2590 PRINT''Else, this data will be saved as: ``;M1$
2600 PRINT:INPUT''<Enter> to proceed or M for Menu:'';ZQ$
2610 IF ZQ$='M' OR ZQ$='m' THEN 260
2620 OPEN M1$ FOR OUTPUT AS 1
2630 PRINT #1,M1$:FOR K=1 TO 19:PRINT #1,ST(K)

```



```

2640 FOR J=1 TO ST(K)
2650 PRINT #1, DT$(K,J);',',',',DS$(K,J);',',',',AM(K,J);',',',',CK$(K,J)
2660 NEXT J,K
2670 CLOSE:GOTO 260
2680 `----- Data Rerieval -----
2690 CLS:X=5:Y=1:GOSUB 170:PRINT STRING$(80,177)
2700 X=7:Y=10:GOSUB 170:PRINT''Data Retrieval:''
2710 X=10:Y=15:GOSUB 170
2720 INPUT''Enter Month and Year (mmmyy) to Retrieve:'';MN$
2730 M1$=MN$+''.dat''
2740 OPEN M1$ FOR INPUT AS 1
2750 INPUT #1,M1$
2760 X=15:Y=25:GOSUB 170:PRINT''Retrieving: ``;MN$
2770 FOR K=1 TO 19:INPUT #1,ST(K)
2780 FOR J=1 TO ST(K)
2790 INPUT #1, DT$(K,J),DS$(K,J),AM(K,J),CK$(K,J)
2800 NEXT J,K
2810 CLOSE
2820 FOR K=1 TO 19:FOR J=1 TO ST(K)
2830 TT(K)=TT(K)+AM(K,J)
2840 NEXT J,K:TT(K)=-TT(K)
2850 MN$=LEFT$(M1$,5):GOTO 260
2860 END
2870 `----- Error Trap----- (don't fall in) --
2880 CLS
2890 CLS:IF ERR=53 THEN X=10:Y=15:GOSUB 170:PRINT''I can't find
file: ``;MN$:GOSUB 3020:GOTO 260
2900 CLS:IF ERR <> 53 THEN X=10:Y=15:GOSUB 170:PRINT''An Error has
Occurred:'';GOSUB 3020:GOTO 260
2910 -----THE 'save your data' routine at the end
2920 CLS
2930 X=12:Y=20:GOSUB 170
2940 PRINT''Have you saved the data (Y/N)?''
2950 PRINT:PRINT TAB(35);''Y = Terminate Program''
2960 PRINT TAB(35);''N = Go To Disk Access''
2970 PRINT:PRINT TAB(45);'' _____ ''
2980 G$=INKEY$:IF G$=''' THEN 2980
2990 IF G$='''N'' OR G$='''n'' THEN CT=20:GOTO 2050
3000 PRINT:PRINT''Thanks fer ur time!''
3010 END
3020 `----- Counting (delay) Loop -----
3030 X=23:Y=72:GOSUB 170:PRINT''Counting''
3040 X=23:FOR Y=1 TO 71:GOSUB 170
3050 FOR Z=1 TO 30:NEXT Z
3060 PRINT''+'''

```

```

3070 NEXT Y
3080 RETURN
3090 `----- Total all Categories before Printing Summary -
3100 FOR X=1 TO 19:TT(X)=0:NEXT X
3110 FOR CT=1 TO 19
3120 FOR X=1 TO ST(CT)
3130 TT(CT)=TT(CT)+AM(CT,X)
3140 NEXT X,CT
3150 RETURN
3160 `----- Subroutine ** 'Category full' -----
3170 CLS:X=10:Y=30:GOSUB 170:PRINT''Category Full:''
3180 GOSUB 3020:GOSUB 370
3190 RETURN
3200 `----- Subroutine ** 'Category Empty' -----
3210 `
3220 PRINT:PRINT CT$(CT);''      Empty:'' :GOSUB 3020:GOSUB 370
3230 RETURN
3240 ` Print page heading
3250 GOSUB 370:X=4:Y=1:GOSUB 170:PRINT W$(ZQ)
3260 X=6:Y=1:GOSUB 170:PRINT HD$
3270 X=7:Y=1:GOSUB 170:PRINT STRING$(80,196)
3280 RETURN
3290 `----- Line by Line or Loop Print ---
3300 PRINT I;TAB(13);DT$(CT,I);TAB(26);DS$(CT,I);TAB(58);
3310 PRINT USING ZW$;AM(CT,I);:PRINT TAB(74);CK$(CT,I)
3320 RETURN

```

Notes

There have been several requests on how to scroll backward and forward in GW BASIC and MS DOS. Apparently, the books on the subject are not too clear. However, Mr. Robert Hood, of Bremerton, Washington, has found the way to do it and kindly let us know about it. Our thanks to Mr. Hood for this information. Here is how it works:

1. List the desired line number (list 130)
2. Use arrow keys to move the cursor to the beginning of the listed line.
3. Press CTRL and X together to scroll in decreasing line numbers.
4. Press CTRL and Y together to scroll in increasing line numbers.

On another note, be careful about using remarks in DATA lines. The READ statement will probably read it like data. Here's an example:

```

10 DATA 1,2,3,4,5, ' daily receipts
20 DATA 6,7,8,9,10
30 FOR I=1 to 10
40 READ A$(I)
50 NEXT I
60 FOR I=1 to 10
70 PRINT A$(I);" ";
80 NEXT I
90 END
run
1 2 3 4 5 ' daily receipts 6 7 8 9 10
Oops!

```

Sort on Input

Sort as You Enter Information

David Leithauser, New Smyrna Beach, Florida. This is David's second appearance in CodeWorks. Here, he tells us about a method to keep sorted lists by sorting at the time of input. Two sample programs follow his discussion.

One of the features that most data base programs have is a sort function. This useful feature allows the program to arrange the data in alphabetical or numerical order. Having the data in alphabetical order makes it easy to find specific items on the printout. Having the data in numerical order allows you to see relationships at a glance. For example, you could have a program that records people's addresses sort the addresses by zip code. This would allow you to easily find all the people who live in one area. You can even have the program sort data by dates, so the data (such as your checkbook) will be in chronological order.

Unfortunately, sorting data takes a fairly long time. The amount of time required for a sort goes up rapidly as the amount of data increases. Sorting 10 items on a Tandy 1000 SX using a common bubble sort takes about .4 seconds. Sorting 100 items takes about 31 seconds. Sorting 200 items takes about 2 minutes. Sorting 500 items takes about 13.5 minutes. Depending somewhat on what type of sort technique you use, the time required for the sort is not affected much by whether the items are already close to being in order.

This can be particularly annoying when the nature of the data is such that you often add or delete one or two items. Imagine, for example, that you are keeping a phone directory of your friends or clients. About once a week you must add one or two names to the list. If you want to keep it sorted so that you can display an alpha-

betical listing any time, you must stop and wait for the sort each time you add those few names.

There is a way to avoid these long sorts. Instead of having the program add new data onto the end of the list and then sort the entire list, you can have it insert the new data into the list in the proper place as soon as you input it. This takes only a fraction of a second, and the data is always in the correct order.

The Usual Method

Before explaining how to insert the data into the proper place on input, let's take a look at the way data base programs usually handle data. Listing 1 is a very simple data base program with a bubble sort. Since this is for demonstration purposes only, I have not included most of the usual features, like a save on disk routine. This program inputs names and phone numbers, sorts them, and displays them.

Lines 200-230 input the data. Line 200 inputs the name. If the user just presses ENTER when the computer asks for the name, line 210 sends the computer back to the main menu. This is how you would finish putting in names.

Line 220 inputs the phone number. Line 230 then stores the data in the A and B arrays. N is the number of names in the file. Line 230 first increases N by 1. It then stores the name in the A array and the phone number in the B array.

Because of the DEFSTR statement in line 10, C and D and the A and B arrays are all string variables.

You can see that the data is not in alphabetical order as it is input. Instead, it is in the order the user inputs the data. Even once the program has sorted the data, any new data that you input will not be in the proper position.

Lines 300-340 delete data from the file. This is necessary to make corrections. Line 300 asks what to delete. The user would input the name of the person to delete from the phone directory. Lines 310-330 search for that name in the data file. When line 320 finds the name, it moves the last name and phone number in the file into that position. It then deletes the last name and number from the end of the file by setting them to an empty string. Finally, it decreases N by 1 to indicate that there is one less item in the file.

Notice that this disturbs the order of the items in the file. Deleting an item is as bad as adding one. Unfortunately, you could not simply set A(X) and B(X) to an empty string. That would leave the number of items in the file unchanged and cause the computer to print a blank line whenever it listed the data.

Lines 400-450 sort all the items. This is a simple bubble sort. Lines 500-530 list the data. Lines 100-140 are the main menu.

Sorting on Input

Now that we have seen one way to do it, let's look at the sort on input method. Listing 2 is the simple data base program modified to sort the data as it is input.

Lines 200-270 are the modified input routine. Lines 200-220 are the same as in Listing 1. They simply input the data to save in the arrays.

Line 230 handles the special case where the new data should go at the end of the data list instead of being inserted somewhere in the list. Remember that the data input by this routine is always sorted. Therefore, if C is greater than or

equal to (or comes after alphabetically in the case of strings) the last item in the file, then it is greater than all the items in the file. The THEN clause of line 230 puts the new data at the end of the list and increases the number of items (N) by 1.

If the program does need to insert the new data item into the data list, lines 240-260 find out where. They go through each item in the data file, looking for the first one that is more than C. When they find this item, the computer goes to line 270, with X retaining the number of that data item. Line 270 shifts each item in the data file up one. Line 280 then stores the new values in the newly vacated position in the file. It also increases N by one to record that one item has been added to the file. The computer then goes back to line 200 to get another data item.

Since the data is always in order as soon as you input it, there is no need for a sort function. You may notice that Listing 2 does not have one. The only thing that can disorganize your data now is if you delete something. Remember that the delete function in Listing 1 did reorganize the data. Therefore, Listing 2 needs a special delete function that preserves the order of the data.

Lines 300-340 are this delete function. Lines 300-310 and 330-340 are the same as in Listing 1. I have modified line 320, however, so that it preserves the order of the data. The Y FOR-NEXT loop shifts A(X) and B(X) to position N at the end of the data list and shifts everything above A(X) down one. Then A(N) and B(N) are set equal to empty strings, deleting them. N is decreased by 1 to indicate that there is one less data item, and the computer goes back to the main menu.

Drawbacks

There are two drawbacks to this sort-on-input method. The first is that there is a short pause as you enter each data item while the program puts the data item into the correct position. This pause is only about half a second for every 100 items already in the data file when you input the

new data. If you are inputting only a few items, this delay is trivial. Waiting a few seconds after each item as you input two or three items is certainly better than waiting hundreds of times this long for the computer to sort the data after you finish inputting those items.

Of course, if you are inputting many items at a time, you could become frustrated with the delay after each entry. In this case, you might be better off inputting the data with the procedure in Listing 1 and then letting the computer sort the data while you go to lunch. The sort-on-entry technique is most useful in special purpose data bases where you normally input only a few new items each day and you like to keep this data sorted at all times.

There is no reason you could not have both techniques available in one program. When the user selects "Input data" from the main menu, the program can ask "Sort on entry (Y/N)?" If the user presses Y, the program could branch to an input routine like the one in Listing 2. Otherwise, it goes to an input routine like the one in Listing 1. Such a program would have to include a sort routine for the people to use if they chose to input the data out of order.

The second disadvantage to the sort-on-input technique is that it only sorts by one property. This is fine for some special purpose data bases, where you always want the data sorted by the same property. Sometimes, however, you need a choice of which property to sort by. For example, you might want to be able to sort your checks by either date or amount.

The solution to this is to leave the user the option. You could have the program sort the data on input based on the property that you would normally want sorted. You could also include in the program a sort routine that would allow the users to sort by any property, for those rare occasions when they want it sorted some other way.

If you do allow the users to sort by various properties, you must warn them to resort the

data by the usual property when they are finished. The sort-on-input routine does not work properly if the data is not already sorted by the usual property. The fastest way to restore the data to the usual order is to save it on disk before sorting it by any other property. Then you can sort the data by some other property, output a paper listing of it, and then reload the data from the disk before you input any more data.

Different Computer Versions

Although I wrote these programs on an IBM clone, I used a very limited subset of BASIC. They will therefore work on most computers. You may need to modify the SWAP statement that I used in a few lines. This statement swaps the values of two variables. If your computer does not have the SWAP statement, you can replace SWAP V1,V2 with $T=V1:V1=V2:V2=T$ where V1, V2, and T are any variables as long as they are the same variable type. For example, you would replace SWAP A(Y),A(Y-1) with $A=A(Y):A(Y)=A(Y-1):A(Y-1)=A$.

You may also need to remove the DEFSTR A-D and DEFINT N-Z statements from the program for some computers. DEFSTR A-D causes all variables starting with A through D to be strings. You can achieve the same effect by placing a \$ after any variable name that begins with A through D. DEFINT N-Z causes all variables starting with N through Z to be integers. You may produce the same results by putting % after each variable name that begins with N through Z. You may also simply omit this entirely from your program. The program will simply run a bit slower.

Listing 1 for Sort on Input

```
5 REM * inpsort1.bas * D Leithauser *
10 DEFINT N-Z:DEFSTR A-D:DIM A(1000),B(1000):CLS
99 ` Menu
100 CLS
110 PRINT `` Menu:'' :PRINT:PRINT ``1) Input'' :PRINT ``2) Delete'' :PRINT
    ``3) Sort'' :PRINT ``4) Display'' :PRINT ``5) End'' :PRINT
120 INPUT ``Number of your choice'' ;T
130 ON T GOTO 200,300,400,500,600
140 GOTO 110
199 ` Input data
200 INPUT ``Name'' ;C
210 IF C=```` THEN 100
220 INPUT ``Phone'' ;D
230 N=N+1:A(N)=C:B(N)=D:GOTO 200
299 ` Delete data
300 INPUT ``Delete what'' ;C
310 FOR X=1 TO N
320 IF A(X)=C THEN A(X)=A(N):B(X)=B(N):A(N)=````:B(N)=````:N=N-1:GOTO
    100
330 NEXT X
340 PRINT C;`` not found.`` :GOTO 300
399 ` Sort data
400 FOR X=1 TO N-1
410 FOR Y=N TO X+1 STEP -1
420 IF A(Y)<A(X) THEN SWAP A(Y),A(X):SWAP B(Y),B(X)
430 NEXT Y
440 NEXT X
450 GOTO 100
499 ` Display data
500 CLS:FOR X=1 TO N:PRINT X;TAB(9);A(X);TAB(30);B(X):NEXT X
510 PRINT ``Press Space Bar for Menu.`` ;
520 IF INKEY$<>`` `` THEN 520
530 GOTO 100
599 ` End program
600 END
```

Listing 2 for Sort on Input

```
5 REM * inpsort2.bas * D Leithauser *
10 DEFINT N-Z:DEFSTR A-D:DIM A(1000),B(1000):CLS
99 ` Menu
100 CLS
110 PRINT `` Menu``:PRINT:PRINT ``1) Input``:PRINT ``2) Delete``:PRINT
    ``3) Display``:PRINT ``4) End``:PRINT
120 INPUT ``Number of your choice``;T
130 ON T GOTO 200,300,400,500
140 GOTO 110
199 ` Input data
200 INPUT ``Name``;C
210 IF C=```` THEN 100
220 INPUT ``Phone``;D
230 IF C=>A(N) THEN N=N+1:A(N)=C:B(N)=D:GOTO 200
240 FOR X=1 TO N
250 IF C<A(X) THEN 270
260 NEXT X
270 FOR Y=N+1 TO X+1 STEP -1:SWAP A(Y),A(Y-1):SWAP B(Y),B(Y-1):NEXT
    Y
280 A(X)=C:B(X)=D:N=N+1:GOTO 200
299 ` Delete data
300 INPUT ``Delete what``;C
310 FOR X=1 TO N
320 IF A(X)=C THEN FOR Y=X TO N-1:SWAP A(Y),A(Y+1):SWAP B(Y),B(Y+1):
    NEXT Y:A(N)=````:B(N)=````:N=N-1:GOTO 100
330 NEXT X
340 PRINT C;`` not found.``:GOTO 300
399 ` Display data
400 CLS:FOR X=1 TO N:PRINT X;TAB(9);A(X);TAB(30);B(X):NEXT X
410 PRINT ``Press Space Bar for Menu.``;
420 IF INKEY$<>`` `` THEN 420
430 GOTO 100
499 ` End program
500 END
```

Flow.Bas

A Line Number Reference Utility

Staff Project. This program will chart the flow of your program, giving line number references and whether or not the GOTOs and GOSUBs are conditional or directed. For best results, it should be compiled.

Flow.Bas is a utility program that works on your programs saved in ASCII and prints out a list of line numbers and associated GOTOs and GOSUBs. In addition, it will tell you whether or not those program branches were conditional or "hard." By "hard" we mean an unconditional or directed program branch.

The program comes in handy when you have an especially convoluted program flow and are trying to figure out where it's going next. This can happen on programs you have written a long time ago, or when trying to figure out someone else's program. Provision has been made to print the output directly to the screen, or, to the screen and printer as well.

One of the things we had to consider when designing this program was that not all GOTOs say GOTO. A GOTO can be implied with the THEN and ELSE statements. Also, a line number implying a GOTO can follow the RESUME and RETURN statements as well. On the other hand, an IF anywhere in a line automatically makes a conditional out of any branch which follows on that same line.

Basically, the idea of the program is to open the target file (which must have been saved in ASCII) and read in one line at a time. Then examine that line looking for specific keywords which would imply that a line number follows. When that happens, we determine what the

branch statement was (a GOTO or GOSUB) and stuff three arrays with the line number calling the branch, the line number to branch to and the type of branch it was. After the completion of the examination of the target program, we can then take the three arrays and consolidate the information in them and print out a summary.

As you have already guessed, the program is slow because it must examine every character in every line that is not a remarked line. This program is therefore an excellent candidate for the compilation process. We did, and it increased its speed by a factor of about 20.

Program Details

Initialization starts at line 170, where if your BASIC is prior to version 5.0, you should clear some string space. In line 180 we dimension the A, B and C arrays at 650. This allows examining a program that is up to 650 lines long. You can change these if you find it necessary. Array T is also dimensioned here, but if you use the SWAP command (lines 640-660 and lines 780-790) you don't even need the T array. If your computer does not have the SWAP command, you will need to dimension array T.

Lines 190 and 200 format some print strings to be used later with PRINT USING statements. Line 210 defines three string variables to hold words we will need several times later.

After the usual CodeWorks opening we are asked the name of the file to examine, and this is held in variable FF\$. We are then asked if we want printed output also and variable PR holds the indicator: PR=0 says no printer while PR=1 says we want printer output.

Most of what happens in this program occurs in the following lines, from 360 to 570. But first, in line 340, we open the target file for input. There are two loops in this block of code, one of them is inside the other. Let's cruise through the loops just like the program would and see what happens.

As is usual in computing, the first thing you do is look for a way out, and line 370 does that by testing for end-of-file on the input (target) file. If we are not at EOF of the input file, line 380 reads in one line of code and calls it A\$. In line 390 we find the length of the A\$ just read in. We won't use that length immediately, but we will a bit later. In line 400 we find the space after the line number of the line just read in, and in line 410 we let variable LN contain the line number itself. Note that we have to take the VAL of it because it was read in as a string.

In lines 420 and 430 we check the first item in the line after the space following the line number to see if it is a remark line. If the line is a remark, we don't even want to mess with it so we jump right down to line 560 and get the next line from the file to look at.

The inner loop starts at line 440 and goes to 540. In this loop we are taking the line of code from the space after the line number to the end, and we will let C\$ be a little "window" that looks at four characters at a time. Meantime, we will have another "window" that is two characters long to find the keyword "IF." S\$ is going to be our four-character window (in line 460), and in line 470 we look at two characters of the four in the C\$ window to see if they are "IF." If they are, then flag F1 gets set to one. This will later tell us that the GOTO or GOSUB was "conditional."

As we step down the line, C\$ will examine four characters at a time. If those four characters happen to be "THEN", "ELSE", "GOTO", "OSUB", "SUME" or "TURN" then we jump out and go to a subroutine to do some further processing. Before we go there though, let's look at those terms a bit. THEN and ELSE are implied GOTOs. Because they are unique, we can determine GOSUB, RESUME and RETURN by just looking at the last four characters. But why RESUME and RETURN? Well, it's because they can both be followed by line numbers, and therefore, another implied GOTO. Let's assume we have found one of our keywords. In every case, flag F1 will be set to either a one or a two, and we GOSUB to 1010. Remember that the Q loop is still sitting at the point where we found the keyword.

At subroutine 1010 the first thing we do is find the value (VAL) of the nine characters following our keyword (in line 1020.) We know that line numbers can only be a maximum of five characters long, but some people like to put a couple of extra spaces in their lines - so nine characters ought to catch them. In any case, VAL will not return anything for alpha characters, so we will get the line number reference we want. In line 1030, if A is equal to zero, it means that what followed the keyword was no number. That can happen in lines like: IF X=1 THEN IF Q=2..., in which case the THEN is not followed by a line number reference. In this case we simply return to where we came from.

If we do have a line number reference, however, then we will increment our J count by one. J is going to be the count for the three arrays we mentioned earlier, in which we will keep all the information we glean from the lines. The next thing we do in line 1050, is jump the Q counter (way back in line 440) past the length of the line number reference. In other words, we are going to advance the Q count to the end of the line number reference we just found. Next, in lines 1080 through 1150, we look at the flags that have been set, and depending on what they are we can print a line on the screen (or printer, if

selected) that says: 370 GOTO 570 CONDITIONAL. In this case, we got the 370 line number from LN back in line 410, the GOTO because we got here from line 500, the 570 came from variable A which we just found, and the CONDITIONAL came from the fact that flag F had been a one. Now that we know all that, we stuff the proper values into the A, B and C arrays, using J as the array counter. Then we return.

Since there may be further references to line numbers in A\$, we continue looking down the line to find them. When we have finished one line of A\$, we clear C\$, A, F and F1 to zero and get ready for the next line from the target file.

When all the lines in the target file have been examined we close the file at line 570 and then sort the A, B and C arrays into ascending order on array C. If your computer supports the SWAP command, you can use the optional lines in lines

**Flow.Bas for GW BASIC
changes for other machines
follows the listing.**

```
100 REM * Flow.Bas * Examines flow of a program *
110 REM * Written for CodeWorks Magazine, 3838 South Warner St.
120 REM * Tacoma, WA 98409 (206)475-2219 voice 475-2356 download
130 REM * (C)1988 80-NW Publishing Inc. Placed in public domain.
140 REM * Suggestion: For max speed, compile this program.
150 `
160 `Do some initialization
170 `CLEAR 1000 ` only if your BASIC is prior to ver. 5.0
180 DIM A(650),B(650),C(650),T(650)
190 F1$='##### \ \ ##### \ \ '
200 F2$='##### \ \ #####'
210 T1$='GOTO':T2$='GOSUB':T3$='CONDITIONAL'
220 CLS
```

640 to 660. It will speed things up a bit if you do.

Next, we do a second level sort on the B array. This is a "sort within a sort" and arranges all items in similar C array entries into ascending order. See Beginning BASIC in this issue for a complete look at two-level sorting.

The consolidate routine at line 850 comes next. Array C tells us if an entry in the corresponding A or B array is a conditional GOTO, hard GOTO, conditional GOSUB or a hard GOSUB. Since the numbers go from one to four, we can use them in an ON GOTO statement, as in line 880. The code from lines 930 to 950 arranges the output into something that is easy to read.

Although not the most used program in the world, it really does come in handy when you need to follow the flow of an unknown or forgotten program. You might also use it to help you restructure a badly written program and perhaps get rid of some unnecessary GOTOS.

```

230 PRINT STRING$(22,45);'' The CodeWorks '';STRING$(23,45)
240 PRINT''          PROGRAM FLOW ANALYZER
250 PRINT''          shows all program branches
260 PRINT STRING$(60,45)
270 PRINT
280 INPUT''What ASCII file do you wish to examine'';FF$
290 INPUT''Do you wish printer output also (y/n)'';PR$
300 IF PR$='Y' OR PR$='y' THEN PR=1 ELSE PR=0
310 IF PR=1 THEN LPRINT''Examining program '';FF$:LPRINT'' ''
320 '
330 ' Open the file and read one line at a time
340 OPEN''I'',1,FF$
350 CLS
360 FOR I=1 TO 650
370   IF EOF(1) THEN 570
380   LINE INPUT #1,A$
390   L=LEN(A$)+1
400   S=INSTR(A$,''' ''
410   LN=VAL(LEFT$(A$,S))
420   IF MID$(A$,S+1,3)='REM' THEN 560
430   IF MID$(A$,S+1,1)=''' THEN 560
440   FOR Q=S TO L
450     C$=C$+MID$(A$,Q,1)
460     S$=RIGHT$(C$,4)
470     IF RIGHT$(S$,2)='IF' THEN F=1
480     IF S$='THEN' THEN F1=1:GOSUB 1010
490     IF S$='ELSE' THEN F1=1:GOSUB 1010
500     IF S$='GOTO' THEN F1=1:GOSUB 1010
510     IF S$='OSUB' THEN F1=2:GOSUB 1010
520     IF S$='SUME' THEN F1=1:GOSUB 1010
530     IF S$='TURN' THEN F1=1:GOSUB 1010
540   NEXT Q
550   C$='''':A=0:F=0:F1=0
560 NEXT I
570 CLOSE 1
580 '
590 REM * sort the arrays
600 FL=0
610 FOR I=1 TO J-1
620   L=I+1
630   IF C(I)<C(L) THEN 680
640   T(I)=C(I):C(I)=C(L):C(L)=T(I) 'or SWAP C(I),C(L)
650   T(I)=A(I):A(I)=A(L):A(L)=T(I) 'or SWAP A(I),A(L)

```

```

660 T(I)=B(I):B(I)=B(L):B(L)=T(I) 'or SWAP B(I),B(L)
670 FL=1
680 NEXT I
690 IF FL=1 THEN 600
700 PRINT
710 `
720 ` * now do a second level sort *
730 FL=0
740 FOR I=1 TO J-1
750 L=I+1
760 IF C(I)<>C(L) THEN 810
770 IF B(I)=<B(L) THEN 810
780 T(I)=A(I):A(I)=A(L):A(L)=T(I) ` or SWAP A(I),A(L)
790 T(I)=B(I):B(I)=B(L):B(L)=T(I) ` or SWAP B(I),B(L)
800 FL=1
810 NEXT I
820 IF FL=1 THEN 730
830 PRINT
840 `
850 REM * consolidate routine
860 FOR I=1 TO J
870 L=I-1
880 ON C(I) GOTO 890,900,910,920
890 D$='' COND GOTO'':GOTO 930
900 D$='' hard GOTO'':GOTO 930
910 D$=''COND GOSUB'':GOTO 930
920 D$=''hard GOSUB''
930 IF B(I)=B(L) THEN PRINT A(I);'' ``;ELSE PRINT:PRINT ``Line'';
B(I);''is a ``;D$;'' called from:'';A(I);'' ``;
940 IF PR=0 THEN 960
950 IF B(I)=B(L) THEN LPRINT A(I);'' ``;ELSE LPRINT'' ``:
LPRINT''Line'';B(I);''is a ``;D$;'' called from:'';A(I);'' ``;
960 NEXT I
970 IF PR=1 THEN LPRINT CHR$(13):LPRINT''Done.''
980 PRINT
990 PRINT ``Done'':END
1000 `
1010 REM * subroutine to find trailing numbers
1020 A=VAL(MID$(A$,Q+1,9))
1030 IF A=0 THEN RETURN
1040 J=J+1
1050 Q=INSTR(Q,A$,STR$(A))+LEN(STR$(A))-1
1060 `

```

```

1070 'and then print the results
1080 IF F1=1 AND F=1 THEN PRINT USING F1$;LN;T1$;A;T3$:A(J)=LN:
      B(J)=A:C(J)=1
1090 IF PR=1 AND F1=1 AND F=1 THEN LPRINT USING F1$;LN;T1$;A;T3$
1100 IF F1=1 AND F=0 THEN PRINT USING F2$;LN;T1$;A:A(J)=LN:B(J)=A:
      C(J)=2
1110 IF PR=1 AND F1=1 AND F=0 THEN LPRINT USING F2$;LN;T1$;A
1120 IF F1=2 AND F=1 THEN PRINT USING F1$;LN;T2$;A;T3$:A(J)=LN:
      B(J)=A:C(J)=3
1130 IF PR=1 AND F1=2 AND F=1 THEN LPRINT USING F1$;LN;T2$;A;T3$
1140 IF F1=2 AND F=0 THEN PRINT USING F2$;LN;T2$;A:A(J)=LN:B(J)=A:
      C(J)=4
1150 IF PR=1 AND F1=2 AND F=0 THEN LPRINT USING F2$;LN;T2$;A
1160 RETURN

```

Flow.Bas change lines for Tandy I and III

```

Changed->100 REM * Flow/Bas * Examines flow of a program *
Changed->170 CLEAR 1000 ' only if your BASIC is prior to ver. 5.0
Changed->190 F1$=' '##### %      % ##### %      %''
Changed->200 F2$=' '##### %      % #####''
Changed->440   FOR Q=S+1 TO L

```

Flow.Bas change lines for Tandy II and IV

```

Changed->100 REM * Flow/Bas * Examines flow of a program *
Changed->440   FOR Q=S+1 TO L

```

Pay2.Bas

An Updated Pay.Bas with New Changes

Staff Project. Thanks to the many of you who suggested changes to Pay.Bas. These changes, and others, are now incorporated into the new Pay2.Bas program.

Way back in Issue 4 (March 1986) we published the program Pay.Bas. In Issue 14 (November 1987) it went through its first revision. That revision was so that more employees could be added and to fix the case where you couldn't get rid of an employee.

Hanging in our craw was the hokey way you had to initialize the program the first time you used it. Typing stuff in command mode and ignoring an error message is just not cool. Aside from that, some readers gave us the works for the way the pay stub and reports printed cents. They said it just wasn't the way to do things like that, and they were right.

The changes included with this article are intended to fix all of that. In addition, these changes can be made whether or not you have made the changes listed in Issue 14. Further, you can make these changes to your program without the need to start all over and enter all of your employees again. For those of you who do not have the original program, a complete listing of the new version is included. Also included, are the change lines for those who already have the program and just want to update it. Even if you are typing the program in for the first time, you might want to refer to the change lines, since they are printed sideways so you can see the layout of the print format lines.

The first thing is to change the name of the program to PAY2.Bas, to show that it is the updated version. Next, we added an error trap,

and in the added lines from 3240 on, we create the "Paynames" file if it doesn't already exist. It all happens without you even seeing it. No more command mode inputs and ignored error messages.

The changes you will notice in the everyday operation of the program are in added and changed lines from 171 to 1700. Lines 172, 173 and 174 set up three print formatting lines using the PRINT USING format. Line 171 is a guide line so that you can get the spacing right on the following three lines. We are pasting the listing in sideways so that you will see the continuous lines without wrapping. Spacing is rather important in these lines so that amounts line up with headings. Actually, it somewhat simplifies lines 910-1700. Now, all your dollar amounts will line up nicely and the cents will always print out in two places like they should. Line 2660 was changed to keep the words from running together when you clear the quarter or year. It's what they call a "cosmetic" change only.

If you intend to make these changes to an active payroll program, it may be best if you backed up your original copy with all your employees on it and get this working on the backup first. Then make the changes to your "live" copy. There is nothing in the world like a messed up payroll. You catch it from all angles when that happens, and we are way past the point where people will believe that it was the computer that messed up.


```

100 REM ** PAY2.BAS ** FOR CODEWORKS MAGAZINE updated Aug 88
110 REM ** 3838 S. WARNER ST. TACOMA, WA 98409 (206)475-2219
120 REM ** PLEASE DO NOT REMOVE THESE CREDIT LINES
130 REM ** 1st time initialization not required. See CodeWorks Issue 4
140 REM ** for complete details and operating instructions, including
150 REM ** how to reinstate a deleted employee.
160 CLEAR 1000: ' Use only if your machine needs to clear space.
165 NE=15 'sets max number of employees you can have.
166 N1=NE
168 ON ERROR GOTO 3240
170 DIM E(18),E$(NE+1) : ' E$( ) sets the max number of employees plus
171 ' 1234567890123456789012345678901234567890123456789012345678901
172 XA$=' ' ##   ##.##   ##   $##.##   $$#####.##   $$#####.##
173 XB$=' ' ###.##   ###.##   ###.##   ###.##   ###.##
174 XC$=' ' #####   #####.##   #####.##   #####.##   ###.##   ###.##   ###.##
      #####.##
180 INPUT 'Enter the date (any way you like)';D$
190 REM ** If you have DATE$ change above line to: D$=DATE$
200 ' ----- Define some important variables -----
210 ' E$ current employee's file name. Also used in array E$( )
220 ' E1$ is any employee's full name.
230 ' S$ is any employee's social security number.
240 ' E( ) is any employee's data array - see edit/review code.
250 ' HO = hours worked this pay period.
260 ' VA = VC*HO, how much vacation was earned this pay period.
270 ' HT = Vacation Hours Taken this pay period.
280 ' GP = Gross Pay for this pay period.
290 ' CF = Current pay period FICA deduction.
300 ' CT = Current pay period FedTax deduction.
310 ' CS = Current pay period State Tax deduction.
320 ' CM = Current pay period Medical deduction.
330 ' CL = Current pay period Workman's Compensation deduction
340 ' NP = Net Pay for the current period.
350 GOSUB 3010 : ' Read in the employee file names file.
360 GOSUB 2920 : ' Write the file names file back out.
370 VC=.03846 : ' Vac earned per hour - given 2 weeks per year.
380 FR=.0705 : ' FICA rate withheld from each employee.
390 WC=.0276 : ' State Workman's Compensation deduction rate.
400 FM=37800! : ' Maximum gross from which FICA can be deducted.
410 DEF FNI(X)=INT(X*100+.5)/100 : ' Define rounding as a function.
420 C1$='Magarac's Widget Company' : ' Put your company name here.
430 C2$='1234 Tool Steel Road' : ' And your address,
440 C3$='Skunk Hollow, WA 98000' : ' and city state and zip too.
450 CLS
460 PRINT STRING$(22,'-'); ' The CodeWorks ` `;STRING$(23,'-')
470 PRINT '          S M A L L   B U S I N E S S   P A Y R O L L
480 PRINT '          for companies where you know them by their first name
490 PRINT STRING$(60,'-')
500 PRINT

```



```

510 PRINT TAB(10);''1 - Do the Payroll''
520 PRINT TAB(10);''2 - Edit or Review a Pay Record''
530 PRINT TAB(10);''3 - Print Payroll Reports''
540 PRINT TAB(10);''4 - Add or Delete Employees''
550 PRINT TAB(10);''5 - End of Quarter/Year Clearing''
560 PRINT TAB(10);''6 - End Session''
570 PRINT
580 PRINT''Your choice'';
590 X$=INKEY$:IF X$=''' THEN GOTO 590
600 X=VAL(X$):IF X<1 OR X>6 THEN GOTO 590
610 ON X GOTO 630,1060,1370,1920,2410,2690
620 END:REM -----> Do the Payroll module **
630 CLS:PRINT TAB(10);'' **** DO THE PAYROLL ****''
640 PRINT
650 IF N1=<5 THEN PRINT''One 8 x 11 sheet of paper will do.'':GOTO 670
660 PRINT''You will need two 8 x 11 sheets for the pay stubs.''
670 PRINT''Adjust your paper, your printer should be set for 66 line''
680 PRINT''pages and 60 lines per page, width 80 columns.''
690 PRINT
700 PRINT''To skip an employee, enter 0 for hours worked.''
710 PRINT
720 FOR I=1 TO N1
730 E$=E$(I):HT=0:GOSUB 2830
740 PRINT''Hours ``;E$;'' worked this period'';:INPUT HO:IF HO=0 THEN
    GOTO 1020
750 PRINT''Did ``;E$;'' use any vacation this period (Y/N)'';:INPUT X$
760 IF X$<>'Y' AND X$<>'y' THEN GOTO 790
770 INPUT''How many hours were taken'';HT
780 E(4)=E(4)-HT
790 VA=VC*HO:E(4)=E(4)+VA:GP=E(0)*HO:CF=GP*FR:CT=GP*E(1):
    CS=GP*E(2):CM=E(3):CL=HO*WC
800 IF E(10)=>FM THEN CF=0
810 NP=GP-(CF+CT+CM+CL+CS)
820 E(7)=E(7)+CF:E(9)=E(9)+CT:E(6)=E(6)+CS:E(5)=E(5)+CL:E(8)=E(8)+
    CM
830 E(17)=E(17)+HO:E(11)=E(11)+GP:E(12)=E(12)+CF:E(13)=E(13)+CT:
    E(14)=E(14)+CL:E(15)=E(15)+CS:E(16)=E(16)+NP:E(10)=E(10)+GP:
    E(18)=E(18)+CM
840 CF=FNI(CF):CL=FNI(CL):CT=FNI(CT):CS=FNI(CS):CM=FNI(CM):
    NP=FNI(NP):GP=FNI(GP):E(10)=FNI(E(10))
850 FOR K=5 TO 18:E(K)=FNI(E(K)):NEXT K
860 LPRINT C1$+''' ``+C2$+''' ``+C3$
870 LPRINT E1$;TAB(26);S$;TAB(40);''Pay period ending: ``;D$
880 LPRINT'' ``
890 E(4)=INT(E(4)*10+.5)/10
900 LPRINT'' HOURS'';TAB(7);''VacAvail'';TAB(17);''Taken'';TAB(25);''Rate'';
    TAB(35);''GrossPay'';TAB(50);''NetPay''
910 LPRINT USING XA$;HO;E(4);HT;E(0);GP;NP
920 LPRINT'' ``
930 LPRINT TAB(32)'' -- Deductions ---''
940 LPRINT TAB(20);''FICA'';TAB(30);''FedTax'';TAB(40);''StateTax'';

```

```

        TAB(50);''Medical'';TAB(60);''WorkmnComp''
950  LPRINT''Current Period --'';:LPRINT USING XB$;CF;CT;CS;CM;CL
960  LPRINT''Year to Date ----'';:LPRINT USING XB$;E(7);E(9);E(6);E(8);
      E(5)
970  LPRINT''YTD Gross $'';E(10)
980  LPRINT STRING$(64,45)
990
1000  GOSUB 2750
1010  IF I=6 THEN LPRINT CHR$(12)
1020  NEXT I
1030  LPRINT CHR$(12)
1040  GOTO 450
1050  END:REM -----> Edit or Review a Pay Record module **
1060  CLS:PRINT TAB(10);''EDIT/REVIEW A PAY RECORD''
1070  PRINT
1080  INPUT''Enter Employee's First name '';E$
1090  GOSUB 3090
1100  GOSUB 2830
1110  CLS:PRINT TAB(20);''  EDIT/REVIEW''
1120  PRINT''Filename is: '';E$
1130  PRINT''1-Name:'';E1$;TAB(32);''11-YTD Med ded ---- '';E(8)
1140  PRINT''2-SS# ----- '';S$;TAB(32);''12-YTD FedTax ded -- '';E(9)
1150  PRINT''3-Rate/Hr -- '';E(0);TAB(32);''13-YTD Gross pay -- '';E(10)
1160  PRINT''4-FedTax % - '';E(1);TAB(32);''14-Gross this qtr -- '';E(11)
1170  PRINT''5-StTax %  -- '';E(2);TAB(32);''15-FICA this qtr -- '';E(12)
1180  PRINT''6-Med ded -- '';E(3);TAB(32);''16-FedTax this qtr - '';E(13)
1190  PRINT''7-Vac avail- '';E(4);TAB(32);''17-WkComp this qtr - '';E(14)
1200  PRINT''8-YTD WkComp '';E(5);TAB(32);''18-StTax this qtr -- '';E(15)
1210  PRINT''9-YTD StTax- '';E(6);TAB(32);''19-Net pay this qtr- '';E(16)
1220  PRINT''10-YTD FICA- '';E(7);TAB(32);''20-Hours this qtr -- '';E(17)
1230  PRINT TAB(32);''21-Med ded this qtr- '';E(18)
1240  PRINT
1250  INPUT''Correct which item number, enter 0 for none '';XX
1260  IF XX=0 AND X2$<>'''' THEN PRINT''You chose to change something,
      which number'';:INPUT XX:IF XX=0 THEN GOTO 1250 ELSE GOTO 1280
1270  IF XX=0 THEN FOR I=0 TO 18:E(I)=0:NEXT I:GOTO 450
1280  IF XX<1 OR XX>21 THEN GOTO 1250
1290  LINE INPUT''Enter the correct information '';X$
1300  IF XX=1 THEN E1$=X$
1310  IF XX=2 THEN S$=X$
1320  IF XX>2 THEN E(XX-3)=VAL(X$)
1330  INPUT''Any more changes (Y/N)'';X2$
1340  IF X2$='''Y'' OR X2$='''y'' THEN GOTO 1250
1350  GOSUB 2750:X2$='''':GOTO 1110
1360  END:REM -----> Print Payroll Report module **
1370  CLS:PRINT TAB(10);''  PAYROLL REPORTS''
1380  PRINT
1390  PRINT''Get your printer ready''
1400  PRINT
1410  PRINT''1 - Report of Amounts Paid/Withheld (IRS 941 info)''
1420  PRINT''2 - Employee Information Report.''

```

```

1430 PRINT''3 - To return to main menu.''
1440 PRINT
1450 PRINT''Your Choice'';
1460 X$=INKEY$:IF X$=''' THEN GOTO 1460
1470 X=VAL(X$):IF X<1 OR X>3 THEN GOTO 1460
1480 LPRINT TAB(20);C1$
1490 LPRINT TAB(20);C2$
1500 LPRINT TAB(20);C3$
1510 LPRINT'' ''
1520 ON X GOTO 1530,1750,450
1530 CLS:PRINT''This report will show accumulated amounts during the''
1540 PRINT''quarter. It is used primarily to have a record and to''
1550 PRINT''calculate IRS 941 liability. It will fit on one page.''
1560 PRINT
1570 PRINT''Press ENTER when ready'';:INPUT X
1580 LPRINT''Accumulated Pay Amounts Report for period ending '';D$
1590 LPRINT'' ''
1600 LPRINT''Hours'';TAB(10);''Gross'';TAB(20);''FICA'';TAB(30);''FedTax'';
    TAB(40);''WkComp'';TAB(47);''StTax'';TAB(56);''Med'';TAB(65);''NetPay''
1610 LPRINT'' ''
1620 FOR I= 1 TO N1
1630   E$=E$(I):GOSUB 2830
1640   LPRINT E1$,S$
1650   LPRINT USING XC$;E(17);E(11);E(12);E(13);E(14);E(15);E(18);
    E(16)
1660   T1=T1+E(17):T2=T2+E(11):T3=T3+E(12):T4=T4+E(13):T5=T5+E(14):
    T6=T6+E(15):T7=T7+E(18):T8=T8+E(16)
1670 NEXT I
1680 LPRINT'' ''
1690 LPRINT''*** TOTALS ***''
1700 LPRINT USING XC$;T1;T2;T3;T4;T5;T6;T7;T8
1710 LPRINT'' ''
1720 LPRINT''Total 941 liability so far is $'';(2*T3)+T4
1730 LPRINT CHR$(12)
1740 GOTO 450
1750 CLS:PRINT''Employee List Report''
1760 PRINT
1770 PRINT''This report provides a list of your employees and their''
1780 PRINT''fixed deductions. It will fit on one 8 x 11 page.''
1790 PRINT
1800 PRINT''Press ENTER when ready'';:INPUT XX
1810 LPRINT''Employee List as of '';D$
1820 LPRINT'' ''
1830 LPRINT''Name'';TAB(26);''SS#'';TAB(38);''Rate'';TAB(50);''FedTax%'';
    TAB(60);''StTax%'';TAB(70);''Med Ded''
1840 LPRINT'' ''
1850 FOR I=1 TO N1
1860   E$=E$(I):GOSUB 2830
1870   LPRINT E1$;TAB(26);S$;TAB(38);E(0);TAB(50);E(1);TAB(60);E(2);
    TAB(70);E(3)
1880 NEXT I

```

```

1890 LPRINT CHR$(12)
1900 GOTO 450
1910 END:REM -----> Add or Delete Employee module **
1920 CLS:PRINT TAB(10);'' ADD OR DELETE AN EMPLOYEE PAY RECORD ''
1930 PRINT
1940 PRINT TAB(10);''1 - To ADD a New Employee Record.''
1950 PRINT TAB(10);''2 - To DELETE an Employee Record.''
1960 PRINT TAB(10);''3 - To return to main menu.''
1970 PRINT''Your Choice'';
1980 X1$=INKEY$:IF X1$=''' THEN GOTO 1980
1990 X1=VAL(X1$):IF X1<1 OR X1>3 THEN GOTO 1980
2000 ON X1 GOTO 2010,2230,450
2010 CLS:PRINT TAB(10);''ADD a new Employee Record''
2020 PRINT
2030 PRINT''Follow the prompts to create a new employee record.
2040 PRINT''Enter zero amounts where applicable.''
2050 LINE INPUT''Employee Full Name ----- '';E1$
2060 LINE INPUT''Social Security Number ----- '';S$
2070 INPUT''Hourly Rate of pay ----- '';E(0)
2080 INPUT''Federal Tax Deduction % (i.e., .12)- '';E(1)
2090 INPUT''State Tax Deduction % (i.e., .08)-- '';E(2)
2100 INPUT''Medical Insurance per period ----- '';E(3)
2110 PRINT
2120 INPUT''Enter Employee File name '';E$
2130 FOR I=1 TO NE
2140 IF E$(I)=E$ THEN PRINT''That name already exists, use another'':
      GOTO 2120
2150 NEXT I
2160 INPUT''Press ENTER to create this record'';XX
2170 GOSUB 2750
2180 FOR I=1 TO 10
2190 IF E$(I)=''' OR E$(I)='''ONE'' THEN E$(I)=E$:GOTO 2210
2200 NEXT I
2210 GOSUB 2920
2220 GOTO 450
2230 CLS:PRINT TAB(10);'' BEFORE YOU DELETE AN EMPLOYEE!''
2240 PRINT
2250 PRINT''You must carry an employee through the current quarter''
2260 PRINT''so that your reports used for IRS forms 941 will be''
2270 PRINT''correct. To carry a terminated employee through''
2280 PRINT''the end of the quarter, when you do the payroll, simply''
2290 PRINT''enter 0 for hours worked. It will then skip over that''
2300 PRINT''employee. NOW -- if you still want to delete, go ahead:''
2310 PRINT''Answer the next question with 0 if you opt not to delete.''
2320 PRINT
2330 INPUT''Enter File name of employee to delete '';E$
2340 IF E$='''0'' THEN GOTO 1920
2350 GOSUB 3090
2360 FOR I=1 TO NE
2370 IF E$(I)=E$ THEN E$(I)='''
2380 NEXT I

```

```

2382 FOR I=1 TO NE
2384   L=I+1
2386   IF E$(I)=''' THEN E$(I)=E$(L):E$(L)='''
2388 NEXT I
2390 GOSUB 2920:GOTO 350
2400 END:REM -----> End of Quarter/Year Clearing module **
2410 CLS:PRINT TAB(20);'' QUARTER / YEAR END CLEAR ''
2420 PRINT
2430 PRINT''Be sure you have printed your payroll reports for the''
2440 PRINT''quarter before clearing. Clearing the quarter will remove''
2450 PRINT''all quarterly data for ALL employees. Clearing the year''
2460 PRINT''will clear everything except basic employee data for''
2470 PRINT''ALL employees. Use Edit/Review option to verify clear.''
2480 PRINT
2490 PRINT''At the end of the year, clearing the year will clear the''
2500 PRINT''last quarter as well. >> Print your reports first! <<''
2510 PRINT''To prevent inadvertent clearing, you must type in the''
2520 PRINT''word QUARTER or YEAR, otherwise, you will be sent''
2530 PRINT''back to the main menu.
2540 PRINT
2550 PRINT''CLEAR what: '';:INPUT X$
2560 IF X$<>'''QUARTER'' AND X$<>'''YEAR'' THEN GOTO 450
2570 IF X$='''QUARTER'' THEN Q1=11 ELSE Q1=5
2580 FOR I=1 TO N1
2590   E$=E$(I):GOSUB 2830
2600   PRINT''Clearing the '';X$;' for: '';E$
2610   FOR J=Q1 TO 18
2620     E(J)=0
2630   NEXT J
2640   GOSUB 2750
2650 NEXT I
2660 PRINT''All '';X$;' amounts have been cleared. Press ENTER'';:INPUT
    X
2670 GOTO 450
2680 END:REM -----> End Session module **
2690 CLS:PRINT TAB(10);''END SESSION''
2700 PRINT
2710 PRINT''Be sure to backup your diskettes after each update.''
2720 PRINT''It is advisable to keep a Father, Son and Grandfather''
2730 PRINT''set and rotate the backups.''
2740 END:REM -----> Open employee file and write subroutine **
2750 OPEN ''O'',1,E$
2760 PRINT #1,E1$+'',',',SS$+'',',',
2770 FOR J=0 TO 18
2780   PRINT #1,E(J);
2790 NEXT J
2800 CLOSE 1
2810 RETURN
2820 END:REM -----> Open employee file and read subroutine **
2830 OPEN ''I'',1,E$
2840 INPUT #1,E1$,SS$

```

```

2850 FOR J=0 TO 18
2860   IF EOF(1) THEN 2890
2870   INPUT #1,E(J)
2880 NEXT J
2890 CLOSE 1
2900 RETURN
2910 END:REM -----> Write the PAYNAMES file to disk **
2920 OPEN ``O``,1,``PAYNAMES``
2930 N1=0
2940 FOR I=1 TO NE
2950   IF E$(I)="" THEN GOTO 2980
2960   PRINT #1, E$(I)
2970   N1=N1+1
2980 NEXT I
2990 CLOSE 1
3000 REM -----> Read the PAYNAMES file from disk **
3010 OPEN ``I``,1,``PAYNAMES``
3020 FOR I=1 TO N1
3030   IF EOF(1) THEN GOTO 3060
3040   INPUT#1,E$(I)
3050 NEXT I
3060 CLOSE 1
3070 RETURN
3080 REM -----> Who is Real? Subroutine **
3090 FOR I=1 TO NE
3100   IF E$(I)=E$ AND NOT (E$="") THEN RETURN
3110 NEXT I
3120 PRINT
3130 PRINT E$;" is NOT a valid pay name."
3140 PRINT"``These are:``"
3150 PRINT
3160 FOR I=1 TO N1
3170   PRINT E$(I);" ";
3180 NEXT I
3190 PRINT:PRINT
3200 INPUT"``Which one do you want`` ";E$
3210 GOTO 3090
3220 `
3230 ` error trap to create paynames file if it doesn't exist.
3240 IF ERR <> 53 THEN ON ERROR GOTO 0
3250 ` IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0 ` basic prior to ver
    5.0
3260 OPEN``O``,1, ``Paynames``
3270 PRINT #1,``ONE``
3280 CLOSE 1
3290 GOTO 350
3300 END ` of program

```

Handy Order Form

| | | |
|--|---------|--|
| RENEW Subscription Nov/Dec 88 through Sep/Oct 89 | \$24.95 | |
| All third year issues, Nov 87 through Sep 88 | \$24.95 | |
| All second year issues, Nov 86 through Sep 87 | \$24.95 | |
| All first year issues, Sep 85 through Sep 86 | \$24.95 | |
| 1st Year Program Disk (issues 1 through 7) (Specify computer type below) | \$20.00 | |
| 2nd Year Program Disk (issues 8 through 13) (Specify computer type below) | \$20.00 | |
| 3rd Year Program Disk (issues 14 through 19) (Specify computer type below) <i>Available now</i> | \$20.00 | |
| NEW! "Starting with MS DOS" 40-page book explains all | \$7.00 | |

Postage and handling charges already included in all prices.

We can supply program diskettes for PC/MS DOS, TRSDOS 1.3 (Tandy Model 3), TRSDOS 6.x (Tandy Model IV) and most CP/M MBASIC formats, on 5 1/4 inch diskettes. When ordering diskettes, specify your computer type.

COMPUTER TYPE _____

Check/MO enclosed

Charge to my VISA/MC _____ exp _____

Ship to: Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to:
CodeWorks, 3838 S. Warner St. Tacoma WA 98409

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

VISA/Master Card only

389

Index & Download

What's happening with both

Here are the updates to bring CWinindex.Dat up to date through the last issue. The entire index for the first three years of CodeWorks is on the download and on our yearly diskette.

Notes, shell, erase and rename, issue 21, page 3

Notes, using more than 64K of memory, issue 21, page 4

Notes, direct cursor positioning with CP/M, issue 21, page 4

Notes, error message patch for Tandy III, issue 21, page 5

Beginning BASIC, a look at variable types, issue 21, page 6

Fileutil.bas, main program, issue 21, page 7, a collection of file utilities

Frame.bas, main program, issue 21, page 14, building costs and materials

Trust.bas, main program, issue 21, page 32, computerized loan payment book

Notes, direct execution from screen in MS DOS, issue 21, page 38

Download, notes on the download, issue 21, page 40

CWinindex.dat, updates to this index, issue 21, page 40

There hasn't been too much excitement on the download lately. It's running smoothly, except for one power failure during early February, which didn't last more than an hour or so.

We have noticed that several of you are still having problems when signing on. Use your last name and your subscriber number, and you will get right in. Your subscriber number is the number at the upper right of your mailing label.

CodeWorks

3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
U.S. Postage
PAID
Permit 774
Tacoma, WA

2910
ERICKSON, MIKE / KRJB
BOX 250
MONTE RIO CA 95462

CODEWORKS

Issue 23

May/June 1989

CONTENTS

| | |
|-----------------------------------|-----------|
| Editor's Notes | 2 |
| Forum | 3 |
| Beginning BASIC | 5 |
| Matrix.Bas | 8 |
| Invoice.Bas | 17 |
| Notes | 26 |
| Outline.Bas | 27 |
| Order form..... | 39 |
| CWindex and Download | 40 |

Issue 23

May/June 1989

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

(c)1989 80-Northwest Publishing Inc. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, the publisher assumes no responsibility for errors or omissions. Unless otherwise noted, all programs presented in this publication are placed in public domain. Please address all correspondence to CodeWorks, 3838 South Warner Street, 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. 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, WA.

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.

How many computers can you afford to buy in one lifetime? If you read the mainstream computer magazines, it almost seems that they expect you to trade up just as fast as new CPU chips are released. It started with the 8086, then the 80186, the 80286, 80386 and now they're talking about an 80486 chip. Each new chip offers a new higher operating speed. They're talking now about clock rates as high as 25 Mhz; so fast, in fact, that you will probably have your answer on the screen before your finger leaves the run button.

But what about us "normal" folks who can't afford to follow along with the latest fad in computer hardware? Admittedly, there are those among us who are hardware junkies and have not one, but several, relics. The rest of us are still justifying the purchase of our first computer and wondering how and when to upgrade.

Electronics is such a fast moving industry it seems that today's marvel is marked down tomorrow and obsolete the day after. That's both good and bad. It really messes up your resale value if you want to sell. But on the other hand, you probably bought it at a bargain price, especially if you didn't run right out and buy it when it was first announced.

One good thing has emerged in the past few years, that being the compatibility and universal standard of MS DOS. Whatever bad things you can say about it, it does have a few good things going for it. As a case in point, I just finished writing an income tracking and payroll program for beauty salons.

This was not for publication; it was for a local computer store. The nice thing about the programs was that after I compiled them using Quick-BASIC, you could just pop them into any IBM PC, XT, AT or compatible and they run. That covers a whole lot of machines. Not only that, but they were written on an 8086 machine and compiled there, and they still run on 80X86 machines. That's nice.

But back at the question of if and when to upgrade: Personally, if I still had a 64-column screen machine, or a machine that required a kludge to put lower-case characters on the screen, I'd upgrade in a minute. You can buy some pretty high-powered machines these days for under \$1000, and in some cases that even includes a hard drive. Not only that, but it's a fairly safe bet that MS DOS and all the compatibles will be around for a good long time. There are just too many of them in use everyday, which makes a market for writers of software. In fact, there is so much software available for these machines that several of our readers have simply given up on programming and have written to tell us so. In a way, it's easy to see why, although we still believe that the only way to get what you really want is to program it yourself.

And it all depends on what you are doing with your computer. If you like to program your own and are doing it for your own enjoyment - then that old computer might still have a few good years left in it.

Irv

Forum

An Open Forum for Questions and Comments

Your magazine has been coming to me since the first edition. It has been quite enlightening. In the Forum for Issue 21 I read with considerable interest the item of John M. Gregg about the Mormon genealogical program. I subsequently wrote to Salt Lake City, inquiring as to whether the program was available on 8 inch SSSD disks in the IBM format for the CP/M operating system. Their very prompt reply stated that the PAF was not published on the 8 inch format.

I operate a Heath H-11A (DEC LSI-11) computer equipped with a coprocessor so that I can also run in the CP/M operating system. It requires 8 inch disks SSSD and in the IBM format.

Do you have information that will tell me whom to go to who may have already converted this program (PAF) to run under the system I have available?

Clay E. Lewis
1652 Garfield Avenue
Wyomissing, PA 19610

We can't, but perhaps some of our readers may be able to help.

How can I get or write a little BASIC program that can read my data files (which I can't read otherwise) from Genesis, a financial data service down in Texas? I have an IBM computer and am familiar a little with programming.

Walter Jung
6225 Brightlea Drive
Lanham, MD 20706

It depends entirely on the format those files are written in. Also, what form do you want the output to be in? Some commercial programs use very odd file structures. You might try our FileUtil.Bas program from Issue 21 to at least see what some of your file looks like. That may give you a clue as to how to proceed.

In Issue 21 Charles B. Steele was looking for a cross reference utility for use with BASIC on an MS DOS machine. I was in the same boat as Mr.

Steele when I switched from a CP/M machine to MS DOS. I have found use of a cross referencing utility most helpful in keeping track of variable names used in a BASIC program as well as in debugging the programs. For example it's easy to pick out a misspelled variable from the alphabetical listing because it will usually have only one line number associated with it.

Although I don't want to discourage you from publishing a cross reference program in CodeWorks (it would be interesting to follow the logic), Mr. Steele and other BASIC programmers might be interested in trying PC-XREF, version 6.1, by James T. Demberger. I have found it to be fast, comprehensive and easy to use. I also like the fact that it reads BASIC programs saved in tokenized format rather than ASCII because I usually save my programs that way. PC-XREF is a shareware program which may be downloaded from CompuServe and Genie on-line data services. A disk containing documentation and program files may be ordered directly from Mr. Demberger for \$5.00 plus \$1.00 shipping and handling. The registration fee is \$15.00 if one likes and wants to use the program. The address is PC-*. *Shareware, 9862 Lake Seminole Drive, West, Seminole, Florida 34643, phone (810) 397-2930.

I'm looking forward to another year of helpful and interesting articles in CodeWorks. I was sorry to read about Mr. Dettmann's disk problems. Please tell him we hope his reconstruction period is short.

Robert L. Anderson
St. Albans, WV

Re: Notes, Issue 22, How to scroll in GW BASIC. As stated in the magazine did not work. How come? Did it work for you? Sure would love a program that does scroll forward and backward.

I'm curious as to why you don't use the backspace to slash your zeros. Mistaking Oh's for

zeros was the worst experience when I first was learning BASIC when typing other programs and avoiding it is so easy with the slash. Are you not using BASIC to put your magazine together?

Fran Hynes
San Francisco, CA

Scrolling as per the note in Issue 22 worked on all the machines with GW BASIC we tried it on.

We are using BASIC as much as possible to put the magazine together. The programs are saved in ASCII and loaded into the desktop publishing program and set in Courier. Note the Oh's are fat and round, while the zeros are egg-shaped. We haven't yet found a way to slash the zeros using this method of printing the listings.

David Charlton's letter in Issue 22 got my attention as it relates to a problem I am trying to solve (actually I want someone else to solve it for me). The two major high resolution BASIC programs for TRS-80 computers (Models I,III,4) differ greatly in the speed of the circle or arc drawing routines. BASICG from Radio Shack is nearly 10 times faster than GBASIC from Micro Labs. Otherwise, the Micro Labs program is much better for my purposes. I dearly wish someone would "fix" GBASIC so it doesn't take forever to draw a curve.

With my talent for machine language programming, I could never get the job done in this century. I believe that I do know the reason the BASICG draws circles so much faster than GBASIC, however.

This brings me back to David Charlton's letter and his circle program, which uses the trig functions SIN and COS. The routines "borrowed" from BASIC ROM for trig functions are quite slow compared to the arithmetic functions.

The square root SQR function is somewhat better and can also be calculated rapidly in machine language using arithmetic functions if an estimate is available. But that's another story...

...The question is, who would like to make us a patch to speed up the GBASIC circle routine? I would help as long as I don't have to do the machine language hacking.

Not to overlook Charlton's problem with

round circles, it is due to pixels seldom being square with the result that Y or X has to be multiplied by the pixel aspect ratio. In TRS-80 Hi-Res the ratio is 2:1. There is a way to adjust picture tube horizontal height to get it just right. Other systems may have different ratios, and if you're really lucky, maybe 1:1.

Bob Keegan
112 W. Center St. # 615
Fayetteville, AR 72701

Perhaps someone with the computers you mention and the inclination to program in machine code will be able to give you a hand.

That's Forum for this issue. See you all again in July. - Irv



I discovered that between the time my computer records a stock sale and the main computer dispenses the info, I could make four million dollars.

Beginning BASIC

All About Strings - Part 1

What makes your computer different than a calculator? They both add, subtract, multiply, divide and do various other operations with numbers, don't they? True, but your computer has one other ability that most programmable calculators do not have - the ability to handle strings of text.

But your computer is full of nothing but binary numbers, so how can it tell the difference between a number and a letter? Well, we let certain of those numbers represent letters and punctuation and other symbols. If you look at the ASCII character set, you will find that the numbers from 1 through 31 are used for control symbols; from 31 to 122 they are used for numbers, both upper and lower case letters and punctuation; and from 123 through 255 they are used for special characters and graphics symbols. One eight-bit byte then, can represent any character, since there are 256 possible combinations of one byte.

If we tell the computer that `A=5`, then some memory location called `A` will contain the ASCII value for 5. When you then `PRINT A`, it will go to location `A` and fetch the value (5) stored there and print it on the screen. The actual translation from the binary 5 to the decimal 5 is made in the character generator of your computer, between memory and your video screen. Actually, the value stored in `A` will not be 5, but will be the ASCII value for 5, which is 53 or binary 00110101. The digits 0 through 9 are ASCII 48 through 57. Most operator manuals on BASIC which came with your computer have, as an appendix, a list of the ASCII codes.

Now you can have input a value of 5 and another of 8 and another of 10, and they could be in almost any location in memory. Assuming they were called `A`, `B` and `C`, you could always retrieve them by `PRINT A,B,C`. But when we

input a string, we want the entire string to be in one contiguous chunk, so that when we print it, it will look right. Obviously then, strings need something special to tell the computer that they are, indeed, strings and not simple integer numbers. The device used to denote a string is the dollar sign (\$), after the variable designation, as in `A$`. In addition to that, when you define a string inside a BASIC program, you must enclose it in quotes, as in: `A$="This is a string."`

Now, instead of going into simple variable space, `A$` will go into what is called "string space" and will be treated somewhat differently than other variables. Can you now see where the designation "string" came from? It is probably due to the fact that the string occupies a "string" of spaces in memory, dependent upon the length of the string. Each character of the string, of course, would occupy one byte.

Let's go back for a moment and look at the exceptions to inputting a string. If your program has the statement: `INPUT "What is your name";A$` you can answer without using quotes. If the statement were, however, `INPUT "What is your last name, first name";A$`, you would need to put quotes around your answer because it contains a comma. The same would hold true if the answer contained a semi-colon. `DATA` statements can be read in as strings and need not be enclosed in quotes. In other words, this would be possible:

```
DATA one,two,three
```

But - and there always seems to be a but - if any of your data elements contained a comma or semi-colon, or if there were any leading or lagging spaces, you would have to enclose the entire element in quotes. An example:

```
DATA "Jones, John", "items; 12 ", "three, four
```

Again, back to our input example above, a variation of INPUT, called LINE INPUT, will allow input of any punctuation. As in: LINE INPUT "Last, first name ";A\$. These little differences can be rather confusing to those just starting out in BASIC. Also notice that the portion of the INPUT statement enclosed in quotes is itself a string. These "literal" strings are built right into the program and do not reside in the "string space" set aside for definable string variables. But they are strings, nonetheless.

The letter A, stored in some memory location and then called back, is easily recognizable as a string simply because it is a letter. But what about the number 3? Digits can be stored as either string or integer, and sometimes it's hard to tell the difference. And it does make a difference: You simply cannot add a string 3 with an integer 5 and get 8. We'll find out much more about this a little later on.

When a string variable is stored in memory other pertinent information is appended internally to it. Well, like most things in computing, that's not quite true, but almost. Actually, a variable pointer block is set up to point to the string's location and it contains the length of the string as well as the memory location of the actual string. Since strings do not have a fixed length, it is important to know just how long a string is. The length of a string includes all spaces and punctuation, in fact, everything between the quotes is included in the length of the string. A zero-length string is called a "null string" as in A\$="".

BASIC includes the function LEN(string), which will return the length of the string specified. You could say: L=LEN(A\$), and variable L would contain the length of A\$. We don't always care how long a string is, but we certainly can use something that defines that length. Confusing? Well, if we want to look at each character in a string with a loop, we can use this:

```
A$="This is a string"  
FOR I = 1 TO LEN(A$)
```

do something, then NEXT

in which case we really didn't care about the length of the string as long as the program knew it and looked at every character in the string. LEN(string) is a **function** and returns an integer value representing the length of the string.

A small aside here, while we look at the difference between functions and statements. A statement is an instruction to the computer telling it to do something, like GOTO or PRINT. Functions are like little subroutines that calculate something, like SQR(N), or LEN(string), or VAL(string). There are functions that operate on integers and those that operate on strings. A function has two parts, a title and an argument. The title describes what the function does, like VAL(A\$) (value of A\$) or LEN(A\$) (length of A\$). The argument is the input to the function (A\$ in both of the above cases). The function takes the input (A\$) and returns a result.

If A\$="this is a string" and we PRINT LEN(A\$) the answer we will get is 16. Now that we know how to get the length of a string, let's go on to another interesting statement used with strings: INSTR (read: INSTRing). The whole form of this statement is:

```
INSTR(position,string,sub-string)
```

INSTR searches *string*, beginning at *position*, looking for *sub-string*. It returns the position number in the string where sub-string was found. It returns zero if the sub-string was not found. If position is omitted, the search begins at the first character of string, position 1. INSTR is case-sensitive, which means that you can't find "A" by looking for "a". Here is an example:

```
A$="this is a string"  
P=INSTR(1,A$,"a")  
PRINT P
```

If you run this bit of code, variable P will contain a 9 because the lower-case "a" occupies the ninth position in A\$. Later, we will see how valuable INSTR is when trying to find an exact

match within a string. Before we leave this part of the discussion, let's add that the maximum length a string in BASIC can have is 255 characters.

Earlier we mentioned that each string has a variable control block to tell it how long the string is and where it is located in memory. BASIC has an interesting function called `VARPTR` (stands for "Variable Pointer"). Assuming that we still have the same `A$` from the previous paragraphs, if you `PRINT VARPTR(A$)`, you will get a memory address. If you then `PEEK` that address, you will get the length of the string. In our case (and it will vary, depending on your computer and BASIC) when we took the `VARPTR(A$)`, we got 4452, the decimal address of the variable control block. Actually, the control block is three bytes long, and 4452 is the address of the first of those three bytes. When we did a `PEEK(4452)` it returned a 16. How about that? It turns out that our `A$` is exactly 16 characters long. So the first byte of the three tells how long the string is.

The next two bytes of the variable pointer (addresses 4453 and 4454) contain the memory address of the start of the actual string. In our case, again, the actual `A$` started at memory location 4061. `PEEK`ing into that address and the 15 following addresses we got a series of ASCII numbers. When we took the `CHR$` of each of the numbers, guess what we got? They translated into "this is a string"! But why two bytes for the address of the start of the string? Well, the first byte is the low-order byte and the second byte is the high-order byte of the address. That way, two bytes can address up to 65536 memory locations. Each byte has 256 discreet states and 256 times 256 just happens to be 65536. And, by the way, the so-called 64K memory is really 65.536K. See Figure 1 for a little program that you can type in and try with various different strings and lengths. Keep in mind that you won't necessarily get the same numbers for memory locations that we did. `VARPTR` is not the most used function in BASIC. It's handy when you need it, though.

Now comes a real interesting question. If the string, itself, is out there all by itself, and the variable pointer has only three bytes telling length and where the string actually is, what happens when you have an `A$` and a `B$`? How does the computer know which variable pointer block to go to? We scoured all the available books we have on the subject, and the best we can come up with at this point is that BASIC also keeps a variable table. That table, then, must point to the variable pointer, which points to the string. It seems that we run and hide from ourselves just to get excited when we find ourselves again!

In the next issue we will continue with more practical string-handling commands and functions. We will also try some bits of code that demonstrate how the various commands work. String handling is, after all, almost half of computing!

Figure 1

```
10 A$="this is a string"
20 P=VARPTR(A$)
30 PRINT"The variable control block is at address ";P
40 PRINT"That address contains the length ";PEEK(P)
45 PRINT"The next two bytes contain the address of the string."
50 X=PEEK(P+1)+256*PEEK(P+2)
60 PRINT"The starting address of the actual string ";X
70 PRINT
80 PRINT"The contents of address ";X;" to ";X+PEEK(P);" are:"
90 FOR I=X TO X+PEEK(P)
100 PRINT PEEK(I)="CHR$(PEEK(I));"
110 NEXT I
```

Matrix.Bas

A Complete Set of Matrix Functions

Staff Project. Solving for 20 unknowns with 20 equations is what's possible with this program. In addition, it provides all the other important matrix manipulations you may have need of from time to time.

In science, mathematics, business and many other fields that use number systems it becomes necessary to be able to solve systems of linear equations. One system might take on the form:

$$3x + 4y = 10$$

$$7x - 5y = 22$$

In this example, there are two equations and two unknowns. (Unknowns refer to the variables x and y .) These kinds of equations are relatively simple to solve and there are many methods with which to solve them.

But what if there is a need to solve 18 equations and 18 unknowns? Do you really want to try that with pencil and paper?

In trying to maximize profits, or determine the total pressure needed to inflate 14 different balloons of 14 different sizes, or how much voltage is needed in order for 20 circuits to carry 20 different currents - the list goes on forever - systems of linear equations must be solved. Matrix.Bas is a program that will do the work for you.

In order to solve these systems of equations it becomes necessary to work with matrices. And it becomes necessary to multiply matrices, mainly to find the inverse of an $m \times n$ matrix. (m rows and n columns are usually designated by $m \times n$.) Some of the earlier BASICs of about 10 years ago included the MAT functions which would allow you to perform these functions easily. Somehow,

with the evolution of BASIC and newer computers these MAT functions dropped by the way-side. Matrix.Bas is a program that gives you back these functions.

The program will do the following:

1. Add two matrices.
2. Subtract two matrices.
3. Transpose a matrix.
4. Multiply a matrix by a constant.
5. Multiply two matrices.
6. Find the inverse of a matrix and/or solve for a solution to n unknowns with n equations.

The first four items are straightforward and probably will seldom be used except for large matrices. The key to solving equations is in the last two items. If you have never before worked with matrices, take a look at any math book and follow the steps required to multiply two matrices. It will help you appreciate this program that much more.

The program will first display the menu, giving the above commands. Enter the number of your choice and the computer will come back asking you the order (number of rows and columns) of your matrix. Some matrix algebra requires square matrices (as many rows as columns) but others can have a different number of rows and columns.

Then it will allow you to see the matrix that is just completed, giving you a chance to make sure that is was entered correctly. If you have a printer you can send output to it as well as the screen.

If you are solving for inverses the input works the same way, but on output you will notice that all of the matrices are double precision. This is to allow for round-off errors inside the computer on the calculations needed for finding inverses and using double precision variables will take care of most of this. They are, by the way, returned to normal precision as soon as the inverse has been computed and printed.

Once the computer has solved for the inverse, it will prompt you with a question asking if you want to use that answer positively and enter the solution matrix when requested to. (Solution matrices are on the order of $1 \times n$). It will then calculate all of the variables in the solution system and come back with the answer. Again,

you can send this output to the printer, too.

Here is an example of solving three unknowns in three equations using option 6 of the menu. You need to tell the program that the matrix is 3×3 . Make sure that all your unknowns are lined up properly. Then enter 2, -9, -5, 7, -6, 5, 9, -6 and 5. When asked for the solution matrix enter: 2, -35 and -39. The program will then tell you that $x = -2$, $y = 1$ and $z = -3$, which is the answer you wanted.

$$\begin{aligned}2x - 9y - 5z &= 2 \\7x - 6y + 5z &= -35 \\9x - 6y + 5z &= -39\end{aligned}$$

The program is currently set for matrices that are 20×20 , which will allow you to solve for 20 unknowns in 20 equations. If you have the memory space, you can adjust the DIM statement to use larger matrices. Wouldn't this have been a nice program to have back when we were taking Algebra 1?

Matrix.Bas for MS DOS and Tandy Models II and IV

```
100 REM * Matrix.Bas * Manipulation of Matrices
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 DIM A(20,20),B(20,20),T(20,20),I(20)
170 ON ERROR GOTO 3030
180 CLS
190 PRINT STRING$(22,45);'' The CodeWorks ``;STRING$(23,45)
200 PRINT''           M A T R I X   P R O G R A M
210 PRINT''           provides a full set of matrix functions
220 PRINT STRING$(60,45)
230 PRINT
240 PRINT TAB(5);'' 1 - Add Matrices A and B
```

```

250 PRINT TAB(5);'' 2 - Subtract Matrices A and B
260 PRINT TAB(5);'' 3 - Transpose Matrix A
270 PRINT TAB(5);'' 4 - Multiply Matrix A by a constant
280 PRINT TAB(5);'' 5 - Multiply Matrices A and B
290 PRINT TAB(5);'' 6 - Compute the inverse of Matrix A and/or
300 PRINT TAB(5);''      determine the solution to system equations
310 PRINT TAB(5);'' 7 - Quit
320 PRINT
330 PRINT''Your choice'';
340 INPUT CH
350 ON CH GOSUB 1030,1350,1810,2050,2540,390,3080
360 GOTO 180
370 `
380 REM * Inverse of a Matrix
390 CLS
400 PRINT TAB(10);''Inverse of a Matrix''
410 PRINT
420 INPUT''Enter the number of rows and columns in Matrix A'';N
430 DEFDBL A,Q,T,X
440 C=N:R=N
450 A$=''A''
460 GOSUB 2230
470 FOR I=1 TO N
480   FOR J=1 TO N
490     A(I,J)=T(I,J)
500   NEXT J
510 NEXT I
520 FOR C=1 TO N
530   FOR I=1 TO N-1
540     Q(I)=A(1,I+1)/A(1,1)
550   NEXT I
560   Q(N)=1.00000001#/A(1,1)
570   FOR I=1 TO N-1
580     FOR J=1 TO N-1
590       A(I,J)=A(I+1,J+1)-A(I+1,1)*Q(J)
600     NEXT J
610     A(I,N)=- (A(I+1,1))*Q(N)
620   NEXT I
630   FOR J=1 TO N
640     A(I,J)=Q(J)
650   NEXT J
660 NEXT C
670 PRINT''The inverse has been computed.''

```

```

680 FOR I=1 TO N
690   FOR J=1 TO N
700     T(I,J)=A(I,J)
710   NEXT J
720 NEXT I
730 PRINT''The results are ready to be printed.''
740 C=N:R=N
750 GOSUB 2350
760 PRINT:PRINT''The above is the inverted Matrix A. Would you like
to:
770 PRINT TAB(10);''1 - Return to the menu
780 PRINT TAB(10);''2 - Enter a solution Matrix to solve the
system.
790 PRINT
800 INPUT''Your choice '';AN$
810 IF LEFT$(AN$,1)=''1" THEN RETURN ELSE IF LEFT$(AN$,1)<>'2" THEN
760
820 PRINT''Input the solution matrix
830 X=1E-08
840 FOR I=1 TO N
850   PRINT''Solution #'';I;;INPUT Q(I)
860 NEXT I
870 FOR I=1 TO N
880   FOR J=1 TO N
890     X=X+A(I,J)*Q(J)
900   NEXT J
910   Q1(I)=X
920   X=1E-08
930 NEXT I
940 PRINT:PRINT''This is the solution set:''
950 FOR I=1 TO N
960   PRINT Q1(I)
970 NEXT I
980 DEFSNG A,T,Q,X
990 INPUT''Press enter to continue'';A$
1000 RETURN
1010 `
1020 REM * Add Matrix A and B
1030 CLS
1040 PRINT TAB(10);''Add two matrices.''
1050 PRINT
1060 PRINT''Enter them as follows:''
1070 PRINT''How many rows in your matrices''

```

```

1080 INPUT'' (A and B must be the same order)'';R
1090 INPUT''How many columns in A and B'';C
1100 A$=''A''
1110 GOSUB 2230
1120 FOR I=1 TO R
1130   FOR J=1 TO C
1140     A(I,J)=T(I,J)
1150   NEXT J
1160 NEXT I
1170 A$=''B''
1180 GOSUB 2230
1190 FOR I=1 TO R
1200   FOR J=1 TO C
1210     B(I,J)=T(I,J)
1220   NEXT J
1230 NEXT I
1240 FOR I=1 TO R
1250   FOR J=1 TO C
1260     T(I,J)=A(I,J)+B(I,J)
1270   NEXT J
1280 NEXT I
1290 PRINT''A + B has been computed and is ready to be printed''
1300 GOSUB 2350
1310 INPUT''Press Enter to continue'';A$
1320 RETURN
1330 `
1340 REM * Subtract Matrix A and B
1350 CLS
1360 PRINT TAB(10);''Subtract Matrix A and B
1370 PRINT
1380 PRINT''Enter them as follows:''
1390 PRINT''How many rows in your Matrices''
1400 INPUT'' (A and B must be of the same order) ``;R
1410 PRINT
1420 INPUT''How many columns in A and B'';C
1430 A$=''A''
1440 GOSUB 2230
1450 FOR I=1 TO R
1460   FOR J=1 TO C
1470     A(I,J)=T(I,J)
1480   NEXT J
1490 NEXT I

```

```

1500 A$=' 'B' '
1510 GOSUB 2230
1520 FOR I=1 TO R
1530   FOR J=1 TO C
1540     B(I,J)=T(I,J)
1550   NEXT J
1560 NEXT I
1570 PRINT''Would you like to:
1580 PRINT TAB(10);''1 - Subtract A from B, or
1590 PRINT TAB(10);''2 - Subtract B from A''
1600 PRINT
1610 INPUT''Enter your choice ``;AN$
1620 IF AN$=' '2" THEN 1700
1630 IF AN$<>' '1" THEN PRINT:GOTO 1600
1640 FOR I=1 TO R
1650   FOR J=1 TO C
1660     T(I,J)=A(I,J)-B(I,J)
1670   NEXT J
1680 NEXT I
1690 GOTO 1750
1700 FOR I=1 TO R
1710   FOR J=1 TO C
1720     T(I,J)=B(I,J)-A(I,J)
1730   NEXT J
1740 NEXT I
1750 PRINT''The results are ready to be printed.''
1760 GOSUB 2350
1770 INPUT''Press Enter to continue ``;A$
1780 RETURN
1790 `
1800 REM * Transpose
1810 CLS
1820 PRINT TAB(10);''The Transpose of Matrix A''
1830 A$=' 'A' '
1840 PRINT
1850 INPUT''Enter the number of rows in Matrix A'';R
1860 INPUT''Now enter the number of columns ``;C
1870 GOSUB 2230
1880 FOR I=1 TO R
1890   FOR J=1 TO C
1900     A(I,J)=T(I,J)
1910   NEXT J
1920 NEXT I

```

```

1930 FOR I=1 TO C
1940   FOR J=1 TO R
1950     T(I,J)=A(J,I)
1960   NEXT J
1970 NEXT I
1980 PRINT''Transpose is done. Results are ready to be printed.'''
1990 TE=C:C=R:R=TE
2000 GOSUB 2350
2010 INPUT''Press Enter to continue '';A$
2020 RETURN
2030 `
2040 REM * Multiply by a constant
2050 CLS
2060 PRINT TAB(10);''Multiply by a constant.'''
2070 PRINT
2080 A$=''A''
2090 INPUT''Enter the number of rows in Matrix A '';R
2100 INPUT''Now enter the number of columns '';C
2110 GOSUB 2230
2120 INPUT''What is the multiplicative constant '';CN
2130 FOR I=1 TO R
2140   FOR J=1 TO C
2150     T(I,J)=CN*T(I,J)
2160   NEXT J
2170 NEXT I
2180 PRINT''Multiplication is done. The results are ready to be
      printed.'''
2190 GOSUB 2350
2200 INPUT''Press Enter to continue '';A$
2210 RETURN
2220 `
2230 REM * Input subroutine *
2240 PRINT:PRINT''Start entering the Matrix '';A$;''':''
2250 FOR I=1 TO R
2260   FOR J=1 TO C
2270     PRINT I;'' , '' ;J;
2280     INPUT T(I,J)
2290   NEXT J
2300 NEXT I
2310 INPUT''Would you like to see the matrix you just entered '';AN$
2320 IF LEFT$(AN$,1)<>'Y' AND LEFT$(AN$,1)<>'y' THEN RETURN
2330 `
2340 REM * Output routine

```

Invoice.Bas

```
2350 INPUT''Would you like the output to go to the printer '';AN$
2360 IF LEFT$(AN$,1)=''Y'' OR LEFT$(AN$,1)=''y'' THEN P=1 ELSE P=0
2370 IF P=1 THEN 2450
2380 FOR I=1 TO R
2390   FOR J=1 TO C
2400     PRINT T(I,J);''  '';
2410   NEXT J
2420   PRINT
2430 NEXT I
2440 GOTO 2510
2450 FOR I=1 TO R
2460   FOR J=1 TO C
2470     LPRINT T(I,J);''  '';
2480   NEXT J
2490   LPRINT''  ''
2500 NEXT I
2510 RETURN
2520 `
2530 REM * Multiplication of two Matrices
2540 CLS
2550 PRINT TAB(10);''Multiplication of two Matrices A and B
2560 PRINT
2570 INPUT''Enter the number of rows in Matrix A'';R
2580 INPUT''Now enter the number of columns in Matrix A '';C
2590 A$=''A''
2600 GOSUB 2230
2610 FOR I=1 TO R
2620   FOR J=1 TO C
2630     A(I,J)=T(I,J)
2640   NEXT J
2650 NEXT I
2660 INPUT''Enter the number of rows in Matrix B'';RB
2670 INPUT''Now enter the number of columns in Matrix B'';CB
2680 IF C=RB THEN 2790
2690 PRINT''** ERROR ** the number of columns in Matrix A must
    equal
2700 PRINT''the number of rows in Matrix B before they can be
    multiplied.
2710 PRINT''Would you like to:
2720 PRINT TAB(10);''1 - End the function and return to the menu,
    or
2730 PRINT TAB(10);''2 - Enter row and column lengths for Matrix B
2740 PRINT
```

```

2750 PRINT''Enter your choice of 1 or 2";
2760 INPUT AN$
2770 IF AN$='1' THEN RETURN ELSE 2660
2780 GOTO 2660
2790 TR=R:TC=C:C=CB:R=RB
2800 A$='B'
2810 GOSUB 2230
2820 FOR I=1 TO R
2830   FOR J=1 TO C
2840     B(I,J)=T(I,J)
2850   NEXT J
2860 NEXT I
2870 R=TR:C=TC
2880 FOR I=1 TO R
2890   FOR J=1 TO CB
2900     T(I,J)=0
2910     FOR K=1 TO C
2920       T(I,J)=T(I,J)+A(I,K)*B(K,J)
2930     NEXT K
2940   NEXT J
2950 NEXT I
2960 C=CB
2970 PRINT''A times B has been computed and is ready to be printed.''

2980 GOSUB 2350
2990 INPUT''Press Enter to continue '';A$
3000 RETURN
3010 `
3020 REM * error trap for division by zero
3030 IF ERR<>11 THEN ON ERROR GOTO 0
3040 `IF (ERR/2)+1<>12 THEN ON ERROR GOTO 0 `Basic prior to ver
5.0
3050 PRINT''Division by zero. No solution possible.''
3060 INPUT''Press Enter to restart'';A$
3070 RUN 160
3080 END `of program

```

Changes for Tandy Models I and III

```

Changed->100 REM * Matrix/Bas * Manipulation of Matrices
Changed->140 CLEAR 2000
Changed->3030 `IF ERR<>11 THEN ON ERROR GOTO 0
Changed->3040 IF (ERR/2)+1<>12 THEN ON ERROR GOTO 0 `Basic prior to ver
5.0

```


Invoice.Bas

An Invoice Writer Program

Staff Project. This program was designed to fill out a pre-printed invoice form. Although yours may not be exactly like it, the ideas behind it are universal and you should be able to adjust the print statements to fit your exact needs.

We wrote Invoice.Bas in response to a request from a local print shop. It was written on an MS DOS machine and then transferred via RS-232 to a Tandy Model II at the print shop.

Some of the design requirements for the program were that it should be able to accommodate up to 250 customers; that the customers should always be sorted in alphabetical order by name; and that provisions should be incorporated to keep the customer phone number and tax identification (resale tax ID) number. The program was to be designed to print invoices on a pre-printed, multi-part form. The program was to charge sales tax only when a tax identification number was not present. A resettable invoice numbering system was to be included. The invoicing portion also needed the capability to have more than one item invoiced, discount allowed and a flat rate cost and cost per thousand figured in to the total.

One-half day was spent organizing the requirements and planning the attack. One full day was spent in coding and another day was entirely spent in fine tuning and debugging. Because the program had a real-world application, it was fun to write and debug. Especially challenging was the need to get the entire "wish list" into the program and make it all work together properly.

How the program works

We start with our customary opening lines and the print@/locate subroutine. The initialization section, starting at line 240, opens with an error trap statement for "file not found" errors. This would happen the very first time the program is run and no customer file exists. The ON ERROR GOTO 1900 in line 240 will take us to line 1900, where we check for error 53 (file not found) and if that is indeed the error, we gosub to 1280. At line 1280 we open the file for output and write blanks to the file and then return to 1930, where we resume program flow at line 330. At line 330 we would then read in the just created file and continue on our way.

The next statement in the initialization section is used to set the printer width to 132 characters (for the tab listing of the customer file). This is a GW-BASIC and CP/M MBASIC specific command and can be remarked for other computers. Line 260 dimensions our data file at 250 records, each seven fields deep. The next three lines are print USING format lines for dollar amounts. Line 300 is where you would insert your particular sales tax rate.

The customer file is read in the next section of code, starting at line 340. The first item in the file is always the last invoice number, and it gets

updated with each invoice written and is saved along with the file when you are done with the session. Since we are starting our file with I equal to one, we used the zero element of the array to hold the invoice number. In line 440, we find out how many items are in the file and store it in variable N1. We'll need to know that several times later in the program.

Customers are retrieved from this file by customer number. Each customer was assigned a number when he was initially entered into the file. So that the user will always know what the next customer number is, we print it right on the menu (see line 580), and in the section of code starting at line 470, we find out what the highest number is. We set variable AC equal to zero first, then make one quick pass through the file and if any customer number is larger than AC we set AC to that new higher number. When we have gone through the file one time, AC will contain the highest customer number.

The next section of code, from line 520 to 700, is the main menu and menu selection area. Note that the menu tells the user to always exit the program through menu option 7. This insures that the updated file will always be saved. You can see how that happens in the next section of code, from line 730 to 750. That's the "end session routine" and it does a GOSUB to save the file first. It then prints the amount of the total billing (if there was any) before it closes the file and ends. Note that if no billing took place during the session (maybe you just entered a new customer or something like that), then TA will equal zero and the second part of line 740 will be skipped. Variable TA, of course, gets updated in the section of code where we actually do the invoicing, which comes later.

The next section of code, from line 780 to 800, is where we can reset the invoice number. This allows the user to start invoice numbers with the current month, as in 880901, 880902, and so on for September 1988, and then 881001, 881002 and so on for October of 1988. Or, you can just start with number 1 at the beginning of the year

and let them accumulate. The invoice number gets updated in the invoice writing section later. Note that the number is in string form, so we will have to convert it to an integer using the VAL function and then add one to it and change it back to a string with the STR function.

The "Add new customer names" section comes next. In this section we put the prompts for the entire record on the screen first and then "fill in the blanks" using direct cursor positioning and our print@/locate subroutine. In line 850 we set the cursor to row 4 and start at the left hand side of the screen. Then we print the prompts for the field headings on the screen, all seven of them. Then in line 940 we set the column position to 22, which will position our cursor just the right of the field prompts, and in lines 950 through 970 we input the information for each field (and yes, you can just enter if there is nothing to put in a particular field.) Notice that we are both printing the input information on the screen and putting it into the A\$ array at the same time. For that reason, we put it into A\$(N1,X-3) in line 960 because X started counting on 4. That will put our first field in A\$(N1,1), our second in A\$(N1,2) and so on. Note that X is being used for two purposes here, one to position the cursor and the other to increment our array count. N1, of course, will get incremented to the next record in line 860, when we come to the next record.

Now, since we are in complete control of our cursor, after entering one record we get the "More Y/N" question and if there are more to enter we must clear out the information just entered. Lines 1010 through 1040 clear away the information just entered on the screen (don't worry, it's already safely tucked away in the A\$ array.) Then, just to be neat about things, in line 1050 we go up to the screen and clear away that More? prompt. If we are going to enter more names, then line 1060 will send us back to do that, otherwise, line 1070 will immediately send us to the sort and save routine with a GOSUB. When we return from the GOSUB, we go to line 330 to read the file in again. Why? Well, for one thing we will then have a new highest customer

number and the most current number of records in the file.

The easiest way to keep a file sorted is to keep it sorted. How about that? Sounds ridiculous but it's true. It is very easy and fast to sort one or two new names into an already sorted file. In fact, you can then use the cheapest sort routine available - the bubble sort. Of course, if you sit down and enter 100 customers the first time you use this program, it will take a little while to sort. But after that it will be a snap to keep up.

The sort routine is at line 1120 and ends at 1240. As just mentioned, it's a bubble sort. Just so that you have something to watch while the sort is in progress, line 1160 prints a string of periods on the screen, one for each record that is out of place. Line 1240 is a "hook" from the edit section of the program. If we edited the name, and only when we edit the name, we come here to do a resort and then return.

Unless we were in the edit mode and edited a name, each time we sort the file we immediately save it. There are two reasons for that: one is so that the most current version of the file is always safely on the disk, the other is to eliminate deleted records. The code from line 1270 to 1370 does it all. Note in line 1320 that when we have deleted a customer their name is removed from the name field in the record. When we save the file, if that field is blank, we don't even bother to save the remainder of the record, but just skip on to the next record. Note also that we always print the most current invoice number to the file first (in line 1290).

A reference (or tab) list of all your customers can be printed on 14 inch wide paper with the code starting at line 1400. This section of code prints a heading on the paper and then puts 50 customers per page. There's room for more per page, and you can change that in line 1490 if you like. It also prints the page number on each page. The page variable is PG and appears in lines 1420, 1430 and in 1490. The last page of the report is page ejected by the LPRINT CHR\$(12)

in line 1500. If your computer has default printer settings you may have to change them to accommodate this report generator. Since we are handling all the paging information in this section of code, you would have to set your defaults so that the number of lines to print is equal to the number of lines on the page (usually 66).

The "Edit and Delete" section comes next. This section of code finds a customer by customer number and prints the entire record on the screen, along with a line number for each field of the record. You can then enter a new line to replace the old one or delete the customer by answering the prompts. Note that when you delete a customer we just set the name field to a null string and go on. The actual delete will be taken care of in the save routine, discussed earlier. The prompts in this section have purposely been set so that you must answer "Y" or "y" if you want something to happen. Any other key (including ENTER) will take you on to the next question or prompt. In line 1640 we ask what line to edit. If we select line 2 (the name field) then line 1730 will send us to the sort subroutine to immediately resort the name into the correct order.

The invoices this program writes are designed to fit into a window envelope, so labels for selected customers are not necessary. The user wanted to be able to print labels for the entire list, however, so that notices of specials and things like that could be sent. The section of code from line 1770 to 1880 prints standard one-up labels for that purpose.

The actual invoice writing portion of this program was put at the end so that you could easily modify it for your own purposes. It is highly unlikely that you would use the exact format for yours. Basically, it must ask for the customer number and then print the applicable information on the invoice. It must then ask questions about the services performed and keep totals of dollar amounts. Note that the total billing for the session is kept track of in line 2310, by variable TA (which we discussed earlier.) Also, the in-

voice number is updated in line 2320, also discussed earlier. If you don't have pre-printed forms to fill you can even use this section to print your company name and headings, especially if you have one of those printers that prints big type and condensed and bold. You can essentially design your own form for this section.

This program reminded us to a large extent of Card.Bas. It has a lot of the same characteristics, and hinted that Card.Bas could be updated to include some of the features of this program, especially using double subscripted arrays for the records and fields. But that's a whole 'nuther project.

**Invoice.Bas for MS DOS
changes for Tandy machines
follow this listing**

```
100 REM * Invoice.Bas * Invoice program 30 Aug 88 IMS
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$(31+Y);:RETURN ` CP/M
220 `
230 ` Initialization
240 ON ERROR GOTO 1900
250 WIDTH LPRINT 132
260 DIM A$(250,7)
270 F1$=' '$$##.##'
280 F2$=' '$$#,##.##'
290 F3$=' '$$#,##.##-'
300 F3=.078 ` sales tax rate goes here.....
310 `
320 ` read in the data file
330 CLS
340 PRINT''Loading the customer file...''
350 OPEN `I',1,'invoice.dat''
```

```

360 INPUT #1,A$(0,0)
370 FOR I=1 TO 251
380 IF EOF(1) THEN 430
390 FOR J=1 TO 7
400 LINE INPUT #1, A$(I,J)
410 NEXT J
420 NEXT I
430 CLOSE 1
440 N1=I-1
450 `
460 `find highest customer number
470 AC=0
480 FOR I=1 TO N1
490 IF VAL(A$(I,1))>AC THEN AC=VAL(A$(I,1))
500 NEXT I
510 `
520 ` main menu and heading
530 CLS
540 PRINT STRING$(22,45);'' The CodeWorks ``;STRING$(23,45)
550 PRINT'' INVOICE WRITING PROGRAM
560 PRINT'' Always exit this program through option 7
570 PRINT STRING$(60,45)
580 PRINT'' The highest customer number used is ``;AC
590 PRINT
600 PRINT TAB(20);''1 - Process Invoices
610 PRINT TAB(20);''2 - Print Customer List
620 PRINT TAB(20);''3 - Add new customer names
630 PRINT TAB(20);''4 - Edit customer data
640 PRINT TAB(20);''5 - Initialize invoice number
650 PRINT TAB(20);''6 - Print Mailing Labels
660 PRINT TAB(20);''7 - END session
670 PRINT
680 PRINT TAB(10);:INPUT''Number of your choice ``;XX
690 ON XX GOTO 1960,1400,830,1540,780,1770,730
700 GOTO 680
710 `
720 ` end session routine
730 GOSUB 1270 ` to save the file
740 CLS:IF TA<>0 THEN PRINT''Total billing for this session ``;USING
F2$;TA
750 CLOSE:END
760 `
770 ` initialize invoice number routine

```

```

780 CLS
790 INPUT''Start invoice numbers at '';A$(0,0)
800 GOTO 530
810 `
820 `add new customer names
830 CLS
840 PRINT TAB(20);'' Add new customer names''
850 X=4:Y=1:GOSUB 170
860 N1=N1+1
870 PRINT''      Customer number:
880 PRINT''      Customer name:
890 PRINT''      Address:
900 PRINT''      City,State:
910 PRINT''      Zip code:
920 PRINT''      Phone number:
930 PRINT''      Tax number:
940 Y=22
950 FOR X=4 TO 10
960   GOSUB 170:LINE INPUT A$(N1,X-3)
970 NEXT X
980 PRINT
990 INPUT''More (y/n)'';XX$
1000 IF XX$=''n'' OR XX$=''N'' THEN 1070
1010 Y=22
1020 FOR X=4 TO 10
1030   GOSUB 170:PRINT STRING$(30,32)
1040 NEXT X
1050 X=12:Y=1:GOSUB 170:PRINT STRING$(20,32)
1060 GOTO 850
1070 GOSUB 1120 ` to sort and save
1080 GOTO 330
1090 `
1100 `the sort and save subroutine
1110 ` first we sort ...
1120 PRINT''Sorting'';
1130 FOR I=1 TO N1-1
1140   L=I+1
1150   IF A$(I,2)=<A$(L,2) THEN 1210
1160   PRINT ``.'';
1170   FOR J=1 TO 7
1180     SWAP A$(I,J),A$(L,J)
1190   NEXT J
1200   F=1

```

```

1210 NEXT I
1220 IF F=1 THEN F=0:GOTO 1130
1230 PRINT
1240 IF CC=2 THEN RETURN
1250 `
1260 ` then we save the file
1270 PRINT''Saving the file ....
1280 OPEN ``O'',1,``invoice.dat''
1290 PRINT #1, A$(0,0)
1300 FOR I=1 TO N1
1310   FOR J=1 TO 7
1320     IF A$(I,2)='' THEN 1350
1330     PRINT #1, A$(I,J)
1340   NEXT J
1350 NEXT I
1360 CLOSE 1
1370 RETURN
1380 `
1390 ` print the customer list
1400 CLS:PRINT TAB(15);'' Printing Customer List
1410 INPUT''Line up your paper and press enter ``;XX
1420 PG=1:CL=0:I=1
1430 LPRINT''The CodeWorks Customer List'';TAB(60);''Page ``;PG;
TAB(100);DATE$
1440 LPRINT STRING$(128,45)
1450 LPRINT''Cust#'';TAB(8);''Customer Name'';TAB(38);''Address'';
TAB(68);''City/State'';TAB(88);''Zip Code'';TAB(100);''Phone #'';
TAB(115);''Resale #''
1460 LPRINT STRING$(128,45)
1470 LPRINT A$(I,1);TAB(8);A$(I,2);TAB(38);A$(I,3);TAB(68);A$(I,4);
TAB(88);A$(I,5);TAB(100);A$(I,6);TAB(115);A$(I,7)
1480 I=I+1:IF I>N1 THEN 1500
1490 CL=CL+1:IF CL>50 THEN PG=PG+1:CL=0:LPRINT CHR$(12):GOTO 1430
ELSE 1470
1500 LPRINT CHR$(12)
1510 GOTO 530
1520 `
1530 ` edit/delete customer names
1540 CLS:CC=0
1550 PRINT TAB(20);'' Edit or Delete Customer names
1560 PRINT
1570 INPUT''Customer number ``;CN$
1580 FOR I=1 TO N1

```

```

1590 IF A$(I,1) <> CN$ THEN 1720
1600 FOR J=1 TO 7
1610 PRINT J;'' '';A$(I,J)
1620 NEXT J
1630 PRINT
1640 INPUT''Change which line (0 to delete or to terminate
search) '';CC
1650 IF CC<>0 THEN 1690
1660 INPUT''Do you wish to delete this customer (y/n)'';XX$
1670 IF XX$='y' OR XX$='Y' THEN A$(I,2)='''
1680 GOTO 1720
1690 LINE INPUT''Enter the correct information '';A$(I,CC)
1700 INPUT''More changes (y/n) '';XX$
1710 IF XX$='Y' OR XX$='y' THEN 1640
1720 NEXT I
1730 IF CC=2 THEN GOSUB 1120
1740 GOTO 530
1750 `
1760 ` print mailing label routine
1770 CLS:PRINT TAB(15);'' Print Mailing Labels''
1780 INPUT''Align your labels and press enter '';XX
1790 FOR I=1 TO N1
1800 LPRINT TAB(24);A$(I,1)
1810 LPRINT A$(I,2)
1820 LPRINT A$(I,3)
1830 LPRINT A$(I,4);'' '';A$(I,5)
1840 FOR J=1 TO 2
1850 LPRINT'' ``
1860 NEXT J
1870 NEXT I
1880 GOTO 530
1890 `
1900 `error trap for file not found
1910 IF ERR <> 53 THEN ON ERROR GOTO 0
1920 GOSUB 1280
1930 RESUME 330
1940 `
1950 `process invoices routine
1960 CLS
1970 INPUT''Enter the date for the invoices '';DA$
1980 TL=0:TA=0
1990 INPUT''Invoice for Customer # (or 0 to quit) '';CN$
2000 IF CN$='0' THEN 530

```



```

2010 FOR I=1 TO N1
2020   IF A$(I,1) <> CN$ THEN 2330
2030   LPRINT TAB(16);A$(0,0);TAB(64);DA$
2040   PRINT A$(I,1);'' '';A$(I,2);'' '';A$(I,3)
2050   LPRINT'' ''
2060   LPRINT'' ''
2070   LPRINT TAB(6);A$(I,2);TAB(52);''Resale # '';A$(I,7)
2080   LPRINT TAB(6);A$(I,3);TAB(52);''Telephone: '';A$(I,6)
2090   LPRINT TAB(6);A$(I,4);'' '';A$(I,5);TAB(52);''Cust # '';A$(I,
1)
2100   FOR K=1 TO 7
2110     LPRINT
2120     NEXT K
2130     P1=0:P2=0:P3=0:Q=0:Q$='''':P4=0:T1=0
2140     INPUT''What is the quantity for this order '';Q
2150     IF Q=0 THEN Q=1
2160     LINE INPUT''Description of services '';Q$
2170     INPUT''What is the per/m price '';P1
2180     INPUT''What is the flat rate price '';P2
2190     T1=((Q/1000)*P1)+P2
2200     INPUT''Is there a discount (y/n) '';XX$
2210     IF XX$='''n'' OR XX$='''N'' THEN 2230
2220     INPUT''What is the discount rate (like .15) '';P3
2230     P4=T1*P3:P4=P4*-1:TL=TL+T1-ABS(P4):P5=F3*TL
2240     LPRINT Q;'' '';Q$;:LPRINT TAB(46);USING F1$;P1;:LPRINT
TAB(56);USING F1$;P2;:LPRINT TAB(65);USING F2$;T1
2250   IF XX$='''Y'' OR XX$='''y'' THEN LPRINT TAB(44);''Less '';P3*100;''%
discount '';TAB(65);USING F3$;P4
2260   INPUT''Are there more charges for this customer (y/n)'';XX$
2270   IF XX$='''Y'' OR XX$='''y'' THEN LPRINT'' '';LPRINT'' '';GOTO 2130
2280   IF A$(I,7)=''' THEN LPRINT'' '';LPRINT TAB(52);''Sales tax'';
TAB(65);USING F2$;P5:TL=TL+P5
2290   LPRINT'' '';LPRINT'' '';LPRINT TAB(35);''Total due per this
invoice '';TAB(65);USING F2$;TL
2300   LPRINT CHR$(12)
2310   TA=TA+TL
2320   AA=VAL(A$(0,0)):AA=AA+1:A$(0,0)=STR$(AA)
2330   NEXT I
2340   TL=0:CLS
2350   GOTO 1990
2360   END `of program

```

Change lines for Invoice.Bas for Tandy Models I and III

```
Changed->100 REM * Invoice/Bas * Invoice program 30 Aug 88 IMS
Changed->140 CLEAR 2000
Changed->170 'LOCATE X,Y:RETURN ' MS-DOS, GW-BASIC
Changed->180 PRINT@((X-1)*64)+(Y-1),,:RETURN ' Tandy Models I/III
Changed->250 'WIDTH LPRINT 132
Changed->350 OPEN 'I',1,'invoice/dat''
Changed->1180 TT$=A$(I,J):A$(I,J)=A$(L,J):A$(L,J)=TT$
Changed->1280 OPEN 'O',1,'invoice/dat''
Changed->1910 IF (ERR/2)+1<>54 THEN ON ERROR GOTO 0
```

Change lines for Invoice.Bas for Tandy Models II and IV

```
Changed->100 REM * Invoice/Bas * Invoice program 30 Aug 88 IMS
Changed->170 'LOCATE X,Y:RETURN ' MS-DOS, GW-BASIC
Changed->190 PRINT@((X-1),(Y-1)),,:RETURN ' Tandy Models II/IV
Changed->250 'WIDTH LPRINT 132
Changed->350 OPEN 'I',1,'invoice/dat''
Changed->1280 OPEN 'O',1,'invoice/dat''
```

Notes

Have you ever looked for a neat way to automatically center headings? Especially when the heading was a string variable and could be of any indetermined length?

Here is one way to do it. Let's assume that A\$ is equal to "CodeWorks Magazine" and is defined as such someplace in your program. If you then PRINT TAB((80-LEN(A\$))/2);A\$ it will automatically center whatever A\$ is on your screen.

Naturally, if your screen is only 64 characters wide, you would use 64 instead of the 80 just shown. And of course, you can LPRINT instead of just PRINT and get the heading centered on your printer as well.

The little formula calculates the tab position by taking the length of the string from 80 and dividing by two. Terry made a defined function out of this in his Randemo series.

Outline.Bas

The Long-awaited Part 3 of 3

Terry R. Dettmann, Associate Editor. Here at long last is the final installment of the outline program which we started last year. It's not as "full blown" as the commercial jobs, but will give you an idea of what they are and how they work. We just got it last night, when Terry came to visit and haven't had that much time to play with it ourselves yet.

Outline - The End of an Odyssey

After all of this time away from CodeWorks, it's good to be back. All of you deserve an ending to my Outline series ... even if none of you are still reading it. I apologize to all of you for the delay in getting the final installment to you.

Since it's been a long time since we started, it's probably a good idea to start off with a little review of just what we're trying to accomplish. After that, we'll go into the program from beginning to end, and then print the completed outline program.

What's an Outline?

Is that a question I need to answer? REALLY? I'm sure Miss Fensterwink in the 7th grade taught you THAT. An outline is a structuring device for organizing information. It's useful in design, research, and just plain learning. But you already knew that didn't you (Miss Fensterwink is grading you on your reply).

Maybe a better question, is "What's an outline PROGRAM?". Have you ever seen one? Programs like ThinkTank, Ready, and MaxThink on the PC and MORE and ACTA on the Macintosh (there are a lot more of them out there) are not only popular, but major tools for the thinking trade. I have several that I use on different computers and have always liked them. It wasn't until the PC though that I found an outline processor that was usable.

So what does an outlining program do? As a bare minimum, it allows you to create an outline and manipulate it as you go along. You can build individual entries in the outline (often called headlines) for categories and sub-categories. You can restructure the outline by moving headlines (with all subheadlines) from one place to another within the outline.

The outline program we've been building here is for the same purpose (but on a much simpler scale). What we're trying to do here is create a simple outline with the basic structure so we can see how such a thing is created and what kinds of work has to be done INSIDE the program to accomplish it. This isn't the ONLY way to do it, but it follows naturally from earlier discussions of linking items together.

The Program and How it Works

The basic idea behind the Outline.Bas program is an attempt to create lines in the outline which fall into an internal structure which mirrors the way an outline works.

If we think about any given headline, there are probably headlines which fit logically below it. For example:

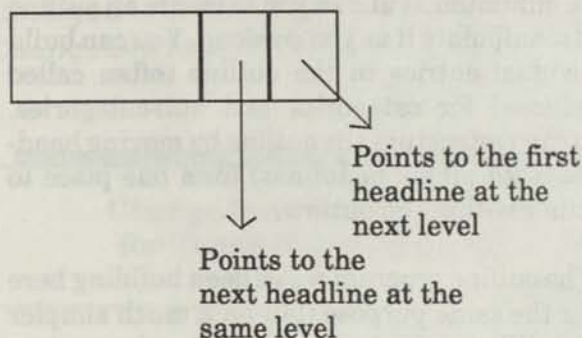
- I. Outlining
 - A. Apologies for delay
- II. What's an Outline
 - A. Outlines are already known
 - B. What's an outline program?

III. The Program and How it Works

A. List Linking

This outline has a structure of lines subordinate to other lines. As you listed a topic, like "What's an Outline", you would want to list subtopics under it which amplify the topic itself. These subtopics don't belong under another headline, they belong as subheads on one and only one headline.

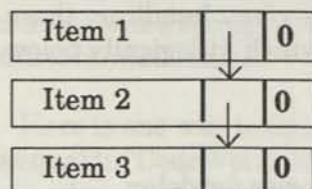
In programming, there is a way to show such structure, the 'Linked List'



[FIGURE 1 - Basic linking structure]

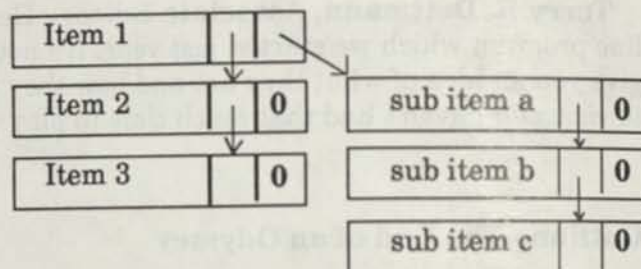
If you go back through your Codeworks magazines, you'll find we've talked about linked lists before. They are simply a way to connecting together related information, in this case the outline pieces.

The structure is two dimensional within an outline. First, there is a given level (headings I. II. III.). These follow a definite order or sequence. To show this sequence, we link each headline to the next:



[FIGURE 2 - One level linking structure]

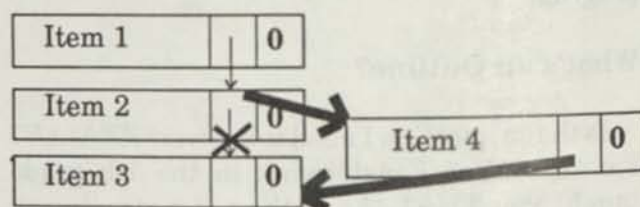
When we add another level below the first, that layer splits into a layer for each of the headings on the layer above. This makes it clear that we need a way to get to the first line of the next level easily. A downward linking accomplishes that:



[FIGURE 3 - Linking to the next level]

The whole objective of the program then is to create this structure to the data and manipulate it. Let's look at the basic operations we need to carry out and see how they're done.

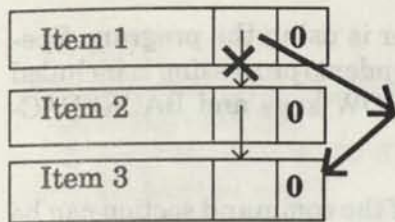
To add a new headline to the document, we need to link our new lines into the existing outline as shown in figure 4.



[FIGURE 4 - Adding new lines to the outline]

All we need to do is to change the links from the old lines to point to the new ones and change the pointer from the new one to point to where the next one is as shown (WHEW! that's a real mouthful!).

To delete a headline, we do the opposite. We disconnect one line from the list and make the pointers around it point around it as if it weren't there. Figure 5 shows this:



[FIGURE 5 - Deleting a line from the outline]

If we want to move a headline from one place to another, we can delete it (keeping it still in tact) and then re-add it back in the new place. Most outline processors refer to this as a MOVE operation.

Implementing these simple operations is a little tougher than the explanation. Let's see how we've done it in Outline.Bas. You've seen some of it before, but we'll do it all again.

Outline.bas

The outline.bas program implements these basic operations to demonstrate how an outline program works. Before we go on to the program itself though, let's make a few things clear. First, outline.bas is a VERY simple example of how outlining can be done on the PC, it is NOT a complete outlining package. Such a thing would run into thousands of lines of code and require a reasonable size book (several hundred pages) of operating manual to explain just how to run it. Second, this program will appear to be rather slow on older machines (early PC's and TRS-80's) because the processors are slow. You can improve the operation by compiling the program (I have tested compiling with Microsoft Quick-BASIC satisfactorily).

If you find this program interesting, I'd highly recommend that you look into a serious outliner from an established source. For the PC, both MaxThink and ThinkTank are good outline processors, each with their own particular view of the world. The program GrandView also includes outlines (as do more modern Word Processors) and much more sophistication about out-

lines. Sidekick Plus for the PC includes a popup outliner feature as does Ready!. The outliner of choice on the PC for serious work is Grandview.

On the Macintosh, Acta is an outliner Desk Accessory and Sidekick for the Mac includes one. MORE is the outliner of choice for serious work on the Mac.

If you have any questions on outliners or outlining, please feel free to write me here c/o Code-works or contact me directly on Compuserve (72076,2611) or via MCI Mail. I'd be happy to give you more information on outliners and outlining than we can put here. I've had experience with all of them I've mentioned and would be glad to give you some advice. If you're on Compuserve, I'd suggest you check into the SYMANTEC forum (GO SYMANTEC) where there are active discussions about outliners and outlining and you'll also find me frequently.

Let's take a tour through the program to see how it works.

Lines 10-190 - Program Initialization

To initialize the program, we have to setup data structures and constants for use during the program's execution. Some languages like C and Pascal include special data types for handling jobs like this (the 'Record' in Pascal and the 'Structure' in C). In the flavor of BASIC we talk about here, we don't have anything like that. Newer versions of BASIC for the IBM-PC and Macintosh now include special data forms.

To simplify our lives, we first define a set of constants which can be changed for different purposes. For example, LN the number of lines on the screen (24) and WD the width of the screen (80). To build the headline list, we can create arrays to hold the headlines (LN\$()), the links (LK()), and other information we'll go into as we need it.

Other constants are created for convenience like TRUE and FALSE (so we can have IF statements which have TRUE or FALSE conditions),

characters for carriage returns and line feeds and so forth. We initialize the outline to an empty state, and then go on.

Lines 200-240 - Main Program Loop

The main program loop is the circulatory system of the program. It controls the basic logic of the program's operation which is:

1. Display the current screen of data
2. Wait for an input
3. Respond to that input
4. Go back to the display step 1.

Everything that makes the program work works from this loop.

Lines 400-999 - General Subroutines

A few general level subroutines are put here to centralize handling for specific tasks which are often computer dependent. The most specific example of this is repositioning the cursor on the screen. Subroutine 400 is implemented to do this on an IBM PC or compatible under Microsoft BASIC. It's also been tested with Microsoft BASIC on a Macintosh successfully. However, on a TRS-80 with an older Microsoft BASIC, we have to use a PRINT @ statement to position the cursor.

Lines 1000-1999 - Display Subroutines

The display subroutines have specific tasks related to getting information to the screen. For example, the simplest is subroutine 1100 which prints the title line. Subroutine 1000 gets more complex as it controls printing the currently visible outline segment to the screen. You'll notice that this subroutine doesn't have ANY print statements in it. The actual printing is done by subroutine 1200 which assembles a line and then prints it. The other routines here support the display processing.

Lines 2000-3999 - Command Subroutines

These subroutines are responsible for interac-

tion with whomever is using the program. Special (computer dependent) processing is included for processing ARROW keys and BACKSPACING during entry.

The basic logic of the command section can be expressed like this:

1. Prompt the user for a command with the COMMAND=> prompt
2. Accept the user's input line
 - a. If the user presses an ARROW key during input
 - b. Respond immediately
3. Check the input line for a command
4. Return to the Main Loop

The critical routines here are 2100 which enters a line from the user and it's supporting routines 2200 - enter a character, 3450 - enter COMMAND MODE, and 2300 - backspace 1 character.

Specific subroutines are assigned single tasks to accomplish (for example, 3000 which moves UP one line). Each of these supports one and only one logical operation. This made it easier to put the program together and makes it easier to create new features by adding specific handlers for each new feature desired.

Lines 4000-4999 - File Operations and outline operations

The simplest operations conceptually are the file operations. Loading a file is simply put:

1. If End of File, we're done
2. Otherwise read a line
3. Determine the line's level in the outline
4. Add it to the growing outline
5. Go back to step 1

Subroutine 4000 front ends this operation and subroutine 5400 actually carries it out.

To write the outline to a file, conceptually again, we do something very simple:

1. If End of Outline, we're done
2. Select next line
3. Print the line to the file
4. Go back to step 1

Here though, we get a bit more complicated by the fact that we have to follow the thread of the linking within the outline structure. Subroutine 4100 front ends the operation, subroutine 5500 carries it out.

The hardest subroutine conceptually is the actual creation of a line within the structure. Subroutines 5700 and 6100 do this for adding lines during reading and during ADD MODE, respectively. This is where most of the program debugging has taken place because subtle logical errors here can show up everywhere else in the program. If you link things to make a circular linkage, then the program will go into an infinite loop just trying to print the outline to the screen.

If you look back to the figures earlier, you'll see how this can be. When we're adding lines to the structure, we can easily run into problems when we lose track of the linking numbers which tell us which is the correct next line. The same thing happens if we delete a headline and then re-add it using the UNDO. The difficulty arises because of the possibilities. They're simple, but require some tricky coding to check for. If you go through each step, one at a time, you'll see that we've had to cover a variety of cases. It's useful to go through them and write them out. You'll get a lot out of the exercise and it will serve you well in writing your own programs which involve linking data.

Lines 7000-7999 - Printing Routines

The very simple printing utility (based on the screen print portion of the program at line 1000) is included here to print the outline to the screen and then wait. Changing the print statements to direct their output to your hard copy printer will allow you to print the outline directly.

Lines 8000-8999 - Move Lines Routines

The move lines routines allow you to restructure the outline by selecting a line and moving it to another line on the same level. When you select a line and press 'M', the line will be deleted using the delete routine, and then you will be prompted for where to put it. Use the up and down arrows to position the cursor to the new location and press RETURN and the line will be pasted in after the current line.

Lines 9000-9999 - Debugging Routines

Useful debugging routines are put in this area and called as needed. Most have been purged from the listing for space, but I'm leaving one which is used after a program crash to get a look at the outline data formats. If a crash takes you back to the BASIC interpreter, then all you need to do is type 'GOSUB 9000<RETURN>' to have the program print the status of the outline structure. You can see what's linked to what by following the numbers.

So how does it all work?

You can start the outline.bas program and it will start up with an empty first headline (the master one which anchors the whole outline). You can't do more than title this one because it is the foundation for the whole document.

When you select 'ADD MODE' by typing 'A<RETURN>', from that point, anything you type until you hit an escape key will be added to the outline. If you press the RIGHT ARROW key, the outline will move down one level. Pressing the LEFT ARROW key will move it back up one level. Remember, this is a demonstration, not a full featured outline processor. You're protected against going too far either up or down, but you can still mess it up. Try it though, you can structure your thoughts as you enter them.

When you press the ESCAPE key, ADD MODE ends. You can move around using the up and down arrow keys. EDIT MODE allows you to change the contents of a single headline and DELETE MODE lets you delete a single head-

line (and those headlines below it). Selecting MOVE moves a line to a new position. Selecting PRINT prints the outline to the screen.

The sample outliner is too simple for really serious work. You're limited to one screen of display (though there is provision to set and change the top line of the screen display). You also aren't protected against many of the errors that you could commit. The program is an EXAMPLE of a practical program design. Commercial products use some similar type of data structure to accomplish what they do, but a full listing would take up more pages than we publish in a whole year.

At its best, this is a simplistic example of outline processing. Full featured outline processors include many features which we aren't even attempting here. The listing by itself for a full featured outliner is larger than a whole issue of CodeWorks. But this is still a practical program for creating small outlines to help refine your thoughts about something. Whether you just want to arrange the information you have or you want to create something new, an outliner like this could be your key.

Outline.Bas for GW BASIC

```
10 REM -- Outline.bas, a Program for Codeworks Magazine
20 REM -- by Terry R. Dettmann
30 DEFINT A-Z
35 `n=max number of lines, nl=number of existing lines
36 `tl=top line on screen, cl=current line
37 `wd=screen width, ln=screen length
40 N = 200: NL = 1: TL = 0: WD = 80: LN = 24: CL = 0: SP = -1
45 `ln$( )=text for each node, lk( )=dynamic linking array
46 `lv( )=indent level of current text line
47 `nm( )=sequence number of current text line
50 DIM LN$(N), LK(2, N), LV(N), NM(N), LL(10), STK(10)
55 `cr$=carrage return, bs$=backspace
60 CR$ = CHR$(13): BS$ = CHR$(8): ESC$ = CHR$(27)
70 FALSE = 0: TRUE = NOT FALSE
80 GOSUB 3550
90 DWN = FALSE: UP = FALSE
190 CLS : GOSUB 3450
200 REM -- Main Program Loop
201 CLS
205 GOSUB 1100
210 GOSUB 1000
220 GOSUB 2000
230 ON CMD GOSUB 3400, 3500, 3600, 4000, 4100, 500, 3450, 7000,
    8000
240 GOTO 200
400 REM -- gotoxy
```



```

410 LOCATE X, Y: RETURN
500 REM -- End of program
510 CLS : PRINT ``Thank you for coming``: END
1000 REM -- Display the current outline segment on the screen
1010 I = 0: NM = TL
1020 IF I >= 20 THEN RETURN
1030     GOSUB 1200
1035     IF SP = -1 AND LK(0, NM) = -1 AND I > 0 THEN RETURN
1036     I = I + 1
1040     IF LK(2, NM) >= 0 THEN GOSUB 1300: NM = LK(2, NM): GOTO
        1020
1050     IF LK(0, NM) >= 0 THEN NM = LK(0, NM): GOTO 1020
1060     GOSUB 1350: IF NM < 0 THEN RETURN
1070     IF LK(0, NM) >= 0 THEN NM = LK(0, NM): GOTO 1020
1080     GOTO 1060
1100 REM -- Print Title Line
1110 X = 1: Y = 1: GOSUB 400: PRINT ``Codeworks Outline Processor``;
1120 X = 1: Y = WD - 15: GOSUB 400: PRINT MD$;
1130 X = LN - 2: Y = 1: GOSUB 400: PRINT STRING$(WD, ``-``);
1140 X = LN - 1: Y = 1: GOSUB 400
1141 PRINT ``(A)dd (C)ommand (D)elete (E)dit (L)oad (M)ove (P)rint
        (Q)uit (S)ave``;
1150 RETURN
1200 REM -- print a single line
1210 X = I + 2: Y = 3: GOSUB 400
1211 IF CL = NM THEN LF$ = ``<``: RT$ = ``>`` ELSE LF$ = `` ``: RT$ = `` ``
1220 PRINT STRING$(LV(NM) * 2, `` ``);
1230 PRINT USING ``!##! ``; LF$; NM(NM); RT$; : PRINT LN$(NM);
1240 RETURN
1300 REM -- push level onto stack
1310 IF SP >= 10 THEN RETURN
1320     SP = SP + 1: STK(SP) = NM
1330     RETURN
1350 REM -- pop top level off stack
1360 IF SP < 0 THEN NM = -1: RETURN
1370     NM = STK(SP): SP = SP - 1
1380     RETURN
2000 REM -- command entry
2010 X = LN: Y = 1: GOSUB 400
2015 NC = WD: GOSUB 2400: GOSUB 400
2020 PRINT ``Command=>``;
2030 NC = WD - 10: GOSUB 2100
2040 GOSUB 2500
2050 RETURN
2100 REM -- input a line

```

```

2110 IN$ = ''
2120 GOSUB 2200: IF C$ = CR$ THEN RETURN
2125 IF C$ = ESC$ THEN GOSUB 3450: IN$ = '' : RETURN
2130 IF C$ = BS$ THEN GOSUB 2300: GOTO 2120
2135 IF C$ < `` `` OR C$ > ``~`` THEN 2120
2140 IF LEN(IN$) >= NC THEN 2120
2150 IN$ = IN$ + C$: PRINT C$; : GOTO 2120
2200 REM -- read one character
2205 ARROW = FALSE
2210 C$ = INKEY$: IF C$ = `` THEN 2210
2220 GOSUB 2250: IF ARROW THEN 2200
2230 RETURN
2250 REM -- check for arrow keys
2260 IF LEN(C$) = 1 THEN RETURN
2270 AW = ASC(MID$(C$, 2, 1))
2280 IF AW = &H48 THEN GOSUB 3000: GOTO 2290
2281 IF AW = &H50 THEN GOSUB 3100: GOTO 2290
2282 IF AW = &H4B THEN GOSUB 3200: GOTO 2290
2283 IF AW = &H4D THEN GOSUB 3300: GOTO 2290
2290 ARROW = TRUE: RETURN
2300 REM -- backspace
2310 IF LEN(IN$) = 0 THEN RETURN
2320 IN$ = LEFT$(IN$, LEN(IN$) - 1)
2330 X = CSRLIN: Y = POS(0) - 1: GOSUB 400
2340 PRINT `` ``; : GOSUB 400
2350 RETURN
2400 REM -- Clear to end of line
2410 PRINT STRING$(NC - 5, `` ``); : RETURN
2500 REM -- Parse the command line
2505 IF MD <> 0 THEN CMD = 0: GOTO 2540
2506 IF IN$ = `` THEN 2540
2510 CS$ = ``AaEeDdLlSsQqCcPpMm``
2520 CMD = INT((INSTR(CS$, LEFT$(IN$, 1)) + 1) / 2)
2530 IF CMD <> 0 THEN 2590
2540 ON MD GOSUB 5100, 5200, 5300, 5400, 5500
2590 RETURN
3000 REM -- move up one line
3010 IF LK(1, CL) > 0 THEN CL = LK(1, CL)
3015 GOSUB 3450: GOSUB 1100: GOSUB 1000
3020 RETURN
3100 REM -- move down one line
3110 IF LK(0, CL) > 0 THEN CL = LK(0, CL)
3115 GOSUB 3450: GOSUB 1100: GOSUB 1000
3120 RETURN
3200 REM -- move up one level
3205 IF LL = 1 THEN RETURN

```

```

3208 IF MD = 0 THEN 3220
3210 UP = TRUE: RETURN
3220 IF LK(1, CL) >= 0 THEN CL = LK(1, CL): LL = LL - 1
3221 DBG$ = ``Line 3221 + `` + STR$(LK(1, CL)): GOSUB 9100
3225 GOSUB 3450: GOSUB 1100: GOSUB 1000
3230 RETURN
3300 REM -- move down one level
3305 IF LL = 5 THEN RETURN
3308 IF MD = 0 THEN 3320
3310 DWN = TRUE: RETURN
3320 IF LK(2, CL) > 0 THEN LL = LL + 1: CL = LK(2, CL)
3325 GOSUB 3450: GOSUB 1100: GOSUB 1000
3330 RETURN
3400 REM -- set add mode
3410 MD = 1: MD$ = ``ADD MODE ``: RETURN
3450 REM -- set command mode
3460 MD = 0: MD$ = ``COMMAND MODE``: RETURN
3500 REM -- set edit mode
3510 MD = 2: MD$ = ``EDIT MODE ``: RETURN
3550 REM -- initialize new outline
3560 LN$(0) = ``Outline Title``: LK(0, 0) = -1: LK(1, 0) = -1: LK(2,
    0) = -1: LV(0) = 0: NM(0) = 1
3570 RETURN
3600 REM -- delete lines
3610 MD = 3: MD$ = ``DELETE MODE ``: RETURN
4000 REM -- Load file
4010 X = LN: Y = 1: GOSUB 400
4020 NC = WD: GOSUB 2400: GOSUB 400
4030 PRINT ``Filename=> ``;
4040 NC = WD - 10: GOSUB 2100
4050 IF IN$ = ```` THEN RETURN
4060 FF$ = IN$ + ``.OUT``
4070 GOSUB 5400: RETURN
4100 REM -- Save file
4110 X = LN: Y = 1: GOSUB 400
4120 NC = WD: GOSUB 2400: GOSUB 400
4130 PRINT ``Filename=> ``;
4140 NC = WD - 10: GOSUB 2100
4150 IF IN$ = ```` THEN RETURN
4160 FF$ = IN$ + ``.OUT``
4170 GOSUB 5500: RETURN
5100 REM -- Add
5110 LN$(NL) = IN$
5120 IF CL = 0 THEN LV(NL) = 1 ELSE LV(NL) = LV(CL)
5125 IF DWN THEN LV(NL) = LV(NL) + 1
5126 IF UP THEN LV(NL) = LV(NL) - 1

```

```

5130 IN = NL: GOSUB 6100: CL = NL
5140 NL = NL + 1
5145 UP = FALSE: DWN = FALSE
5150 RETURN
5200 REM -- Edit
5210 LN$(CL) = IN$
5220 GOSUB 3450: CLS : GOSUB 1100: GOSUB 1000
5230 RETURN
5300 REM -- Delete
5310 DL = CL
5320 IF LK(1, CL) >= 0 THEN LK(0, LK(1, CL)) = LK(0, CL)
5321 IF LK(0, CL) >= 0 THEN LK(1, LK(0, CL)) = LK(1, CL)
5330 XL = LK(0, CL)
5340 IF XL < 0 THEN GOSUB 3450: CLS : GOSUB 1100: GOSUB 1000:
RETURN
5350 NM(XL) = NM(XL) - 1
5360 XL = LK(0, XL)
5370 GOTO 5340
5400 REM -- Load
5410 OPEN ``I'', 1, FF$
5420 IN = 0: LL(0) = 0: LL = 0
5430 IF EOF(1) THEN 5495
5440 LINE INPUT #1, IN$
5450 GOSUB 5600
5460 LN$(IN) = IN$
5470 GOSUB 5700
5480 IN = IN + 1
5490 GOTO 5430
5495 CLOSE : RETURN
5500 REM -- Save
5505 OPEN ``O'', 1, FF$
5510 I = 0: NM = 0: SP = -1
5520 REM -- start of loop
5530 GOSUB 6000
5535 IF SP = -1 AND LK(0, NM) = -1 AND I > 0 THEN CLOSE :
RETURN
5536 I = I + 1
5540 IF LK(2, NM) >= 0 THEN GOSUB 1300: NM = LK(2, NM): GOTO
5520
5550 IF LK(0, NM) >= 0 THEN NM = LK(0, NM): GOTO 5520
5560 GOSUB 1350: IF NM < 0 THEN RETURN
5570 IF LK(0, NM) >= 0 THEN NM = LK(0, NM): GOTO 5520
5580 GOTO 5560
5600 REM -- determine the line's level
5610 LV(IN) = 0
5620 IF MID$(IN$, 1, 1) <> `` `` THEN LN$(IN) = IN$: RETURN

```

```

5630     LV(IN) = LV(IN) + 1
5640     IN$ = MID$(IN$, 2)
5650     GOTO 5620
5700 REM -- link line into structure
5705 IF IN = 0 THEN LK(0, 0) = -1: LK(1, 0) = -1: LK(2, 0) = -1:
      NM(0) = 1: GOTO 5900
5710 IF LV(IN) = LL THEN 5800
5720 IF LV(IN) > LL THEN 5850
5730 LK(0, IN) = -1: LK(1, IN) = LL(LV(IN)): LK(2, IN) = -1
5740 LK(0, LL(LV(IN))) = IN: NM(IN) = NM(LL(LV(IN))) + 1
5750 GOTO 5900
5800 LK(0, IN) = -1: LK(1, IN) = IN - 1: LK(2, IN) = -1
5810 LK(0, IN - 1) = IN: NM(IN) = NM(IN - 1) + 1
5820 GOTO 5900
5850 LK(0, IN) = -1: LK(1, IN) = -1: LK(2, IN) = -1
5860 LK(2, IN - 1) = IN: NM(IN) = 1
5870 GOTO 5900
5900 LL(LV(IN)) = IN: LL = LV(IN)
5910 RETURN
6000 REM -- save one line to file
6010 PRINT #1, STRING$(LV(NM) * 1, ' '); LN$(NM)
6020 RETURN
6100 REM -- link line into structure
6110 IF LV(IN) = LL THEN 6200
6120 IF LV(IN) > LL THEN 6250
6130 LK(0, IN) = -1: LK(1, IN) = LL(LV(IN)): LK(2, IN) = -1
6140 LK(0, LL(LV(IN))) = IN: NM(IN) = NM(LL(LV(IN))) + 1
6150 GOTO 6300
6200 LK(0, IN) = LK(0, CL): LK(1, IN) = CL: LK(2, IN) = -1
6210 LK(0, CL) = IN: NM(IN) = NM(CL) + 1
6220 GOTO 6300
6250 LK(0, IN) = -1: LK(1, IN) = CL: LK(2, IN) = -1
6260 LK(2, CL) = IN: NM(IN) = 1
6270 GOTO 6300
6300 LL(LV(IN)) = IN: LL = LV(IN)
6310 RETURN
7000 REM -- Display the current outline segment on the screen
7010 I = 0: NM = 0: CLS
7020 REM beginning of print loop
7030     GOSUB 7200
7035     IF SP = -1 AND LK(0, NM) = -1 AND I > 0 THEN 7090
7036     I = I + 1
7040     IF LK(2, NM) >= 0 THEN GOSUB 1300: NM = LK(2, NM): GOTO
      7020
7050     IF LK(0, NM) >= 0 THEN NM = LK(0, NM): GOTO 7020
7060     GOSUB 1350: IF NM < 0 THEN 7090

```

```

7070     IF LK(0, NM) >= 0 THEN NM = LK(0, NM): GOTO 7020
7080         GOTO 7060
7090 GOSUB 7100: CLS : GOSUB 3450: GOSUB 1100: GOSUB 1000: RETURN
7100 REM -- time delay (not needed if printing to paper)
7110 FOR IX = 1 TO 10000: NEXT IX: RETURN
7200 REM -- print a single line
7220 PRINT STRING$(LV(NM) * 2, `` ``);
7230 PRINT USING `` ## ``; NM(NM); : PRINT LN$(NM)
7240 RETURN
8000 REM -- move a headline
8010 GOSUB 5300: XL = CL
8020 GOSUB 8100
8030 CL = XL: GOSUB 3450: CLS : GOSUB 1100: GOSUB 1000: RETURN
8100 REM -- insert the deleted line at the current position
8110 GOSUB 8200
8120 LK(0, DL) = LK(0, NL): LK(0, NL) = DL: DL = -1
8130 RETURN
8200 REM -- select line
8201 X = 24: Y = 1: GOSUB 400: PRINT ``SELECT INSERT LOCATION``;
8210 NL = CL
8220 CL = NL: GOSUB 1000: GOSUB 8300: IF C$ = CR$ THEN RETURN
8230 IF MID$(C$, 1, 1) <> CHR$(0) THEN 8220
8240 IF ASC(MID$(C$, 2, 1)) = &H48 THEN GOSUB 8400: GOTO 8220
8250 IF ASC(MID$(C$, 2, 1)) = &H50 THEN GOSUB 8500: GOTO 8220
8260 GOTO 8220
8300 REM -- Get a character
8310 C$ = INKEY$: IF C$ = ```` THEN 8310
8320 RETURN
8400 REM -- Move up one line
8410 IF LK(1, NL) > 0 THEN NL = LK(1, NL)
8420 RETURN
8500 REM -- Mode down one line
8510 IF LK(0, NL) > 0 THEN NL = LK(0, NL)
8520 RETURN
9000 REM - print variable status
9010 FOR I = 0 TO NL
9020     PRINT I; LV(I); NM(I); LN$(I); LK(0, I); LK(1, I); LK(2,
        I)
9030 NEXT I
9040 STOP
9100 REM -- display debugging information
9110 X = CSRLIN: Y = POS(0)
9120 LOCATE 24, 1
9130 PRINT DBG$;
9140 LOCATE X, Y
9150 RETURN

```

Handy Order Form

| | | |
|--|---------|--|
| RENEW Subscription Nov/Dec 88 through Sep/Oct 89 | \$24.95 | |
| All third year issues, Nov 87 through Sep 88 | \$24.95 | |
| All second year issues, Nov 86 through Sep 87 | \$24.95 | |
| All first year issues, Sep 85 through Sep 86 | \$24.95 | |
| 1st Year Program Disk (issues 1 through 7) (Specify computer type below) | \$20.00 | |
| 2nd Year Program Disk (issues 8 through 13) (Specify computer type below) | \$20.00 | |
| 3rd Year Program Disk (issues 14 through 19) (Specify computer type below) <i>Available now</i> | \$20.00 | |
| NEW! "Starting with MS DOS" 40-page book explains all | \$7.00 | |

Postage and handling charges already included in all prices.

We can supply program diskettes for PC/MS DOS, TRSDOS 1.3 (Tandy Model 3), TRSDOS 6.x (Tandy Model IV) and most CP/M MBASIC formats, on 5 1/4 inch diskettes. When ordering diskettes, specify your computer type.

COMPUTER TYPE _____

Check/MO enclosed

Charge to my VISA/MC _____ exp _____

Ship to: Name _____

Address _____

City _____ State _____ Zip _____

Clip or photocopy and mail to:
CodeWorks, 3838 S. Warner St. Tacoma WA 98409

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

VISA/Master Card only

589

Index & Download

What's happening with both

Here are the updates to bring CWinindex.Dat up to date through the last issue. The entire index for the first three years of CodeWorks is on our yearly diskette.

Notes, string packing, issue 22, page 3

Notes, rename for Tandy Models II/IV, issue 22, page 3

Notes, draw a circle in BASIC, issue 22, page 5

Notes, NFL88-89 recap, issue 22, page 5

Beginning BASIC, two-level sorting with demo program, issue 22, page 6

Budget.bas, main program, issue 22, page 9, a home budget program

Notes, how to scroll in MS DOS GW BASIC, issue 22, page 18

Notes, not using remarks in data lines, issue 22, page 18

Insort.bas, main program, issue 22, page 19, sort on input programs

Flow.bas, main program, issue 22, page 24, a line number reference utility

Pay2.bas, main program, issue 22, page 30, an updated Pay.bas program

Download, notes on download, issue 22, page 40

CWinindex.dat, updates to this index, issue 22, page 40

We have had another power hit on the download and this time it took out the modem. As luck would have it, Bob happened to have one and we put that into service. This got us to thinking about the download and what it was actually accomplishing. There are days on end without anyone using it. At new issue time, a very few regulars take down a few things. It's costing us the extra phone line and expensive repair costs when it goes down at least twice per year.

We are considering eliminating the download and publishing our disks twice per year instead. Your thoughts on this would be appreciated.

CodeWorks

3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
U.S. Postage
PAID
Permit 774
Tacoma, WA

2910
ERICKSON, MIKE / KRJB
BOX 250
MONTE RIO CA 95462

CODEWORKS

Issue 24

Jul/Aug 1989

CONTENTS

| | |
|--------------------------------------|-----------|
| Editor's Notes | 2 |
| Forum | 3 |
| Beginning BASIC | 5 |
| Tracker.Bas | 8 |
| Rcard1.Bas | 25 |
| Basic Techniques | 32 |
| Notes | 35 |
| Etax89.Bas | 36 |
| Artificial Intelligence | 37 |
| Renewal order form | 39 |
| Cwindex and Download | 40 |

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

(c) 1989 80-Northwest Publishing Inc. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, the publisher assumes no responsibility for errors or omissions. **Unless otherwise noted, all programs presented in this publication are placed in public domain.** Please address all correspondence to **CodeWorks, 3838 South Warner Street, Tacoma, Washington 98409**

Telephone
(206) 475-2219

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 is \$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. **VISA and MasterCard orders are accepted by mail or by phone (206) 475-2219.**

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in the USA. 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

Did you miss the little curliques on the cover in the last issue? We did a little redesign and forgot to scan them into the computer until it was too late and we had to go to press. They are back now, but screened back so that they don't jump off the page. With that, the entire issue is now produced on desktop publishing equipment. The old overlays we had for the cover, after being punished for some 22 issues, got dirty and worn out. Everyone who printed the magazine cut off just a little more so that they could find the registration marks, until finally there wasn't that much of it left.

As I mention in Forum, we are getting this issue out ahead of schedule, so we haven't heard from you about the download. Well, just after the last issue went out in the mail, the download went up in smoke - literally. It sort of took the decision right out of our hands. After chugging away 24 hours per day for almost four years it finally gave up in a big way. We don't want to offend anyone, but we are not going to replace it. It simply costs too much and we can't afford it, considering the use it was getting.

To compensate for the missing download, we will be producing disks with all the programs on them for each three issues next year. Naturally, they will be made available at a much reduced price. What that means is that in March you will be able to get the November, January and March programs on a disk, and again in September you will get the May, July and September programs. The disk

will be priced at about \$7, which is probably much less than most of you spent on long distance telephone calls to the download. Not only that, but the disks don't have the transmission problems you often encounter with a modem.

And speaking of disks and prices, take a look at page 39 of this issue. We have reduced the price of our back issues to \$18 per set - and the price of our yearly disks to \$15. So if you haven't yet filled out with all the issues and disks, you can now do it at a considerable savings. Besides that, since we are in a cost cutting mood, if we move those back issues it will liberate some much needed space for better purposes. If you order 2nd year issues, you will get Issue 8 on a diskette, because we are plumb out of that one for some reason. So don't forget to tell us what machine the disk should be for. We'll photocopy Issue 8 for you if you insist. Do you suppose the Poker program in Issue 8 had anything to do with it being in short supply? Well, it was a big program, and with the disk you don't have to type it in - so there's some good in everything. The disk for Issue 8, by the way, includes all the text for that issue as well as the programs, so you won't miss anything with the disk.

We're getting ready to enjoy a super Northwest summer around here. Hope it's just as nice where you are and we don't have a repeat of last year's hot spell. Enjoy your vacations, and we'll see you all in September.

Irv

Forum BASIC

An Open Forum for Questions and Comments

I have started to convert to an IBM PC/XT compatible with which I am using Smart to write this letter. It's been exciting and a true learning situation going from a Heath H-89A with CP/M to MS DOS. I have a conversion program for CP/M so all my BASIC programs and test files are not lost. There are lots of similarities between the two operating systems, which makes you realize that here are only so many ways to run a computer. The game now is to give more colorful displays, easier to use programs and do it faster than the old systems.

I have been following your section on BASIC commands in detail. I am presently converting a checkbook program from CP/M to GW BASIC. I have a question on the INPUT\$(1) statement used to get input from the keyboard. On my Heath the cursor remains visible and the delete key works ok if you make a mistake. Under GW BASIC the cursor is not visible and if you use the delete key you get some funny characters on the screen, but in reality they were deleted. The author of the program is merely using this arrangement to check each input to make sure it is valid as this is data for the disk files. If the character is good or in range it is printed to the screen and a string is built which is finished by an enter key or return. My question is how I can get around this as the author uses it extensively throughout the program. In the GW BASIC manual it is doing as expected, but would like to see where I am typing and be able to see the characters as they are deleted. . .

Ronald B. Williamson
Lynchburg, VA

GW BASIC handles both INPUT\$(1) and backspace somewhat differently than MBASIC did. I would change all occurrences of INPUT\$(1) to a standard INPUT statement and let it go at that. Also, see Beginning BASIC in this issue for more on input statements.

In Issue 22, I note Donald E. Williams found

line 1000 in NFL88 was giving "Subscript out of range" sometimes. Perhaps I can shed a bit of dim light on this problem. If I use a TAB instruction such as PRINT TAB(13), and I inadvertently type TAB (space) (13), I am sure to be informed that some subscript is out of range. I wasted an afternoon recently trying to find an offending subscript before I accidentally found the misleading remark was associated with the error of typing the space after the TAB. Why this happens is beyond me, but it seems to be so.

Next item. When doing a BACKUP, why would one want to write to the source disk? Yet, the Tandy Model III manual states: NOTE: Both source and destination disks must be write-enabled. (See page 67). On the other hand, the Model I Tandy manual suggests it would be a good idea to put a write protect on the source disk. Frankly, I have made backups without ever bothering to read the complete instruction manual for the Model III. I thought I knew how. Now I am worried. Can you explain?

D. B. McRae
Grantsville, UT

The surest way to write blank spaces over your good data is to un-protect your source disk. If you forget which drive is going to which drive, you can wipe out a whole disk that way. I suspect the manual is in error, and I always write protect the source disk to keep from writing on it by mistake.

As you know, there is increasing interest in communication between computers by radio. The use of a radio modem permits extensive networking without tying up telephone circuits.

Radio amateurs have developed X.25 packet networks that reach virtually every city in the nation. The equipment to communicate on these networks is very low in cost.

At the present time, it is necessary to learn the Morse code to become a radio amateur. This is clearly an archaic requirement to those who move data at kilobit and megabit rates, but it is

the law.

To encourage computerists to participate in this exciting hobby, our organization is offering a free copy of an extraordinary shareware program called "Super Morse" written by Lee Murrah. Lee's program has helped thousands to learn the code and become radio amateurs. We would certainly appreciate any publicity you could provide for our organization and this program.

If any of your many readers would like a copy of Super Morse, it can be obtained by sending \$3.00, postage and handling, to: The National Amateur Radio Association, 16541 Redmond Way, Suite 232, Redmond, WA 98052, phone (206) 232-2579.

Donald L. Stoner, W6TNS
President, NARA

Being an ex-HAM, I certainly appreciate the problems with the code. Sounds like you have a really exciting thing going here - almost makes me want to jump back into the game.

...Flow.Bas in the Mar/Apr 89 issue seemed a good utility for helping debug a program, so I played with it using one of my larger programs as input. In the "ON X GOTO" statement it picked up only the first line number and said it was a HARD GOTO. I considered it a conditional GOTO and wanted to see all the line numbers identified so I took the liberty of adding a few lines to your program:

```
475 IF RIGHT$(S$,4) = `` ON `` THEN F
= 3
535 IF F = 3 AND RIGHT$(S$,1) = ``,'``
THEN GOSUB 1010
1071 IF PR = 1 AND F1 = 1 AND F = 3 THEN
LPRINT USING F1$;LN;T1$;A;T3$
1072 IF F1 = 1 AND F = 3 THEN PRINT USING
F1$;LN;T1$;A;T3$:A(J) = LN:B(J) =
A:C(J) = 1
```

I had to remark line 1050 to make it work properly. Lines 855 HT = 1 and line 945 IF C(I) > HT THEN HT = C(I):LPRINT were added for aesthetic purposes to produce a blank line between the different conditions. Perhaps others

might be interested in these changes.

Jean M. Hall
San Diego, CA

Yes, I think they would be interested, and I'm wondering how we ever missed that in the original program.

It's a big GO for year 5 of Code-Works! You can help out by renewing early and taking advantage of our great diskette offer. The fourth year diskette will be ready about the 1st of September. Order it now with your renewal and we will send it as soon as the next issue's programs are firmed up. See page 39 for details, and thank you all for another fine year of support!



I'm getting out of touch with reality; think I'll play Dragons and Demons for awhile.

Beginning BASIC

All About Strings - Part 2

Last issue we looked at some of the more fundamental ideas in strings and string manipulation. Mostly, how strings are stored in memory and recalled is of little practical value - but nice to know. This time we will look at more of the commands and functions used in handling strings.

There are two sets of functions which convert back and forth between string and integer values. The first set of these consist of the functions ASC("n") and CHR\$(n). The first of these two, ASC("n"), will return the ASCII value of the string letter or number enclosed in both quotes and parenthesis. If you ask your computer to PRINT ASC("a"), it will return the integer number 97, because 97 is the ASCII value of lower-case a. If A\$="this is a string" and you ask for the ASC(A\$) you will get 116, the ASCII value of the lower-case letter t. ASC() will only return a value for the first letter of a multi-character string.

The opposite member of this team is CHR\$(n). If you ask your computer to PRINT CHR\$(97), it will return a lower-case a. It converts an integer number (97) into its ASCII representation, (a). Where are these useful? In many places. Do you remember that we cannot add or subtract string values? Just as one example of where we can use ASC and CHR\$, how about an upper case converter. If you look at the ASCII table, you will find that the lower-case values for the entire alphabet are exactly 32 higher than the upper-case values. Uppercase A is 65, and 65 plus 32 equal 97, which is the value for lower-case a. Let's demonstrate with a little code:

```
10 A$="this is a string"
20 FOR I = 1 TO LEN(A$)
30 C$ = MID$(A$,I,1)
40 C$ = CHR$(ASC(C$)-32)
```

```
50 MID$(A$,I,1) = C$
60 NEXT I
70 PRINT A$
RUN
THIS IS A STRING
```

We'll get into the MID\$ thing in a little while. For now, let's look at the rest of the code. We already know about LEN(A\$) from last issue. It tells us how long A\$ is without really telling us how long it is. We don't care, as long as we look at each character in A\$. In line 30 we are looking at one character at a time and calling it C\$. Line 40 is where our ASC and CHR\$ come into play. Let's look at the part within the parentheses first. It says that we are taking the ASC of C\$ and subtracting 32 from that number. Taking the ASC of C\$ makes an integer out of the value, so that we can subtract another number (32) from it to get the uppercase letter instead of the lower-case letter. But now that we have the number subtracted, we need to change it back into a string so that C\$ can insert it back into A\$. Look at line 40 again. It says that C\$ now equals the CHR\$ of the integer value of C\$ less 32. The CHR\$ converted the integer back into a string and assigned it to C\$. Remember that we are dealing with only one character at a time here. Line 50 then inserts the converted C\$ back into A\$. When we are all done, we print the converted A\$ and find it has changed from all lower-case to all uppercase. If you have typed in the above sample program, save it, because we will add a few lines and use it again later.

The other "dynamic duo" is the pair, VAL(\$) and STR\$(n). Let's take VAL(\$) first. This function will return the integer value of a string - but - only if the string is a numerical string - and only if the numerical string is not preceded by letters. Picky, what? If A\$="this is a string" and we ask for the VAL(A\$), we will get zero. If A\$="0123",

VAL(A\$) will return 123 in integer form. VAL(\$) will return decimal amounts, as when A\$="12.34" but will not return anything after a comma or semicolon. As we have seen above, it will not return leading zeros either. This one is real handy in making numbers out of dollar amounts, but not when the amount is preceded with a dollar sign!

The other half of this pair, STR\$(n), will change an integer value into a string value. Actually, it will work with decimal amounts too, not just whole integers. If you PRINT STR\$(12.34) you will get 12.34 with a space preceding the 12.34. The space is added by STR\$ for the sign of the number, and since our number was positive, the plus sign is, by convention, left out.

Now let's take a look at two useful functions, LEFT\$ and RIGHT\$. These two commands let you pick off n characters from the left or right end of a string, respectively. Here's our little program that we used earlier, with a few added lines to demonstrate how LEFT\$ and RIGHT\$ work:

```
10 A$="this is a string"
20 FOR I = 1 TO LEN(A$)
30 C$ = MID$(A$,I,1)
40 C$ = CHR$(ASC(C$)-32)
50 MID$(A$,I,1)=C$
60 NEXT I
70 PRINT A$
80 '
90 '
100 B$ = MID$(A$,6,3)
110 C$ = LEFT$(A$,5)
120 D$ = RIGHT$(A$,8)
130 E$ = B$+C$+D$+"?"
140 PRINT E$
RUN
THIS IS A STRING
IS THIS A STRING?
```

Lines 100 through 130 demonstrate how we can take parts of a string and change their positions in the original string. The general form of LEFT\$ is: LEFT\$(\$,n) where \$ is the string we

are dealing with and n is the number of characters we want to pick off. If n is greater than the length of the string, LEFT\$ will return the entire string. The same general form holds true for RIGHT\$, only this time we are taking n characters off the right end of the string.

The sample program above uses MID\$ in two different ways. It is used on both sides of the equal sign. When it is used on the left side of the equal sign, MID\$ is a statement or command. MID\$, as you have probably already guessed, picks parts out of the middle of a string. The form for MID\$ used as a statement is MID\$(\$,n,m) where \$ is the string we are working with, n is the character position we want to start with and m is how many characters we want to pick out of the string. Can you now begin to see how important INSTR (from last issue) is? INSTR will tell us where a certain character falls within a string so that we can plug that number into the MID\$ statement. Line 100 of our sample program takes the MID\$ of A\$, starting at position 6 and taking 3 characters. In our case, it will take the word "is" and the space following it. In line 110, we take the LEFT\$ of A\$ for 5 characters, which would be the word "this" and the space following it. Then, line 120 takes the RIGHT\$ of A\$ for 8 characters, which would include "a string". Now, when we print these in the order shown in line 130, we get "is this a string" but we added a question mark in line 130, so E\$ in line 140 will really print "is this a string?" If m is omitted (how many characters to take), MID\$ will return all the characters to the right of the starting position. In fact, MID\$ can be used in place of both RIGHT\$ and LEFT\$ in many cases. For example, we could ask for MID\$(A\$,1,4) and get "this" off of the left end of our A\$, just like LEFT\$(A\$,4) would have.

MID\$ on the left of the equal sign can be used to replace a portion of a string with another string. We use it that way in line 50 of our sample program, where we are plugging in a new C\$ into A\$ at position I for one character. Remember that in line 40 we changed C\$ from a lower-case character to an uppercase character, and that

now we are going to insert that new uppercase character back into the original A\$. It is important to note that the number of replacement characters cannot exceed the original length of A\$, which means that MID\$ cannot change the length of a string.

The only way to change the length of a string is by concatenation using the plus sign (+). This device "boxcars" pieces of strings together to make a new, longer string. In our sample program, we use concatenation in line 130, where E\$ is a new string made up of pieces of other strings and a literal question mark.

Some earlier versions of BASIC only allow MID\$ on the right of the equal sign. This makes a function out of MID\$ (it's a statement or command when on the left of the equal sign). Look at line 30 of our sample program, where we use MID\$ on the right. It says that we are assigning to C\$ the MID\$ of A\$, starting at position I, for one character. As you can see, the MID\$ function returns characters from the middle of a string. Again, you can pull them off of the right or left end as well.

An interesting string function is STRING\$. The form of this function is: STRING\$(n,c), where n is the number of times to repeat the character c. The character c can be input as a literal, but it must then be in quotes. Otherwise, it can be simply the ASCII number of the character. Here's a sample:

```
10 PRINT STRING$(15,"-")
RUN
```

Ok

We could just as well have said: PRINT STRING\$(15,45) and got the same result because 45 is the ASCII value of the dash. This one comes in handy for making borders and long lines of dashes or underscores.

In some newer versions of BASIC you also have the command SPACE\$, which prints a

number of spaces. But STRING\$(n,32) will do the same thing, where n is the number of spaces you want and 32 is ASCII for the space. The form of SPACE\$ is: SPACE\$(n), where n is the number of spaces you wish to have.

Also in some newer BASICs you will find the command SPC(n) which does the same as the previous two commands - it prints n number of spaces. There is, however, one thing about SPC(n) that is different: it uses no string space.

INPUT\$(n) is another function not found in older versions of BASIC. This function is used to get a string of n characters from the keyboard. Unlike the regular INPUT statement, this one does not echo (print) the characters you enter on the screen. This, of course, makes it a natural for entering passwords. Here is another way it can be used:

```
10 PRINT "Press any key";
20 XX$ = INPUT$(1)
```

XX\$ is a "throw-away" variable which we don't care about. Line 20 will stop execution of the program, waiting for any character to be input from the keyboard after which execution will continue.

INKEY\$ is a function rather similar to INPUT\$, but it will return the last character you type at the keyboard or a null string if no character has been typed. INKEY\$ does not wait for you to enter anything. It is usually used like this:

```
10 A$=INKEY$:IF A$="" THEN 10
```

This forces INKEY\$ to wait for an input other than a null string. Further, the key you press is contained in A\$. This is usually found when yes/no questions need to be answered in a program. The next line of the program would probably be: 20 IF A\$="Y" OR A\$="y" THEN ... ELSE.

This pretty well covers strings. In the next installment, we will look at a host of ways these string statements and functions can be used in everyday programs.

Tracker.Bas

A Mutual Fund Tracking Program

Jean M. Hall, San Diego, California. Jean sent us this program a couple of years ago and we have been waiting for the right time to fit it in. It's long, and we figured that July would be the best time for it. We wrote the article to accompany the program.

The Mutual Fund Tracker program, Tracker.Bas, concept is to enable you to follow the progress of any Mutual Fund and calculate the moving average for that fund with the objective of letting you know when you might want to move in or out of a Money Market Fund.

Files can be set up for weekly, daily or monthly tracking. Weekly (or daily or monthly) closing prices are entered from the keyboard. There are 39 entries for each fund which can be entered as the weeks go by, or previous closing prices can be obtained at your local public library. Once the 39 entries have been made, the program updates by dropping off the oldest entry and adding the new one and recalculating the moving average. When the fund has been tracked for a year or more you will have reached the optimum moving average.

The program prints four reports: An update report showing closing rate, moving average and percent change from previous; a listing showing the date, closing rate, total and average; a chart showing the comparison between the rate and the moving average; and a listing of all funds on a specific file.

Those menu items which say "print" will require a printer on and on-line. Those items which say "display" will print to the screen. Be aware of the distinction between a "file" and a "fund." A file may contain several funds. The author made no provision for lower-case entries, so **run with**

your caps key locked on. Following is a brief description of each menu item, in order:

Option A: Asks for the drive and file name and allows you to set up a new file. However, if you give a name of a file which already exists, it will open that one.

Option B: Allows you to add a new fund to the file or to reuse an existing fund which you are no longer interested in. When setting up a new fund, the program asks how many entries you wish to make and will return to the main menu when that number has been reached. You may have the data for only one or 10, or all 39 records.

Option C: This option is how you will update your files with weekly (daily or monthly, if you choose) rates. You are asked for the fund name then the date to be entered in YYMMDD format. Here is the secret - if your system has a date and it is the date you wish to use for the entry, type S and the date is automatically entered. Enter the closing rate. If you have not updated your file for a few weeks you may want to make more entries. Then you are asked if you want to update another fund. A NO answer to this brings you back to the main menu. The program subtracts the oldest closing rate from the total and adds the current closing rate, then recalculates the moving average. A report is printed as you enter the closing rates for each fund. The report includes date, fund name, close rate, percent change from previous close, moving average,

percent change from previous moving average and percent difference current close from current average.

Option D: Allows you to print a listing for all funds on the file or a specific fund. The listing includes record number, date, closing rate, total and average.

Option E: Will let you print a chart for all funds on the file or a specific fund. The chart is a graphic which includes date, closing rate, lower, middle and upper limits and a graphic representation of rate, moving average and where they cross.

Option F: Will display the fund listing on the screen in two sections. The display will show record number, date, closing rate, total and average. The reason for two sections is that only one section will fit the screen at one time.

Option G: Will display the fund chart on the screen using every third entry (because of space limits).

Option H: Will print a listing of all fund names in the file, including start record number, end

record number and number of records.

Option I: Allows you to correct an erroneous entry. The program asks for the record number to be fixed and displays the date and amount, then asks for the new date and amount.

Option J: Will recalculate a fund which will make the closing rate and moving average to be the same for the first record of that fund on file.

Option K: Ends the run.

Because of the complexity and length of the program we have made no attempt to convert to various machines. It is suggested that the program be run at least on a machine with 24 line video displays. In spite of all that, those with less than 24 line screens can use the program and ignore the screen display sections and send their output directly to the printer, since the screen only repeats what will appear on the printer in all cases.

Some of the areas that will need attention in modification are the multi-lettered variables, the random file opening statements and the screen formatting for screens less than 24 lines.

Tracker.Bas for MS DOS machines

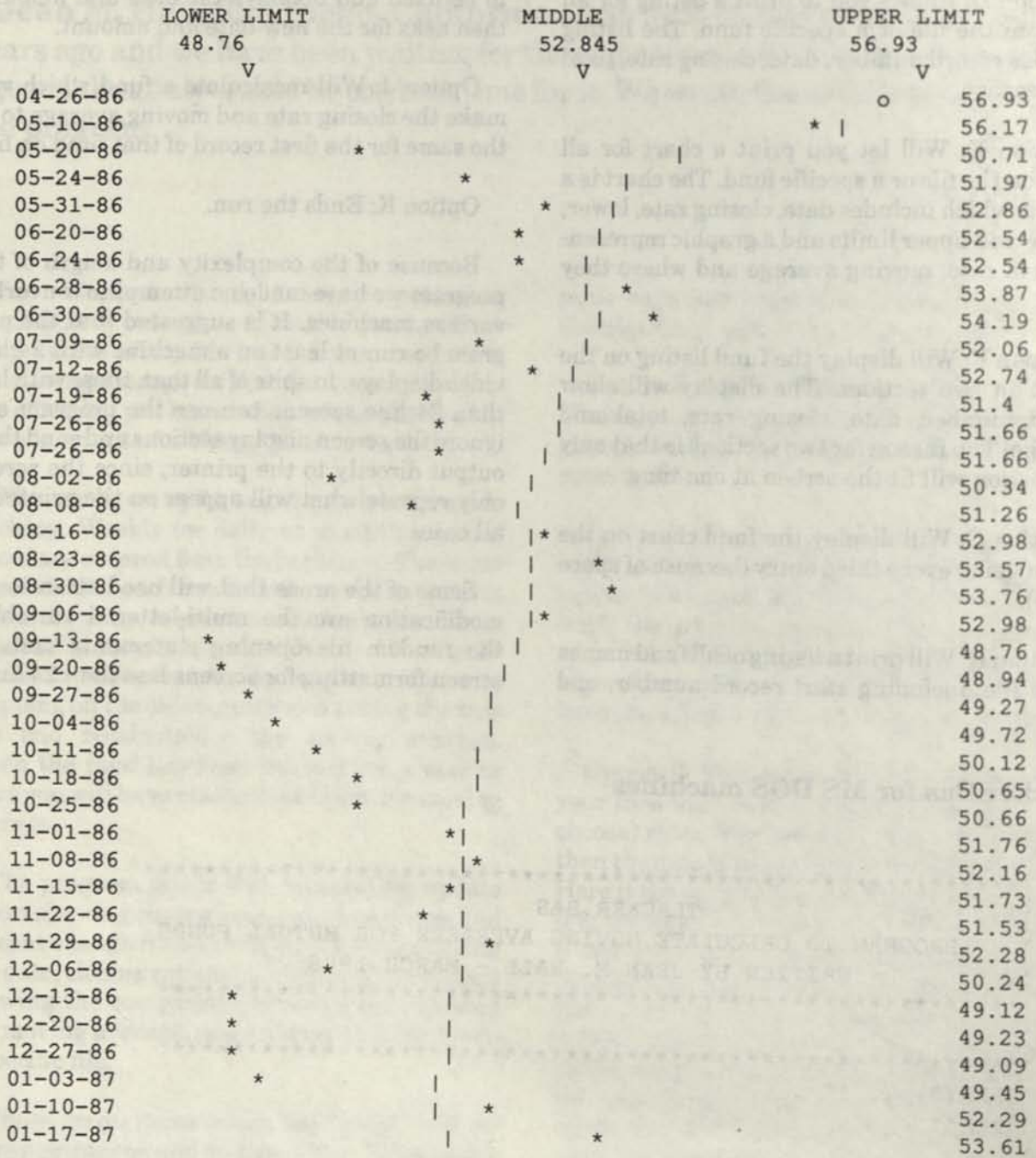
```
100 *****
110 \           TRACKER.BAS
120 \   PROGRAM TO CALCULATE MOVING AVERAGES FOR MUTUAL FUNDS
130 \           WRITTEN BY JEAN M. HALL - MARCH 1986
140 *****
150 \
160 *****
170 'ARRAYS
180 '*****
190 \
200 DIM C$(38)
210 FOR A = 1 TO 38
220   C$(A)=STRING$(36,32)
230 NEXT A
```

Listing continues on page 11

Tracker.Bas

MAGELLAN

RATE = *
 MOVING AVERAGE = |
 WHERE THEY CROSS = o



This is an example of the chart output using Option E for a hypothetical fund, showing the moving average.

```

240 DIM A$(39), V(39), Y(39), Z$(39)
250 *****
260 `DEFINITIONS
270 *****
280 `RN#           = RECORD NUMBER IN FILE #1
290 `RN2#          = RECORD NUMBER IN FILE #2
300 `CRN#          = RECORD NUMBER IN FUND
310 `LST#          = LAST RECORDS NO ON FILE #1
320 `LST2#         = LAST RECORDS ON FILE #2
330 *****
340 FRST#=0        `FIRST TIME FILE IS OPENED
350 *****
360 `CONSTANTS
370 *****
380 HLST#=0        `HOLD LAST RECORD NUMBER
390 HCD#=0         `HOLD NAME SUBSCRIPT
400 HTT#=0         `HOLD TOTAL
410 NF#=0         `NEW FILE FLAG
420 HCRN#=0       `HOLD NO OF INPUT ENTRIES FOR FUND
430 CRN#=0        `NO ENTRIES FOR FUND
440 SRN#=0        `BEGINNING RECORD NO FOR FUND
450 ERN#=0        `END RECORD NO FOR FUND
460 EN1#=0        `END OF FILE #1 FLAG
470 EN2#=0        `END OF FILE #2 FLAG
480 UTL#=0        `UNABLE TO LOCATE FLAG
490 CFL#=0        `CHART FLAG
500 HDG#=0        `HEADING FLAG
510 CF#=0         `CLEAR FUND FLAG
520 S$=STRING$(1,32) `SPACE FOR CLEARING
530 CD$=STRING$(6,32) `CURRENT SYSTEM DATE - YYMMDD
540 MID$(CD$,1,2)=RIGHT$(DATE$,2)
550 MID$(CD$,3,2)=LEFT$(DATE$,2)
560 MID$(CD$,5,2)=MID$(DATE$,4,2)
570 *****
580 `SELECT OPTION
590 *****
600 CLS
610 PRINT `*****`
620 PRINT `*` OPTIONS           A = SET UP NEW FILE           `*`
630 PRINT `*`                   B = SET UP NEW FUND             `*`
640 PRINT `*`                   C = UPDATE FUND WITH CLOSING RATE `*`
650 PRINT `*`                   D = PRINT FUND                   `*`
660 PRINT `*`                   E = PRINT CHART                   `*`
670 PRINT `*`                   F = DISPLAY FUND                   `*`
680 PRINT `*`                   G = DISPLAY CHART                   `*`
690 PRINT `*`                   H = PRINT LISTING OF ALL FUND NAMES `*`
700 PRINT `*`                   I = FIX ERROR IN INPUT             `*`
710 PRINT `*`                   J = RECALCULATE AN EXISTING FUND    `*`

```

```

720 PRINT '*'
730 PRINT '*****'
740 PRINT
750 INPUT 'ENTER OPTION'; O$
760 IF O$ < 'A' OR O$ > 'K' THEN PRINT: PRINT 'INVALID OPTION': PRINT:
    INPUT 'PRESS ANY KEY TO RETURN TO THE MENU'; K$: GOTO 380
770 IF FRST#=0 THEN FRST#=1: GOSUB 5860
780 IF O$='A' THEN HLST#=1: NF#=1: GOTO 600
790 IF O$='B' THEN GOTO 930
800 IF O$='C' THEN GOTO 1860
810 IF O$='D' THEN GOTO 3100
820 IF O$='E' THEN GOTO 3620
830 IF O$='F' THEN 4150
840 IF O$='G' THEN 4540
850 IF O$='H' THEN GOTO 5100
860 IF O$='I' THEN GOTO 5250
870 IF O$='J' THEN GOTO 5650
880 IF O$='K' THEN FRST#=0: PRINT 'END OF RUN': END
890 END
900 '*****'
910 'OPTION B - SET UP NEW FUND'
920 '*****'
930 PRINT
940 PRINT '*****'
950 PRINT 'AT THIS POINT YOU MAY SELECT ONE OF TWO OPTIONS'
960 PRINT 'CLEAR AND REUSE AN EXISTING FUND - Y'
970 PRINT 'ADD A NEW FUND TO THE FILE - N'
980 PRINT 'ENTER Y OR N'
990 PRINT '*****'
1000 PRINT
1010 INPUT K$
1020 IF K$='Y' THEN CF#=1: GOTO 1700
1030 IF K$='N' THEN GOTO 1050
1040 GOTO 930
1050 GOSUB 6470
1060 GOSUB 6570
1070 GOSUB 1100
1080 GOTO 1560
1090 '*****'
1100 PRINT
1110 PRINT '*****'
1120 PRINT 'NAME OF NEW FUND'
1130 PRINT 'UP TO 10 CHARACTERS WITH NO SPACES'
1140 PRINT '*****'
1150 PRINT
1160 INPUT NAM$
1170 PRINT
1180 INPUT 'NO OF ENTRIES'; HCRN#

```

```

1190 SRN#=RN#
1200 CRN#=CRN#+1
1210 IF CRN#=40 THEN ERN#=RN#-1: GOTO 1460
1220 IF CRN#>HCRN# THEN GOTO 1390
1230 INPUT ``DATE - YYMMDD``; HDT$
1240 IF HDT$ < ``!`` THEN GOTO 1230
1250 IF HDT$=``S`` THEN HDT$=CD$: PRINT HDT$
1260 IF MID$(HDT$,3,2) > ``12`` THEN PRINT ``RE-ENTER DATE ``: GOTO 1230
1270 IF RIGHT$(HDT$,2) > ``31`` THEN PRINT ``RE-ENTER DATE ``: GOTO 1230
1280 INPUT ``CLOSE AMOUNT``; HCL#
1290 LSET DT$=HDT$
1300 LSET CL$=STR$(HCL#)
1310 HTT#=HTT#+HCL#
1320 LSET TT$=STR$(HTT#)
1330 HAV#=HTT#/CRN#
1340 LSET AV$=STR$(HAV#)
1350 LSET DC$=STR$(CRN#)
1360 PUT #1, RN#
1370 RN#=RN#+1
1380 GOTO 1200
1390 IF CRN#=40 THEN ERN#=RN#-1: GOTO 1460
1400 LSET AL$=SPACE$(32)
1410 LSET DC$=STR$(CRN#)
1420 PUT #1, RN#
1430 CRN#=CRN#+1
1440 RN#=RN#+1
1450 GOTO 1390
1460 IF CF#=1 THEN CF#=0: GOTO 1520
1470 HRN#=RN#-1
1480 RN#=1
1490 LSET LST$=STR$(HRN#)
1500 LSET RR$=SPACE$(30)
1510 PUT #1, RN#
1520 CLOSE #1
1530 GOSUB 6270
1540 RETURN
1550 *****
1560 GOSUB 6660
1570 PRINT
1580 PRINT *****
1590 PRINT ``DO YOU WISH A LISTING OF ALL FUND NAMES ON THIS FILE?``
1600 PRINT *****
1610 PRINT
1620 INPUT ``ENTER Y OR N``; K$
1630 IF K$=``Y`` GOTO 5100
1640 IF K$=``N`` THEN GOTO 1660
1650 GOTO 1580
1660 GOTO 380

```

```

1670 *****
1680 'CLEAR AND REUSE AN EXISTING FUND
1690 *****
1700 INPUT 'ENTER NAME OF EXISTING FUND'; NAM$
1710 GOSUB 6060
1720 IF UTL#=1 THEN GOTO 380
1730 GOSUB 6470
1740 RN#=SRN#
1750 GOSUB 1100
1760 RN2#=HRN2#
1770 GET #2, RN2#
1780 LSET NM$=NAM$
1790 LSET NR$=STR$(HCRN#)
1800 PUT #2, RN2#
1810 CLOSE #2
1820 GOTO 380
1830 *****
1840 'OPTION C - UPDATE FUND WITH CLOSING RATE
1850 *****
1860 GOSUB 5990
1870 IF UTL#=1 THEN UTL#=0: GOTO 2360
1880 GOSUB 6470
1890 IF HNR#<39 THEN GOTO 2450
1900 RN#=SRN#
1910 GET #1, RN#
1920 IF EOF(1) THEN ER#=1: GOTO 3000
1930 HCLS#=VAL(CL$)
1940 FOR A = 1 TO 37
1950 RN#=RN#+1
1960 IF RN#>ENR# THEN ER#=2: GOTO 3000
1970 GET #1, RN#
1980 IF EOF(1) THEN ER#=3: GOTO 3000
1990 C$(A)=AL1$
2000 NEXT A
2010 RN#=RN#+1
2020 GET #1, RN#
2030 IF EOF(1) THEN ER#=4: GOTO 3000
2040 HTT#=VAL(TT$)
2050 HLCL#=VAL(CL$)
2060 HLAV#=VAL(AV$)
2070 C$(38)=AL1$
2080 RN#=SRN#
2090 FOR A = 1 TO 38
2100 LSET AL1$=C$(A)
2110 PUT #1, RN#
2120 RN#=RN#+1
2130 NEXT A
2140 INPUT 'DATE - YYMMDD '; HDT$

```

```

2150 IF HDT$ < '! ' THEN GOTO 2290
2160 IF HDT$=' 'S' THEN HDT$=CD$: PRINT HDT$
2170 IF MID$(HDT$,3,2) > '12' THEN PRINT 'RE-ENTER DATE ': GOTO 2140
2180 IF RIGHT$(HDT$,2) > '31' THEN PRINT 'RE-ENTER DATE ': GOTO 2140
2190 INPUT 'CLOSE AMOUNT'; HCL#
2200 IF HCL# = 0 THEN GOTO 2190
2210 LSET DT$=HDT$
2220 LSET CL$=STR$(HCL#)
2230 HTT#=HTT#-HCLS#+HCL#
2240 LSET TT$=STR$(HTT#)
2250 HAV#=HTT#/39
2260 LSET AV$=STR$(HAV#)
2270 PUT #1, RN#
2280 GOSUB 2900
2290 PRINT
2300 INPUT 'MORE ENTRIES? - Y OR N'; K$
2310 PRINT
2320 IF K$=' 'Y' THEN GOTO 1900
2330 IF K$=' 'N' THEN GOTO 2350
2340 GOTO 2300
2350 CLOSE #1
2360 PRINT
2370 INPUT 'DO YOU WISH TO UPDATE ANOTHER FUND? - ENTER Y OR N'; K$
2380 IF K$=' 'N' THEN PRINT CHR$(12): GOTO 2410
2390 IF K$=' 'Y' THEN GOTO 1860
2400 GOTO 2370
2410 GOTO 380
2420 '*****
2430 'UPDATE INCOMPLETE FUND
2440 '*****
2450 CRN#=HNR#
2460 RN#=SRN#+HNR#-1
2470 GET #1, RN#
2480 HTT#=VAL(TT$)
2490 HLCL#=VAL(CL$)
2500 HLAV#=VAL(AV$)
2510 RN#=RN#+1
2520 CRN#=CRN#+1
2530 IF CRN#=40 THEN PRINT 'THIS FUND HAS NOW BEEN COMPLETED WITH 39
RECORDS':GOTO 2740
2540 PRINT 'WHEN NO MORE ENTRIES TYPE - N'
2550 INPUT 'DATE - YYMMDD'; HDT$
2560 IF HDT$=' 'N' THEN GOTO 2740
2570 IF HDT$ < '! ' THEN GOTO 2740
2580 IF HDT$=' 'S' THEN HDT$=CD$: PRINT HDT$
2590 IF MID$(HDT$,3,2) > '12' THEN PRINT 'RE-ENTER DATE ': GOTO 2550
2600 IF RIGHT$(HDT$,2) > '31' THEN PRINT 'RE-ENTER DATE ': GOTO 2550
2610 INPUT 'CLOSE AMOUNT'; HCL#

```

```

2620 IF HCL#=0 THEN GOTO 2610
2630 LSET DT$=HDT$
2640 LSET CL$=STR$(HCL#)
2650 HTT#=HTT#+HCL#
2660 LSET TT$=STR$(HTT#)
2670 HAV#=HTT#/CRN#
2680 LSET AV$=STR$(HAV#)
2690 LSET DC$=STR$(CRN#)
2700 PUT #1, RN#
2710 HNR#=HNR#+1
2720 GOSUB 2900
2730 GOTO 2510
2740 CLOSE #1
2750 GOSUB 6270
2760 RN2#=HRN2#
2770 GET #2, RN2#
2780 LSET NR$=STR$(HNR#)
2790 PUT #2, RN2#
2800 CLOSE #2
2810 PRINT
2820 INPUT ``DO YOU WISH TO UPDATE ANOTHER FUND? - ENTER Y OR N``; K$
2830 IF K$='`N`` THEN PRINT CHR$(12): GOTO 2860
2840 IF K$='`Y`` THEN GOTO 1860
2850 GOTO 2810
2860 GOTO 380
2870 `*****
2880 `PRINT UPDATE
2890 `*****
2900 IF HDG#=0 THEN HDG#=1:LPRINT TAB(35) ``UPDATES``:LPRINT :LPRINT ``DA
TE      FUND NAME      CLOSE RATE  %CHG FROM  MOVING AVG  %CHG FROM
%DIFF CR CLS``: LPRINT ``                                PREV CLS
PREV MOV   FROM CR AVG``
2910 PRINT
2920 CLCG#=((HCL#-HLCL#)*100)/HLCL#
2930 AVCG#=((HAV#-HLAV#)*100)/HLAV#
2940 CHG#=((HCL#-HAV#)*100)/HAV#
2950 LPRINT USING ``\          \          \####.##      ###.##
###.##      ###.##      ###.##``;DT$,NAM$,HCL#,CLCG#,HAV#,AVCG#,
CHG#
2960 RETURN
2970 `*****
2980 `ERROR MESSAGE
2990 `*****
3000 PRINT
3010 PRINT ``ERROR `` ER#
3020 PRINT ``THERE IS AN ERROR IN THE STARTING OR ENDING RECORD
NUMBERS``
3030 PRINT ``FOR FUND `` HNMS `` IN THE NAME FILE (.002)``

```



```

3040 PRINT ``THIS RUN IS ABORTED``
3050 CLOSE #1
3060 END
3070 `*****`
3080 `OPTION D - PRINT A SPECIFIC FUND`
3090 `*****`
3100 PRINT
3110 PRINT `*****`
3120 PRINT ``DO YOU WISH TO PRINT ALL FUNDS?``
3130 PRINT `*****`
3140 PRINT
3150 INPUT ``ENTER Y OR N``; K$
3160 IF K$=`Y` THEN GOTO 3300
3170 IF K$=`N` THEN GOTO 3190
3180 GOTO 3110
3190 GOSUB 5990
3200 IF UTL#=1 THEN GOTO 380
3210 GOSUB 6470
3220 RN#=-SRN#
3230 HLR#=-HNR#+SRN#-1
3240 GOSUB 3450
3250 CLOSE #1
3260 GOTO 380
3270 `*****`
3280 `PRINT ALL FUNDS`
3290 `*****`
3300 GOSUB 6270
3310 GOSUB 6470
3320 RN2#=1
3330 GET #2, RN2#
3340 HLST2#=VAL(LST2$)
3350 GOSUB 6360
3360 IF EN2#=1 THEN GOTO 3410
3370 IF RN2#>HLST2# THEN GOTO 3410
3380 IF CFL#=0 THEN GOSUB 3450
3390 IF CFL#=1 THEN GOSUB 3750
3400 GOTO 3350
3410 CLOSE #1: CLOSE #2: GOTO 380
3420 `*****`
3430 `PRINT RECORD`
3440 `*****`
3450 GET #1, RN#
3460 LPRINT HNM$
3470 LPRINT
3480 LPRINT ``RECORD NO      DATE      CLOSE RATE      TOTAL
      AVERAGE``
3490 CNT#=1
3500 RDT$=MID$(AL1$,3,2)+``-``+MID$(AL1$,5,2)+``-``+LEFT$(AL1$,2)

```

```

3510 LPRINT CNT#,RDT$,CL$,TT$,AV$
3520 CNT#=CNT#+1
3530 RN#=RN#+1
3540 IF RN#>HLR# THEN LPRINT CHR$(27);'G': LPRINT S$,S$,CL$,S$,AV$:
      LPRINT CHR$(27);'H': GOTO 3570
3550 GET #1, RN#
3560 GOTO 3500
3570 LPRINT CHR$(12)
3580 RETURN
3590 \*****
3600 \OPTION E - PRINT CHART
3610 \*****
3620 PRINT
3630 PRINT \*****
3640 PRINT \DO YOU WISH TO PRINT ALL FUND CHARTS?'
3650 PRINT \*****
3660 PRINT
3670 INPUT \ENTER Y OR N'; K$
3680 IF K$='N' THEN GOSUB 3710: CLOSE #1: GOTO 380
3690 IF K$='Y' THEN CFL#=1: GOTO 3300
3700 GOTO 3630
3710 GOSUB 5990
3720 IF UTL#=1 THEN GOTO 380
3730 GOSUB 6470
3740 RN#=SRN#
3750 LV=9999.99
3760 HV=0
3770 FOR X = 1 TO HNR#
3780   W = X
3790   GET #1, RN#
3800   A$(W)=DT$
3810   V(W)=VAL(CL$)
3820   Y(W)=VAL(AV$)
3830   Z$(W)=CL$
3840   IF V(W)<LV THEN LV=V(W)
3850   IF V(W)>HV THEN HV=V(W)
3860   IF Y(W)<LV THEN LV=Y(W)
3870   IF Y(W)>HV THEN HV=Y(W)
3880   RN#=RN#+1
3890 NEXT X
3900 LPRINT HNM$,,, \           RATE = '*'
3910 LPRINT ,, , \           MOVING AVERAGE = |'
3920 LPRINT ,, , \           WHERE THEY CROSS = o'
3930 LPRINT
3940 LPRINT
3950 LPRINT TAB(10)'LOWER LIMIT';TAB(37)'MIDDLE';TAB(60)'UPPER LIMIT'
3960 LPRINT TAB(10) LV;TAB(37) (LV+HV)/2;TAB(60) HV
3970 LPRINT TAB(15)'V';TAB(40)'V';TAB(65)'V'

```

```

3980 FOR X = 1 TO HNR#
3990   LPRINT MID$(A$(X), 3, 2); "'-'; RIGHT$(A$(X), 2); "'-'; LEFT$(A$(X), 2);
4000   P#=INT(((V(X)-LV)/(HV-LV))*50)+15
4010   Q#=INT(((Y(X)-LV)/(HV-LV))*50)+15
4020   IF P#=Q# THEN GOTO 4080
4030   IF Y(X) < V(X) THEN GOTO 4060
4040   LPRINT TAB(INT(((V(X)-LV)/(HV-LV))*50)+15)''*''; TAB(INT(((Y(X)-
      LV)/(HV-LV))*50)+15)''|''; TAB(70) Z$(X)
4050   GOTO 4090
4060   LPRINT TAB(INT(((Y(X)-LV)/(HV-LV))*50)+15)''|''; TAB(INT(((V(X)-
      LV)/(HV-LV))*50)+15)''*''; TAB(70) Z$(X)
4070   GOTO 4090
4080   LPRINT TAB(INT(((Y(X)-LV)/(HV-LV))*50)+15)''o''; TAB(70) Z$(X)
4090 NEXT X
4100 LPRINT CHR$(12)
4110 RETURN
4120 '*****
4130 'OPTION F - DISPLAY A FUND
4140 '*****
4150 GOSUB 5990
4160 IF UTL#=1 THEN GOTO 380
4170 GOSUB 6470
4180 RN#=SRN#
4190 HLR#=HNR#+SRN#-1
4200 GOSUB 4260
4210 CLOSE #1
4220 GOTO 380
4230 '*****
4240 'PRINT RECORD
4250 '*****
4260 GET #1, RN#
4270 CLS
4280 PRINT HNM$
4290 PRINT ``RECORD NO      DATE          CLOSE RATE      TOTAL
      AVERAGE``
4300 PRINT
4310 CNT#=1
4320 RDT$=MID$(AL1$, 3, 2)+"'-'+MID$(AL1$, 5, 2)+"'-'+LEFT$(AL1$, 2)
4330 PRINT CNT#, RDT$, CL$, TT$, AV$
4340 CNT#=CNT#+1
4350 IF CNT#=21 THEN GOSUB 4450
4360 RN#=RN#+1
4370 IF RN#>HLR# THEN PRINT S$, S$, CL$, S$, AV$: GOTO 4400
4380 GET #1, RN#
4390 GOTO 4320
4400 INPUT ``PRESS ANY CHARACTER TO RETURN TO THE MENU``; K$
4410 RETURN
4420 '*****

```

```

4430 'GET SECOND HALF
4440 '*****
4450 INPUT 'PRESS ANY CHARACTER TO DISPLAY SECOND HALF OF FUND'; K$
4460 CLS
4470 PRINT HNM$
4480 PRINT 'RECORD NO      DATE      CLOSE RATE      TOTAL
      AVERAGE'
4490 PRINT
4500 RETURN
4510 '*****
4520 'OPTION G - DISPLAY A CHART
4530 '*****
4540 GOSUB 4570
4550 CLOSE #1
4560 GOTO 380
4570 GOSUB 5990
4580 IF UTL#=1 THEN GOTO 380
4590 GOSUB 6470
4600 RN#=SRN#
4610 LV=9999.99
4620 HV=0
4630 CNT#=1
4640 FOR X = 1 TO HNR#/3+1
4650   W = X
4660   GET #1, RN#
4670   A$(W)=DT$
4680   V(W)=VAL(CL$)
4690   Y(W)=VAL(AV$)
4700   Z$(W)=CL$
4710   IF V(W)<LV THEN LV=V(W)
4720   IF V(W)>HV THEN HV=V(W)
4730   IF Y(W)<LV THEN LV=Y(W)
4740   IF Y(W)>HV THEN HV=Y(W)
4750   RN#=RN#+3
4760   CNT#=CNT#+3
4770   IF CNT#>HNR# THEN RN#=RN#+(HNR#-CNT#)
4780 NEXT X
4790 CLS
4800 GOSUB 5000
4810 FOR X = 1 TO HNR#/3+1
4820   PRINT MID$(A$(X),3,2);'-'-';RIGHT$(A$(X),2);'-'-';LEFT$(A$(X),2);
4830   IF HV=LV THEN PRINT: PRINT 'UNABLE TO PRINT CHART - HIGH
      VALUES AND LOW VALUES ARE THE SAME': GOTO 4940
4840   P#=INT(((V(X)-LV)/(HV-LV))*50)+15
4850   Q#=INT(((Y(X)-LV)/(HV-LV))*50)+15
4860   IF P#=Q# THEN GOTO 4920
4870   IF Y(X) < V(X) THEN GOTO 4900
4880   PRINT TAB(INT(((V(X)-LV)/(HV-LV))*50)+15)'*'*';TAB(INT(((Y(X)-

```

```

      LV) / (HV-LV) ) * 50) + 15) ' ' | ' ' ; TAB(70) Z$(X)
4890 GOTO 4930
4900 PRINT TAB(INT(((Y(X)-LV) / (HV-LV) ) * 50) + 15) ' ' | ' ' ; TAB(INT(((V(X)-
      LV) / (HV-LV) ) * 50) + 15) ' ' * ' ' ; TAB(70) Z$(X)
4910 GOTO 4930
4920 PRINT TAB(INT(((Y(X)-LV) / (HV-LV) ) * 50) + 15) ' ' o ' ' ; TAB(70) Z$(X)
4930 NEXT X
4940 PRINT
4950 INPUT ``PRESS ANY KEY TO RETURN TO THE MENU`` ; K$
4960 RETURN
4970 *****
4980 `CREATE HEADING
4990 *****
5000 PRINT HNM$, , , ``          RATE = * ``
5010 PRINT , , , ``          MOVING AVERAGE = | ``
5020 PRINT , , , ``          WHERE THEY CROSS = o ``
5030 PRINT TAB(10) ``LOWER LIMIT`` ; TAB(37) ``MIDDLE`` ; TAB(60) ``UPPER LIMIT``
5040 PRINT TAB(10) LV ; TAB(37) (LV+HV) / 2 ; TAB(60) HV
5050 PRINT TAB(15) ``V`` ; TAB(40) ``V`` ; TAB(65) ``V``
5060 RETURN
5070 *****
5080 `OPTION H - LIST FUND NAMES AND RECORD NOS
5090 *****
5100 GOSUB 6270
5110 RN2#=1
5120 LPRINT ``FUND NAME          START REC          END REC          NO RECORDS``
5130 LPRINT
5140 RN2#=RN2#+1
5150 GET #2, RN2#
5160 IF EOF(2) THEN GOTO 5190
5170 LPRINT NM$, FR$, LR$, NR$
5180 GOTO 5140
5190 CLOSE #2
5200 LPRINT CHR$(12)
5210 GOTO 380
5220 *****
5230 `OPTION I - FIX ERROR IN INPUT
5240 *****
5250 GOSUB 5990
5260 IF UTL#=1 THEN GOTO 380
5270 PRINT
5280 INPUT ``RECORD NUMBER TO BE FIXED`` ; FX#
5290 GOSUB 6470
5300 RN#=SRN#+FX#-1
5310 GET #1, RN#
5320 IF RN#>ENR# THEN PRINT: PRINT ``NO SUCH RECORD
      NUMBER`` : INPUT ``PRESS ANY KEY TO CONTINUE`` ; K$ : CLOSE #1 : GOTO
      380

```

```

5330 PRINT DT$,CL$
5340 HOCL#=VAL(CL$)
5350 INPUT ``NEW DATE``; HDT$
5360 INPUT ``NEW CLOSING AMOUNT``; HCL#
5370 LSET DT$=HDT$
5380 LSET CL$=STR$(HCL#)
5390 HTT#=VAL(TT$)
5400 HHTT#=VAL(TT$)
5410 HTT#=HTT#-HOCL#+HCL#
5420 TTDIF#=HTT#-HHTT#
5430 LSET TT$=STR$(HTT#)
5440 HDC#=VAL(DC$)
5450 HAV#=HTT#/HDC#
5460 LSET AV$=STR$(HAV#)
5470 PUT #1, RN#
5480 RN#=RN#+1
5490 GET #1, RN#
5500 IF EOF(1) THEN GOTO 5600
5510 IF RN#>HNR#+SRN#-1 THEN GOTO 5600
5520 HCL#=VAL(CL$)
5530 HDC#=VAL(DC$)
5540 HTT#=VAL(TT$)+TTDIF#
5550 HAV#=HTT#/HDC#
5560 LSET TT$=STR$(HTT#)
5570 LSET AV$=STR$(HAV#)
5580 PUT #1, RN#
5590 GOTO 5480
5600 CLOSE #1
5610 GOTO 380
5620 `*****
5630 `OPTION J - RECALCULATE A COMPLETE FUND
5640 `*****
5650 GOSUB 5990
5660 IF UTL#=1 THEN GOTO 380
5670 GOSUB 6470
5680 RN#=SRN#
5690 CRN#=CRN#+1
5700 IF CRN#=-40 THEN GOTO 5810
5710 GET #1, RN#
5720 HCL#=VAL(CL$)
5730 HTT#=HTT#+HCL#
5740 LSET TT$=STR$(HTT#)
5750 HAV#=HTT#/CRN#
5760 LSET AV$=STR$(HAV#)
5770 LSET DC$=STR$(CRN#)
5780 PUT #1, RN#
5790 RN#=RN#+1
5800 GOTO 5690

```

Rcard1.Bas

```
5810 CLOSE #1
5820 GOTO 380
5830 *****
5840 `GET FILE NAME
5850 *****
5860 PRINT
5870 PRINT `*****
5880 PRINT `ENTER INPUT DRIVE AND FILE NAME - EXAMPLE B:WKLYAVG`
5890 PRINT `EXTENSION WILL BE SUPPLIED BY PROGRAM`
5900 PRINT `*****
5910 PRINT
5920 INPUT F$
5930 FL1$=F$+`'.001"
5940 FL2$=F$+`'.002"
5950 RETURN
5960 *****
5970 `GET FUND NAME
5980 *****
5990 PRINT
6000 PRINT `*****
6010 PRINT `NAME OF FUND
6020 PRINT `UP TO 10 CHARACTERS WITH NO SPACES`
6030 PRINT `*****
6040 PRINT
6050 INPUT NAM$
6060 GOSUB 6270
6070 RN2#=RN2#+1
6080 GET #2, RN2#
6090 HLST2#=VAL(LST2$)
6100 RN2#=RN2#+1
6110 IF RN2#>HLST2# THEN PRINT: PRINT `UNABLE TO LOCATE THIS FUND`:
    INPUT `PRESS ANY KEY TO CONTINUE`; K$: CLOSE #2: UTL#=1: GOTO
    6230
6120 GET #2, RN2#
6130 LNM#=INSTR(1,AL2$,S$)
6140 RNM$=LEFT$(AL2$, (LNM#-1))
6150 IF NAM$=RNM$ THEN GOTO 6170
6160 GOTO 6100
6170 SRN#=VAL(FR$)
6180 ENR#=VAL(LR$)
6190 HNR#=VAL(NR$)
6200 HRN2#=RN2#
6210 HNM$=NM$
6220 CLOSE #2
6230 RETURN
6240 *****
6250 `OPEN NAMES FILE
6260 *****
```

```

6270 RN2#=0
6280 OPEN `R`, #2, FL2$, 28
6290 FIELD #2, 10 AS NM$, 6 AS FR$, 6 AS LR$, 6 AS NR$
6300 FIELD #2, 6 AS LST2$, 22 AS RR2$
6310 FIELD #2, 28 AS AL2$
6320 RETURN
6330 `*****
6340 `GET NAME RECORD
6350 `*****
6360 RN2#=RN2#+1
6370 GET #2, RN2#
6380 IF EOF(2) THEN EN2#=1
6390 RN#=VAL(FR$)
6400 HLR#=VAL(NR$)+VAL(FR$)-1
6410 HNR#=VAL(NR$)
6420 HNM$=NM$
6430 RETURN
6440 `*****
6450 `OPEN FUND FILE
6460 `*****
6470 RN#=0
6480 OPEN `R`, #1, FL1$, 36
6490 FIELD #1, 6 AS DT$, 8 AS CL$, 10 AS TT$, 8 AS AV$, 4 AS DC$
6500 FIELD #1, 6 AS LST$, 30 AS RR$
6510 FIELD #1, 32 AS AL$, 4 AS DC$
6520 FIELD #1, 36 AS AL1$
6530 RETURN
6540 `*****
6550 `GET FUND LAST RECORD NUMBER
6560 `*****
6570 IF NF#=1 THEN RN#=HLST#: GOTO 6610
6580 RN#=1
6590 GET #1, RN#
6600 HLST#=VAL(LST$)
6610 RN#=HLST#+1
6620 RETURN
6630 `*****
6640 `WRITE NAMES RECORD
6650 `*****
6660 IF NF#=1 THEN RN2#=2: GET #2, RN2#: GOTO 6710
6670 RN2#=1
6680 GET #2, RN2#
6690 RN2#=VAL(LST2$)
6700 RN2#=RN2#+1
6710 LSET NM$=NAM$
6720 LSET FR$=STR$(SRN#)
6730 LSET LR$=STR$(ERN#)
6740 LSET NR$=STR$(HCRN#)
6750 PUT #2, RN2#
6760 ERN2#=RN2#
6770 RN2#=1
6780 GET #2, RN2#
6790 LSET LST2$=STR$(ERN2#)
6800 LSET RR2$=SPACE$(22)
6810 PUT #2, RN2#
6820 CLOSE #2
6830 RETURN
6840 END `of program

```


Rcard1.Bas

Updates for speed, columns and printer control

Gary T. Prescott, Universal City, Texas. It's been some time since we first published Rcard, and even then we said it was slow and could use some refinements. The author not only did the speed up, but added printing in columns and printer control as well.

I have finally found the way to speed up Rcard.Bas from Issue 8, Nov/Dec 1986. The solution was in line 930 all the time and involves the MID\$= function. When this function is used, string space is not continually consumed and "garbage collection" is not required! I have also included two other modifications. One, which Joe Lybrand asked for in Issue 15, allows for two column formatting, and the other incorporates printer font control. I have enclosed a copy of my modified program with the various changes highlighted (see accompanying listing and change lines).

To make effective use of the MID\$= function, the variables must be initially set to the maximum length that they might attain. Since LN\$ and FS\$ are the principal string variables involved in the print process, we initialize them as a blank string the length of WD. WD is the longest either string can be for proper program operation. This is done in lines 1365 and 1370, when we declare the data area, and locates both variables in memory with a length of WD bytes. This is the only time these variables are set equal to any value. After this, all changes to LN\$ and FS\$ are made using the MID\$= function. In this manner, the two variables remain stored in the same location and string space is not consumed.

To accomplish this, we must change lines 540, 780, 880 and 1010. Lines 630 and 890 can be deleted since LEN(LN\$) will always equal WD.

Lines 900-920 require that the variable LN\$ be changed to FS\$. Actually, based on lines 880 and 950, I suspect that this was the original plan when the program was written. The only remaining requirement is done in the added line 485, and is called by the changes to lines 570, 810 and 1040. The subroutine at 700, to delete trailing blanks is no longer necessary. The added benefit is that, unlike before, even if you make a field too short when formatting the PRT file, part of the data element will still print.

The changes to allow two-column printing are a little more complicated. To begin I set Z equal to half of WD plus one. This becomes the tab value for the second column. A new variable, CL, is a flag for the number of columns to print. CL is initialized at 1 in line 160 and can be changed in the PRT file using DCOL=2 (see new line 1535). The major changes are in the subroutines at 760 and 870. Line 942 checks the status of the COL flag and the new ZZ (out of data) flag. Line 943 checks to see if the end of the file will occur in the left column and sets the ZZ flag. Line 944 does the same thing for the right column. If data is available for the right column, then line 946 locates the data in LN\$ by adding Z to X. The subscript for DF\$ determines which data item to take. Basically, it subtracts the "overhead" lines from the page length. The remainder is the number of data lines per page which is then added to IC to get the data element for the right column. Line 948 adds a leading blank in the same manner as line 940.

When the program returns to line 800, it checks the flags and adds 1 to new variable RC, a counter for data in the right column. The addition to line 820 checks to see if the file is completed when RC is added in. Line 840 now adds RC to IC. The final change requires that RC be set to zero in line 1070.

The final change incorporates printer control into the program. The flag PR is initialized at zero in line 160. This can be changed using DPRINT=n in the PRT file (see new line 1537). Printer control routines are added at line 1800 to suit each individual user. These routines are reached from the main loop at added line 415 with an error trap added at line 412. Printer modes are cleared at line 475 after the report is finished.

As an example, I have enclosed a printout using the two-column format with a width of 132 characters and compressed type. This is from an Epson MX/80 printer. I have also enclosed a copy of the PRT file which produced this report (see

following figures). These changes should also apply to Ranprint.Bas, although I haven't modified that program yet. I hope these changes prove useful to others; they certainly have for me.

(These changes to Rcard.Bas really make the program much more versatile and useful. The two-column printing is especially neat. - Ed.)

Figure 1

Here are the format headings in the PRT file referred to in the article:

```
DLINES=5
DPAGE=88
DWIDTH=132
DTOP=3
DBOTTOM=3
DCOL=2
DPRINT=3
```

Notice that the last two are new, and control the number of columns and the printer mode.

RAGS Surname List
Alpha Listing

Prepared: 12 February 1989

| No | Name | County | ST | Country |
|----|------------------------------|--------|----|---------|
| 29 | Akins | | TN | |
| 09 | Angelloz | Orlean | LA | Franc |
| 10 | Aue | | ME | |
| 03 | Averill | | GA | |
| 33 | Aycox | Clark | GA | |
| 33 | Aycox | Fulton | GA | |
| 08 | Bachiler, Stephen | Rockin | NH | |
| 27 | Barnard | | MI | |
| 03 | Barnes | | NY | |
| 36 | Bartholmew | | NY | |
| 25 | Baumgardner, John A. | Whythe | VA | |
| 03 | Beach | Knox | OH | |
| 42 | Bean | Jackso | TX | |
| 08 | Bean, John | Rockin | NH | |
| 38 | Behne | | MI | Germa |
| 38 | Behne | | IA | Germa |
| 38 | Behne | | CA | Germa |
| 04 | Bentley | Lincol | GA | |
| 18 | Bible | Sauk | WI | |
| 18 | Bible | Greene | TN | |
| 18 | Bible | Rock | VA | |
| 18 | Bible | Bucks | PA | |
| 18 | Bible/Beibel | | | Germa |
| 08 | Blake, Jasper | Rockin | NH | |
| 29 | Blevins | Indepe | AR | |
| 25 | Blonska(Belanski), Catherine | Klond | | |
| 17 | Blond | | | |

| No | Name | County | ST | Country |
|----|----------------------------|--------|----|---------|
| 08 | Cram, Benjamin | Rockin | NH | |
| 26 | Crawford | | GA | |
| 33 | Crook | Etowah | AL | |
| 03 | Cross | | VT | |
| 29 | Cutburth | | TN | |
| 41 | Davis | Vicksb | TN | |
| 26 | Davis | | TN | |
| 08 | Dearborn, Godfrey | Rockin | NH | |
| 35 | Dettenmayer | Lackaw | PA | Germa |
| 36 | Dickerman | | CT | |
| 36 | Dickerman | | | Germa |
| 02 | Dickey | | NC | |
| 43 | Dixon | | | |
| 25 | Dockery, Harton (S.C.) | Randol | WV | |
| 25 | Dockery, Rowena | Lee | VA | |
| 25 | Dockery, Rowena | Hancoc | TN | |
| 34 | Draper | Canton | MA | |
| 34 | Draper, John | Bangor | ME | |
| 25 | Drewek, Martha (Marcianna) | Dabrow | | Polan |
| 14 | Dubbs | | | |
| 05 | Durant | Clinto | NY | Canad |
| 35 | Durkin | Lackaw | PA | |
| 18 | Eads | | KS | |
| 18 | Eads | Whites | IL | |
| | | Morgan | | |

Fig 2. An example of two-column printing

Listing for Rcard1.Bas. Difference lines between the original Rcard and Rcard1 are given at the end of this listing.

```
100 REM * RCARD1.BAS * Written for CodeWorks magazine by T R Dettmann
110 REM * 3838 South Warner St. Tacoma, WA 98409 (206) 475-2219 voice
120 REM * (206) 475-2356 300/1200 baud download. See Issue 8 for
    details.
130 'CLEAR 10000 ' Use only if your machine needs to clear string
    space.
140 FALSE=0:TRUE=NOT FALSE
150 HDR=TRUE:FTR=FALSE
160 V=750:NF=8:ML=10:CL=1:PR=0
170 PG=66:WD=80:TP=3:BT=3
180 NT=0:NB=0:NC=0
190 PN=0:SF=-1:SC$='''
200 DIM HD$(ML),RC$(ML),FT$(ML)
210 CLS ' This is a clear screen command, change to suit your
    machine.
220 PRINT STRING$(22,45);' The CodeWorks '';STRING$(23,45)
230 PRINT''
    R E P O R T   C A R D''
240 PRINT''
    for use with the Mini-Card Database System''
250 PRINT STRING$(60,45)
260 PRINT
270 ' If your computer has DATE$ it will use that date
280 ' otherwise line 340 will ask for the date and make DATE$ of it.
290 ' For those with only two letter variables, don't worry,
300 ' DA$ is not used elsewhere in the program.
340 IF DATE$=''' THEN INPUT''Enter the date of the report'';DATE$
350 PRINT
360 LINE INPUT ''NAME OF DATA FILE (default = CARD.DAT): '';FF$
365 IF INSTR(FF$,'.dat')=0 THEN FF$=FF$+'.dat'
370 LINE INPUT ''NAME OF REPORT FORMAT FILE (default = CARD.PRT): '';
    RF$
380 GOSUB 1260
390 GOSUB 1090
400 '
410 REM -- main loop
412 IF PR<0 OR PR>4 THEN PRINT ''Printer code in error'':FOR I=1 TO 500:
    NEXT I:GOTO 1750
415 ON PR GOSUB 1830, 1860, 1890, 1920
420 FOR IC=1 TO NR
430   IF HDR THEN GOSUB 500
440   GOSUB 760
450   IF FTR THEN GOSUB 980
460 NEXT IC
470 IF LC>0 THEN GOSUB 980
475 LPRINT CHR$(27)''@''
480 GOTO 1750
485 MID$(LN$,1,WD)=STRING$(WD,' '):RETURN
```

```

490 `
500 REM -- print header
510 FOR I=1 TO TP:LPRINT'' ``:NEXT I
520 PN=PN+1
530 FOR I=0 TO NT-1
540   MID$(LN$,1)=HD$(I)
550   GOSUB 620
560   `
570   LPRINT LN$:GOSUB 485
580 NEXT I
590 HDR=FALSE:LC=NT+TP
600 RETURN
610 `
620 REM -- insert into header/footer lines
630 `
640 IF INSTR(LN$,'#')=0 THEN RETURN
650   X = INSTR(LN$,'#')
660   IF MID$(LN$,X+1,1)='D' THEN MID$(LN$,X)=DATE$
670   IF MID$(LN$,X+1,1)='P' THEN MID$(LN$,X)=STR$(PN)
680   IF MID$(LN$,X+1,1)='F' THEN MID$(LN$,X)=FF$
690 GOTO 640
700 `
710 REM -- strip off trailing blanks
720 `
730 `
740 `
750 `
760 REM -- print data record
770 FOR I=0 TO NC-1
780   MID$(LN$,1)=RC$(I)
790   GOSUB 870
800   IF CL=2 AND ZZ<>1 THEN RC=RC+1
810   LPRINT LN$:GOSUB 485
820   LC=LC+1:IF IC+RC-NT-TP-1>NR THEN IC=NR:RETURN
830 NEXT I
840 IF LC+BT+NB>PG-1 THEN FTR=TRUE:IC=IC+RC
850 RETURN
860 `
870 REM -- put together a data line
880 MID$(FSS$,1)=LN$
890 `
900 IF INSTR(FSS$,'#')=0 THEN RETURN
910   X = INSTR(FSS$,'#')
920   Y = VAL(MID$(FSS$,X+1))
930   MID$(LN$,X) = DF$(IC,Y-1)+' ' ``
940   IF X>1 THEN MID$(LN$,X-1)=' ' ``
942   IF CL<>2 OR ZZ=1 THEN 950
943   IF IC+PG-TP-BT-NT-NB-1>NR THEN ZZ=1:GOTO 950
944   IF DF$(IC+(PG-TP-BT-NT-NB-1),0)='ZZZ' THEN ZZ=1:GOTO 950
946   MID$(LN$,X+Z)=DF$(IC+(PG-TP-BT-NT-NB-1),Y-1)+' ' ``

```

```

948 IF X>1 THEN MID$(LN$,X-1+Z)=''' ''
950 MID$(FSS$,X)=''' ''
960 GOTO 900
970 '
980 REM -- print footer
990 FOR I=LC+1 TO PG-BT-NB:LPRINT'' '' :NEXT I
1000 FOR I=1 TO NB
1010 MID$(LN$,1)=FT$(I)
1020 GOSUB 620
1030 '
1040 LPRINT LN$:GOSUB 485
1050 NEXT I
1060 FOR I=1 TO BT:LPRINT'' '' :NEXT I
1070 HDR=TRUE:FTR=FALSE:LC=0:RC=0:RETURN
1080 '
1090 REM -- load data file
1100 IF FF$=''' THEN FF$='''CARD.DAT''
1110 PRINT ''Loading data file '' ;FF$
1120 OPEN ''I'',1,FF$
1130 NR=0
1140 FOR I=1 TO V
1150 IF EOF(1) THEN 1230
1160 LINE INPUT#1,DF$(I,0)
1170 IF DF$(I,0)='''ZZZ'' THEN 1230
1180 FOR J=1 TO NF-1:LINE INPUT#1,DF$(I,J):NEXT J
1190 REM DEBUG: FOR J=0 TO NF-1:PRINT J;'' : '' ;DF$(I,J):NEXT J
1200 IF SF>=0 THEN IF INSTR(DF$(I,SF-1),SC$)=0 THEN 1150
1210 NR=NR+1
1220 NEXT I
1230 PRINT NR;'' records found''
1240 CLOSE:RETURN
1250 '
1260 REM -- load print file
1270 IF RF$=''' THEN RF$='''CARD.PRT''
1280 PRINT ''Loading Report Format File '' ;RF$
1290 OPEN ''I'',1,RF$
1300 IF EOF(1) THEN 1350
1310 LINE INPUT#1,LN$
1320 GOSUB 1390
1330 GOTO 1300
1340 '
1350 REM -- declare data area
1360 DIM DF$(V,NF)
1365 FSS$=STRING$(WD, '' '' ):Z=INT(WD/2+.5)+1
1370 LN$=STRING$(WD, '' '' ):RETURN
1380 '
1390 REM -- decode the line
1400 REM DEBUG: PRINT LN$
1410 IF LEFT$(LN$,1)='''D'' THEN GOSUB 1480:RETURN
1420 IF LEFT$(LN$,1)='''H'' THEN GOSUB 1570:RETURN

```

```

1430 IF LEFT$(LN$,1)='R' THEN GOSUB 1610:RETURN
1440 IF LEFT$(LN$,1)='F' THEN GOSUB 1650:RETURN
1450 IF LEFT$(LN$,1)='S' THEN GOSUB 1690:RETURN
1460 RETURN
1470 `
1480 REM -- declare report parameters
1490 IF MID$(LN$,2,5)='LINES' THEN NF=VAL(MID$(LN$,8)):GOTO 1540
1500 IF MID$(LN$,2,4)='PAGE' THEN PG=VAL(MID$(LN$,7)):GOTO 1540
1510 IF MID$(LN$,2,5)='WIDTH' THEN WD=VAL(MID$(LN$,8)):GOTO 1540
1520 IF MID$(LN$,2,3)='TOP' THEN TP=VAL(MID$(LN$,6)):GOTO 1540
1530 IF MID$(LN$,2,6)='BOTTOM' THEN BT=VAL(MID$(LN$,9)):GOTO 1540
1535 IF MID$(LN$,2,3)='COL' THEN CL=VAL(MID$(LN$,6)):GOTO 1540
1537 IF MID$(LN$,2,5)='PRINT' THEN PR=VAL(MID$(LN$,8)):GOTO 1540
1540 REM DEBUG: PRINT NF,PG,WD,TP,BT
1550 RETURN
1560 `
1570 REM -- header line
1580 HD$(NT)=MID$(LN$,2):NT=NT+1
1590 RETURN
1600 `
1610 REM -- record line
1620 RC$(NC)=MID$(LN$,2):NC=NC+1
1630 RETURN
1640 `
1650 REM -- footer line
1660 FT$(NB)=MID$(LN$,2):NB=NB+1
1670 RETURN
1680 `
1690 REM -- selection criteria
1700 SF=VAL(MID$(LN$,2))
1710 X = INSTR(LN$,'=')
1720 IF X=0 THEN SF=-1:RETURN
1730 SC$=MID$(LN$,X+1)
1740 RETURN
1750 RUN'MCARD.BAS'
1800 ` -- printer control (epson mx/80)
1810 `
1820 ` -- emphasized type 66 lpp
1830 LPRINT CHR$(27)'E':;:RETURN
1840 `
1850 ` -- compressed type 66 lpp
1860 LPRINT CHR$(15);:RETURN
1870 `
1880 ` -- compressed type 88 lpp
1890 LPRINT CHR$(27)'O';CHR$(27)'O';CHR$(15);:RETURN
1900 `
1910 ` -- compressed type, double strike, 88 lpp
1920 LPRINT CHR$(27)'O';CHR$(27)'O';CHR$(15);CHR$(27)'G':;:RETURN

```

```

Changed->100 REM * RCARD1.BAS * Written for CodeWorks magazine by T R Dettmann
Changed->160 V=750:NF=8:ML=10:CL=1:PR=0
Added-->365 IF INSTR(FF$, '' .dat'')=0 THEN FF$=FF$+''.dat''
Added-->412 IF PR<0 OR PR>4 THEN PRINT ''Printer code in error'':FOR I=1 TO 500:NEXT
I:GOTO 1750
Added-->415 ON PR GOSUB 1830, 1860, 1890, 1920
Added-->475 LPRINT CHR$(27)''@''
Added-->485 MID$(LN$,1,WD)=STRING$(WD, '' '' ):RETURN
Changed->540 MID$(LN$,1)=HD$(I)
Changed->560 '
Changed->570 LPRINT LN$:GOSUB 485
Changed->630 '
Changed->720 '
Changed->730 '
Changed->740 '
Changed->780 MID$(LN$,1)=RC$(I)
Changed->800 IF CL=2 AND ZZ<>1 THEN RC=RC+1
Changed->810 LPRINT LN$:GOSUB 485
Changed->820 LC=LC+1:IF IC+RC-NT-TP-1>NR THEN IC=NR:RETURN
Changed->840 IF LC+BT+NB>PG-1 THEN FTR=TRUE:IC=IC+RC
Changed->880 MID$(FSS$,1)=LN$
Changed->890 '
Changed->900 IF INSTR(FSS$, '' #'')=0 THEN RETURN
Changed->910 X = INSTR(FSS$, '' #'')
Changed->920 Y = VAL(MID$(FSS$,X+1))
Added-->942 IF CL<>2 OR ZZ=1 THEN 950
Added-->943 IF IC+PG-TP-BT-NT-NB-1>NR THEN ZZ=1:GOTO 950
Added-->944 IF DF$(IC+(PG-TP-BT-NT-NB-1),0)=''ZZZ'' THEN ZZ=1:GOTO 950
Added-->946 MID$(LN$,X+Z)=DF$(IC+(PG-TP-BT-NT-NB-1),Y-1)+' ' ''
Added-->948 IF X>1 THEN MID$(LN$,X-1+Z)=' ' ''
Changed->1010 MID$(LN$,1)=FT$(I)
Changed->1030 '
Changed->1040 LPRINT LN$:GOSUB 485
Changed->1070 HDR=TRUE:FTR=FALSE:LC=0:RC=0:RETURN
Added-->1365 FSS$=STRING$(WD, '' '' ):Z=INT(WD/2+.5)+1
Changed->1370 LN$=STRING$(WD, '' '' ):RETURN
Added-->1535 IF MID$(LN$,2,3)=''COL'' THEN CL=VAL(MID$(LN$,6)):GOTO 1540
Added-->1537 IF MID$(LN$,2,5)=''PRINT'' THEN PR=VAL(MID$(LN$,8)):GOTO 1540
Added-->1800 ' -- printer control (epson mx/80)
Added-->1810 '
Added-->1820 ' -- emphasized type 66 lpp
Added-->1830 LPRINT CHR$(27)''E'';:RETURN
Added-->1840 '
Added-->1850 ' -- compressed type 66 lpp
Added-->1860 LPRINT CHR$(15);:RETURN
Added-->1870 '
Added-->1880 ' -- compressed type 88 lpp
Added-->1890 LPRINT CHR$(27)''O'';CHR$(27)''O'';CHR$(15);:RETURN
Added-->1900 '
Added-->1910 ' -- compressed type, double strike, 88 lpp
Added-->1920 LPRINT CHR$(27)''O'';CHR$(27)''O'';CHR$(15);CHR$(27)''G'';:RETURN

```

Basic Techniques

Printing Numbers in Text Lines

Robert L. Anderson, St. Albans, West Virginia. In this article, the author shows how to make the most of PRINT USING to insert numbers correctly in various print statements, something we have all had problems with at one time or another.

GW BASICs PRINT USING statement is a powerful, convenient way of printing string and numeric values in almost any form one could desire. Most user manuals on BASIC contain several pages describing the various parameters which may be utilized to format data. A common use of the PRINT USING statement is to print data in tabular form. The IRA.Bas program in CodeWorks (Issue 11) uses the technique to print a table of IRA projected figures with the values aligned in each of the five columns.

Of course, the PRINT USING statement also may be used to print the value of a numeric variable in a line of text. This use appears in lines 470 to 490 of the IRA.Bas program. A shorter example is the following:

```
100 A=1234.563
110 A$="$$####,##"
120 PRINT "The estimate of";USING
A$;A;:PRINT" is high."
```

These lines produce a properly spaced line:

The estimate of \$1,234.56 is high.

However, if the value of A is only 1.23, we get a line with four unwanted spaces between "of" and "\$" which doesn't look very good:

The estimate of \$1.23 is high.

If the value of the variable contains fewer digits

than the field specified in the PRINT USING statement, spaces are printed in front of the first digit to fill out the field. Conversely, if the value contains more digits than specified for the field, all the digits are printed, but are preceded by a % sign to indicate the error. In this case, there would be no spaces between the preceding word and the number. If the value has a negative sign, an extra space is required to print it. Again, this may result in no space between the preceding word and the number.

The appearance of extra spaces or of no spaces separating the text and the number is unsightly and unnecessary. The addition of a few lines of code can be used to make the program calculate the proper format field specification so that values of a variable between -999,999,999 and +999,999,999 will be printed in a line of text with the proper spacing.

There are several ways of doing this. The one given in Listing 1 is the shortest I have been able to devise. It is designed to print a number within the above range of values without punctuation (commas inserted every third place before the decimal). Listing 2 prints dollar amounts (with the dollar sign immediately preceding the number) without punctuation. Similarly, Listings 3 and 4 will print numbers and dollar amounts with punctuation.

All four listings were written to permit testing of the output by entering various values and ob-

showing how each value appears in a test line of text. Sample runs are provided for each listing.

Listing 1

Line 10 defines the variable V as double precision. This permits testing of variables containing up to 16 digits. Line 20 prompts for the value to be printed.

Lines 30-50 are the heart of the routine. Line 30 calculates DP, the number of digits in front of the decimal point in value V. The FIX function returns the truncated integer of V without rounding which might occur if INT or CINT were used. The resulting whole number is converted to a string number by the STR\$ function, and the length of the string number is measured by the LEN function. Because the string value contains a leading space, value of DP is greater than the actual number of digits by one. This provides the single space we want to separate the number from the preceding word. If the value of V is negative, we add one to DP in line 40 to make room for the minus sign.

The value of DP is used in line 50 to add the proper number of #'s in front of the basic field specifier .## which positions the decimal point and provides for two decimal places. Obviously, the number of decimal places to be printed can be altered by changing the number of #'s after the decimal point. The decimal point and its following #'s can be omitted if the values are to be printed as whole numbers.

Line 60 prints the first part of the sample text, the value of V using the format defined in line 50, and the rest of the sentence. Line 70 prints a blank line and cycles back to line 20 for another number to print. The program is exited by pressing the BREAK or CTRL-C keys.

In a "real" program the definition of V as a double precision variable would occur (if needed) near the start of the program. Lines 30-50 could occur anywhere after the value of V had been determined and before the value was to be printed.

They could even be a subroutine, as illustrated in Listing 5.

Listing 2

This program differs from Listing 1 only in that the value of DP (in line 30) is decreased by one to compensate for the extra character added to the format field by the presence of \$\$ in line 50.

Listing 3

Three additional lines (32, 34 and 36) are needed to compensate for the extra print spaces caused by inserting a comma every third position in front of the decimal point. No comma is needed if V has a value less than 1,000 (DP<5); therefore, line 32 subtracts one from DP. One comma is printed for values between 1,000 and 999,999 (DP<8). Since the comma in front of the decimal point in line 50 adds one position, no correction to DP is needed and line 34 does not change DP. Two commas are printed for values between 1,000,000 and 999,999,999. This requires DP be increased by one which line 36 does.

Actually, the need for line 36 can be obviated by changing lines 30, 32 and 34 as shown in Listing 3A. However, the logic of these changed lines is more difficult to follow.

Listing 4

Line 50 contains both \$\$ and a comma in the field specification. As before, the \$\$ symbols add one position to the field, and the comma punctuation adds one or two print positions depending upon the value of V. Lines 32 and 34 make the needed adjustments to DP.

Listing 1 print a number without commas in a text line.

```
10 DEFDBL V
20 INPUT "Enter a number: ";V
30 DP=LEN(STR$(FIX(V)))
40 IF V<0 THEN DP=DP+1
```

```

50 VF$=STRING$(DP,35)+".##"
60 PRINT "The number";USING VF$;V;" was entered."
70 PRINT:GOTO 20

```

Sample run:
Enter a number: 1234.56
The number 1234.56 was entered.

Enter a number: -.12
The number -0.12 was entered.

Enter a number: 1234567.89
The number 1234567.89 was entered.

Enter a number: 1.236
The number 1.24 was entered.

Listing 2 print a dollar amount without commas in a text line.

```

10 DEFDBL V
20 INPUT "Enter a number: ";V
30 DP=LEN(STR$(FIX(V)))-1
40 IF V<0 THEN DP=DP+1
50 VF$="$$"+STRING$(DP,35)+".##"
60 PRINT "The number";USING VF$;V;" was entered."
70 PRINT:GOTO 20

```

Sample run:
Enter a number: 12.34
The number \$12.34 was entered.

Enter a number: -1234.562
The number -\$1234.56 was entered.

Listing 3 print a number with commas in a text line.

```

10 DEFDBL V
20 INPUT "Enter a number: ";V
30 DP=LEN(STR$(FIX(V)))
32 IF DP<5 THEN DP=DP-1:GOTO 40
34 IF DP<8 THEN 40
36 DP=DP+1
40 IF V<0 THEN DP=DP+1
50 VF$=STRING$(DP,35)+",.##"

```

```

60 PRINT "The number";USING VF$;V;" was entered."
70 PRINT:GOTO 20

```

Sample run:
Enter a number: 1.23
The number 1.23 was entered.

Enter a number: 1234.56
The number 1,234.56 was entered.

Enter a number: 1234567.89
The number 1,234,567.89 was entered.

Listing 3A print a number with commas in a text line.

```

10 DEFDBL V
20 INPUT "Enter a number: ";V
30 DP=LEN(STR$(FIX(V)))+1
32 IF DP<5 THEN DP=DP-2:GOTO 40
34 IF DP<8 THEN DP=DP-1
40 IF V<0 THEN DP=DP+1
50 VF$=STRING$(DP,35)+",.##"
60 PRINT "The number";USING VF$;V;" was entered."
70 PRINT:GOTO 20

```

Sample run:
Enter a number: 0
The number 0.00 was entered.

Enter a number: -1234.56
The number -1,234.56 was entered.

Listing 4 print a dollar amount with commas in a text line.

```

10 DEFDBL V
20 INPUT "Enter a number: ";V
30 DP=LEN(STR$(FIX(V)))
32 IF DP<5 THEN DP=DP-2:GOTO 40
34 IF DP<8 THEN DP=DP-1:GOTO 40
40 IF V<0 THEN DP=DP+1
50 VF$="$$"+STRING$(DP,35)+",.##"
60 PRINT "The number";USING VF$;V;" was entered."
70 PRINT:GOTO 20

```

Sample run:

Enter a number: .23

The number \$0.23 was entered.

Enter a number: 123.451

The number \$123.45 was entered.

Enter a number: -1234.56

The number -\$1,234.56 was entered.

Listing 5 print a dollar amount with commas in a test line, using a subroutine.

```
10 DEFDBL V,A
120 A1=1532.96 ' Actual current value
130 A2=999.756 ' Estimated current value, de-
    termined earlier in the program.
140 PRINT "The estimated current value of";
```

```
150 V=A2:GOSUB 1000:PRINT USING VF$;V;
160 PRINT " differs from":PRINT "the actual";
170 V=A1:GOSUB 1000:PRINT USING VF$;V;
180 PRINT " by";V=A2-A1:GOSUB
    1000:PRINT USING VF$;V;PRINT "."
190 END
1000 ' Subroutine to calculate format specifica-
    tion in punctuated $ and cents.
1010 DP=LEN(STR$(FIX(V)))
1020 IF DP<5 THEN DP=DP-2:GOTO 1040
1030 IF DP<8 THEN DP+DP-1
1040 IF V<0 THEN DP+DP+1
1050 VF$="$$"+STRING$(DP,35)+",.##"
1060 RETURN
```

Sample run:

The estimated current value of \$999.76 differs from the actual \$1,532.96 by -\$533.20.

Notes

James C. McCord of Fairbanks, Alaska writes: "I need some help. How do you write a program so that no two numbers are the same. In other words, if we want to generate from 1 to 6, the program might come up with this series, 6,2,4,3,1,5. I figured it out once but now have forgotten how to do it.

I am writing a program for a friend who teaches sign language to deaf people and to hearing people who work with the deaf. He wants to throw the words on the screen in various orders. I could follow the example of your Drill.Bas program but that always shows the words in the same order.

In other words, if the words were boy, girl, tell, give, you; one time they would come up on the video as girl, give, et., and the next time as boy, give, etc. He claims that if the flash cards are always in the same order, the words are learned in that order, not as individual words to be used in any order.

I thought if I had a random generator that would not duplicate the numbers, then I could use that as a marker to go to various lines to

```
100 REM * Randupe.Bas * random w/o dupes
110 DATA boy,girl,man,woman,cat,dog,house
120 DATA chair,table,spoon,seat,radio,car,bus
130 DIM A$(14)
140 FOR I=1 TO 14
150   READ A$(I)
160 NEXT I
170 '
180 FOR I=1 TO 10
190   RANDOMIZE TIMER
200   A=INT(RND(1)*14)+1
210   IF A$(A)=''' THEN 200
220   PRINT A$(A);'' ''';:A$(A)='''
230 NEXT I
```

show the words in different orders. I could also use timing loops and all the other goodies.

If you could help me on the random generator it would be appreciated."

See the accompanying listing. Each time you run this code the names will come up in a different order. The TIMER in line 190 is a feature of GW BASIC which seeds the random number generator from the internal clock. By nulling out the data after we have used it, in line 220, we insure that it will not be picked again in line 210.

Thanks for the neat problem.

Etax89.Bas

Update to last year's Estimated Tax Program

Col (Ret) John J. Betz, Jr., Gig Harbor,
Washington

ETAX88.Bas was published in the May/June 1988 issue of CodeWorks. The program was designed to enable users to easily estimate their Federal Income Tax withholding obligations for 1988 taxes. The provisions of the tax law change annually and affect both standard and personal deductions, the allowance for personal interest and the income thresholds for the tax rates. The following changes to ETAX88.Bas incorporate the changes in the law and convert the program to ETAX89.Bas.

Most of the changes involve only one number in each line. To expedite the editing of the program the number that has been changed is underlined.

Change in the title:

100 REM ETAX89.Bas Compute estimated income tax for 1989

Changes in the standard deduction:

560 IF FS=1 THEN SD=3100+(TA*750)

570 IF FS=2 OR FS=5 THEN SD=5200+(TA*600)

580 IF FS=3 THEN SD=2600+(TA*600)

590 IF FS=4 THEN SD=4550+(TA*600)

Change in personal credit allowance:

1090 PRINT "Only 20% of personal interest (credit cards, etc...)

Change in personal deduction:

1170 AD=ND*2000:TI=TH-AD

Changes in computation-low bracket:

1730 IF FS=1 AND TI><44900 THEN 1870

1740 IF FS=2 AND TI>>74850 THEN 1870
1750 IF FS=3 AND TI>>37425 THEN 1870
1760 IF FS=4 AND TI>>64200 THEN 1870
1780 DATA 1,0,18550,.15,0,15
1790 DATA 1,18550,250000,.28,2783,28
1800 DATA 2,30950,.15,0,15
1810 DATA 2,30950,250000,.28,4643,28
1820 DATA 3,0,15475,.15,0,15
1830 DATA 3,15475,250000,.28,2322,28
1840 DATA 4,0,24850,.15,0,15
1850 DATA 4,24850,250000,.28,3728,28

Changes in tax rates-high bracket:

1880 IF FS=1 THEN A=44900:IF FS=2 THEN A=74850:IF FS=3 THEN A=37425:IF FS=4 THEN A=64200

1890 IF FS=1 AND TI>>93130 THEN 1970

1900 IF FS=2 AND TI>>155320 THEN 1970

1910 IF FS=3 AND TI>>117895 THEN 1970

1920 IF FS=4 AND TI>>128810 THEN 1970

Changes in tax rates-super bracket:

1980 IF FS=1 THEN A=93130:IF FS=2 THEN A=155320:IF FS=3 THEN A=117895:IF FS=4 THEN A=128810

2000 TC=TX+(ND*560)

The IRS has changed the formula for computing the tax in the super bracket and the following are new lines that must be inserted into the program:

1992 IF FS<>3 THEN 2000

1994 CLS:INPUT "Did you include your spouse in your total number of dependents? Enter Y for yes or N for no ";QD\$

1996 IF QD\$="Y" or QD\$="y" THEN 2000 ELSE TC=TX+((ND+1)*560)

1998 GOTO 2010

Artificial Intelligence

Part I - some background

Irvin Schmidt, Editor. What's intelligence, anyway? And what's the difference between "real" intelligence and "artificial" intelligence. It's always been an intriguing subject, so we did a little research and came up with the following. Future installments will include some sample programs, but keep in mind that AI is difficult to do in BASIC, especially in short programs.

Is artificial intelligence fact, fancy or fraud? After all, does anyone really know what intelligence is in the first place? No one has come up with an exact definition of intelligence, but Douglas Hofstadter (author of the book, *Goedel, Escher, Bach: An Eternal Golden Braid*) suggested the following characteristics: "To respond to situations flexibly", "to make sense out of ambiguous or contradictory messages", "to recognize the relative importance of different elements of a situation", "to find similarities between situations despite differences which may separate them", and, "to draw distinctions between situations despite similarities which may link them."

We all do the above as a matter of course, and, in fact, would lump those characteristics together into what we call "common sense." Machines can't handle these characteristics easily, except of course, in fiction. Fiction, in its own way, often leads the way towards reality, and in artificial intelligence we find that this is more often true than not. In Isaac Asimov's last three books (added to his Foundation Trilogy) he has a shipboard computer upon which you simply place your hands and think destination. The computer reads your thoughts and sets in a course automatically. (One wonders what would happen if the operator were disgusted and was thinking "go to hell" while his hands were on this computer!)

Then, of course, there was HAL in *2001*, by Arthur Clarke. HAL showed off characteristics which are currently the subject of much AI (artificial intelligence) research. It even had a survival instinct, and killed one of the crew in order to keep itself "alive." And you all remember C3-PO and R2D2 from *Star Wars*. And how about that intelligent robot in the television series *"Lost in Space,"* with his swivel head, constantly warning: "Danger! Will Robinson!"

But can machines think? Does it make any difference whether they can or not, so long as something useful is accomplished? Alan Turing helped design one of the first computers ever built. He devised a test, now commonly called the "Turing Test," which essentially says that if you can't tell the responses you get from a computer from those you would get from a human, then you may as well say that the machine is thinking. Well there was, and continues to be, argument on that one.

According to the Encyclopedia Britannica, "Artificial intelligence is the branch of computer science that deals with ways of representing knowledge using symbols rather than numbers and with rules-of-thumb, or heuristic, methods for processing information."

There are several areas of research going on in AI:

Expert systems are designed to act in the place of an expert in any given field. They are also known as "knowledge based systems" and include a knowledge base and rules for applying that knowledge.

Natural language processing is an attempt to make communication with machines easier. There are two areas of consideration here, one is natural language understanding and the other is natural language generation.

Speech recognition research is an attempt to allow computers to understand humans directly and includes aspects of natural language processing.

Computer vision is an area wherein computers are equipped to receive visual images and process information contained in those images. This one is important in the field of robotics.

Robotics includes research in the manufacture of machines that can sense conditions and do mechanical work. Much has been done already in this field, especially in production line work, where robots weld automobile frames, for example.

It is interesting to note that the word "robot" was made universal by the Czech playwright, Karel Capek, who in 1920 wrote a play called "*Rossem's Universal Robots*." "Robot" is the Czech word for "slaves." The play is long since forgotten, but the word lingers on in our vocabulary. For 20 years thereafter, robots were usually depicted as an evil menace. Then, in about 1940, Isaac Asimov and John Campbell devised the "three laws of robotics" which Asimov included in his many "robot" stories. His treatment of robots turned them from evil and menacing into useful, if not endearing, characters.

AI was born, and named, at Dartmouth College in the summer of 1956. The name "artificial intelligence" was suggested by John McCarthy, one of the organizers of a conference held at

Dartmouth that summer. The three other organizers of the conference were Marvin Minsky, Nathaniel Rochester and Claude Shannon. John McCarthy invented the programming language LISP, the most commonly used AI programming language. He also contributed to the development of time-sharing.

Marvin Minsky is best known for his work in the organization and representation of knowledge structures. He founded the Artificial Intelligence Laboratory at MIT. Nathaniel Rochester was Manager of Information Research for IBM.

Claude Shannon, of Bell Labs, used Boolean algebra to describe the operation of electrical switching circuits in 1937. His ideas contributed greatly in the field of information science and to the binary system of information that we currently use in our computers today.

Other countries are engaged in AI research, especially in Great Britain and France. The Japanese have started one of the largest AI projects. It is a ten-year program and is called "The Fifth Generation Project."

Meanwhile, back in this country, let's look at some of the accomplishments in AI that have taken place since the middle 1960s. One of the earliest expert systems was called DENDRAL, and was created by Ed Feigenbaum. With this program he pioneered the rule-based approach in expert systems. DENDRAL was created to help chemists analyze mass spectrograph data. It allows them to identify the chemical structure of an unknown compound. There are millions of such chemical structures. DENDRAL reduces the millions of possibilities to a few, and then works in more detail on those few. It uses two different types of expertise, one for each part of its work. Its work falls into the "best fit" category of AI.

(We will continue this article in future issues and give examples of some elementary BASIC artificial intelligence programs.)

Handy Order Form

- RENEW SUBSCRIPTION:**
Nov/Dec 89 through Sep/Oct 90 _____ \$24.95

- All 4th year issues:**
Nov 88 through Sep 89 _____ \$18.00

- All 3rd year issues:**
Nov 87 through Sep 88 _____ \$18.00

- All 2nd year issues:***
Nov 86 through Sep 87 _____ \$18.00

- All 1st year issues:**
Sep 85 through Sep 86 _____ \$18.00

- DISKS (specify year and computer type) _____ \$15.00**
4th year disk will be ready Sep 1, 1989 Year(s) _____

- "Starting with MS DOS" booklet _____ \$7.00**

*Note
new
lower
prices
on back
issues
and all
disks!*

Postage and handling charges already included.

Diskettes are available for MS DOS, Tandy IV, Tandy III and most CP/M formats ***Please specify your computer type!***

*In year 2 issues, Issue 8 is out of print and will be supplied on diskette. Please specify your computer type if ordering 2nd year issues.

Computer type: _____

- Check/MO enclosed**
- Charge to VISA/MC _____ Exp _____**

Name _____

Address _____

City/State/Zip _____

Clip or photocopy and mail to: **CodeWorks, 3838 South Warner Street,
Tacoma, Washington 98409**

**We accept VISA & MasterCard. You may call in your order:
(206) 475-2219 Thank you.**

789

Index & Download

What's happening with both

Notes, more on scrolling in MS DOS, issue 23, page 4

Beginning BASIC, all about strings, Part 1, issue 23, page 5

Misc, program, using VARPTR for finding strings, issue 23, page 7

Matrix.bas, main program, issue 23, page 8, matrix manipulations

Invoice.bas, main program, issue 23, page 17, an invoice writing program

Notes, keeping files sorted, issue 23, page 19

Notes, automatically centering headings, issue 23, page 26

Outline.bas, main program, issue 23, page 27, an outliner program

Download, notes on the download. issue 23, page 40

As noted earlier in this issue, the download went up in smoke right after the May/June issue went into the mail. We certainly weren't happy about it, since it would entail considerable expense to replace.

To those of you who used it on a somewhat regular basis, we apologize. To compensate, we will be offering disks every three issues starting next year. The price of those diskettes should be less than one or two long-distance calls to the download.

We are aware that many of you used it to get your statistics for the NFL programs. We will be running the NFL series again this coming season, and the only alternative you now have is to get the stats from the newspaper or one of the sporting news magazines. It may be more trouble, but is cheaper in the long run.

CodeWorks

3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
U.S. Postage
PAID
Permit 774
Tacoma, WA

ERICKSON, MIKE / KRJB 2910
BOX 250
MONTE RIO CA 95462

CODEWORKS

Issue 25

Sep/Oct 1989

CONTENTS

| | |
|---|----|
| <i>Editor's Notes</i> | 2 |
| <i>Forum</i> | 3 |
| <i>Beginning BASIC</i> | 5 |
| <i>Artificial Intelligence, Part 2,</i> | 8 |
| <i>Notes</i> | 16 |
| <i>Drill.Bas</i> | 17 |
| <i>Addbook.Bas</i> | 24 |
| <i>Cardconv.Bas</i> | 27 |
| <i>NFL89.Bas</i> | 31 |
| <i>Stat89.Bas</i> | 35 |
| RENEWAL FORM | 39 |
| <i>Index & things</i> | 40 |

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

(c) 1989 80-Northwest Publishing Inc. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, the publisher assumes no responsibility for errors or omissions. **Unless otherwise noted, all programs presented in this publication are placed in public domain.** Please address all correspondence to **CodeWorks, 3838 South Warner Street, Tacoma, Washington 98409**

Telephone
(206) 475-2219

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 is \$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. **VISA and MasterCard orders are accepted by mail or by phone (206) 475-2219.**

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in the USA. 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

Back in the days when computing was a hobby and before IBM got into the game and made it "respectable" it was just fine to have the most convoluted means of getting anything done. After all, you really felt like you were "computing" if you had plenty of buttons to push and were intimately involved with whatever you were running on your machine.

When IBM entered the game in 1981 every large corporation and company put at least an AT model on the desk of middle management and expected great things to happen. Sometimes it did; sometimes it didn't.

The hardware has become more sophisticated and faster than the software has. A recent article in the newspapers lamented this fact in their business sections. They suspected that there were a lot of computer software boxes with manuals and disks inside, gathering dust on the shelves of many businesses. Their complaints are not unfounded.

One such complaint is that there is no consistency from one program to another. It seems that every company which creates a new software product seems to think that they are going to set a new "standard" and that everyone else will follow suit. That's just not the way it works. Microsoft has gone a long way towards some sort of consistency with their Windows. When programs run under Windows the user is always presented with a familiar screen display. Not only that, but the printer connections and other housekeeping is already done in Windows, and need not be altered when installing a new program that runs under Windows. But not everyone is following that idea, either. There are at least three other presentation managers vying for first place in that arena, and they do things differently. Add to that the fact that even Microsoft is readying a new presentation manager for OS/2 that is different than Windows.

Another complaint is that the software is too complex. Even the installation procedure throws most people.

Says the article, "if the manuals are written by a programmer, they're unreadable, and if written by a documentation person, they don't understand the software or the application."

Back in the hobby days, adventure programs were the rage. In those programs, you had to figure your way out of some hidden cave or castle and it was up to you to come up with the way to do it. If you didn't have the "smarts" to figure it out by yourself that was just tough luck. That mentality seems to have carried over to the present day programmers, who seem to say: "I've done something very clever here and I'll bet you can't figure it out."

Either that or they take an awful lot for granted. People who use computers to help make a living just aren't interested in clever programming or pressing CTRL and SHIFT and F5 at the same time to make something happen, especially if you have to press ALT at the same time too. And those advertisements that claim "intuitive" operation are a joke. I recently had to use one and found that to move the cursor you had to push the TAB key, of all things. What's wrong with the arrow keys to move the cursor? That would seem more intuitive to me. My intuition runs on its own and I don't care to have it channeled by someone else.

There are 101 various and sundry keys on the keyboards these days, including at least 10 (and sometimes 12) function keys. These, along with CTRL, ALT and SHIFT ought to be enough for anyone, if they would just get together and decide on some kind of standard. After all, we did just fine when keyboards only had 73 keys. (Yes, I just counted the keys, and there are indeed, 101 of them, and the label on the keyboard says that it's "enhanced" - whatever that is.)

Aside from that, I've done some of my best work on an "un-enhanced" keyboard - which is commentary for this issue.

Irv

Forum

An Open Forum for Questions and Comments

Issue 24 Notes demonstrated a method to null out words from a random list. Jim McCord of Fairbanks might also find useful a slightly longer method I have used for random selection of string items. Rand.Bas determines how many items you want listed, then if there are at least that many items available, it produces non-duplicated numbers which are then used to produce the sequence of string items from an array. As the requested number gets closer to the total items available, the program slows down somewhat in selecting, or more accurately, checking.

```
10 REM * pick randomize seed from clock
20 RANDOMIZE TIMER
30 DIM R(70),A$(70)
40 INPUT "Number to select ";CT:N=0
50 WHILE N<CT
60 N=N+1
70 R=INT(RND*100)
80 IF R > 26 THEN 70
90 FOR CK=1 TO N
100 IF R=R(CK-1) THEN 70
110 NEXT CK
120 R(N)=R
130 WEND
140 DATA a,b,c,d,e,f,g,h,i,j,k,l
150 DATA m,n,o,p,q,r,s,t,u,v,w,x,y,z
160 FOR N=1 TO 26
170 READ A$(N)
180 NEXT N
190 FOR N=1 TO CT
200 PRINT A$(R(N));
210 NEXT N
```

Paul Gillespie
Mesa, AZ

Thanks, it does exactly what you said it would, and should be easy to incorporate into a larger program where it may be needed.

The letter from Fran Hynes of San Francisco in Issue 23 referring to scrolling in GW BASIC prompted me to write. I have the same result, In

BASIC when I press CTRL and Y I get a down arrow. If the cursor is on the "1" in line 100, I get a down arrow replacing the "1." If the cursor is on the line below the "Ok" prompt, I get a down arrow on that line and the "Ok" prompt on the line below. CTRL and X give me an up arrow. I use Microsoft's GW BASIC 3.20 which came with MS DOS 3.21, generic version, used on an XT clone. Any suggestions?

Mary Ann Dobson
Placerville, CA

We finally got the opportunity to try this on various other computers, like a Compaq and an IBM. You are right. It didn't work there. Further checking reveals that Microsoft licenses versions of MS DOS and GW BASIC to vendors and in most cases, does a little customizing for the vendor. The scroll works on all Tandy MS DOS machines, and as yet, we have not found another supplier on whose machines it works, although we obviously can't check them all. Thanks for prodding us into checking this out.

...I would like to cast my vote for part of CodeWorks to be devoted to Microsoft's QuickBASIC. I am curious how others feel. I strongly suspect that graphics work notwithstanding, it would be easy to move from GW BASIC to QuickBASIC and back. Wonder how many readers are aware that GW BASIC actually stands for "GEE WHIZ" BASIC, that even though written by Microsoft, it is not commercially available from them. Rather, it is a product licensed to various manufacturers of computers for packaging with their products...

John R. Miller
Anderson, SC

We were under the impression the the GW stood for "Graphics / Windows" but we could be wrong. As for QuickBASIC, we have been dabbling in it for a little while and like it a lot. In fact, in this issue, we do a program in both just for comparison. The differences aren't that great because we

This is the
LAST
issue
of
YEAR 4

It's time for
everyone

to



RENEW

We have made special arrangements with the U.S. Government to have a uniformed government official actually deliver each and every issue to you.*

*Lest we mislead you, the arrangement is called Third Class Bulk Mail and the Official is your trusted postman!

haven't yet got into the finer points of QuickBASIC. If there is enough interest, maybe a "getting started" series in QuickBASIC would be in order. There are people who will jump on anything new just because it's new, but this does have advantages, and I think it will be around and be supported for quite some time, making it worth the effort to change.

Well folks, we are heading into our fifth year of CodeWorks. There were those, back in Issue 2, who said we had better all get another job. Actually, they were not far from being correct. We do other jobs, in printing and publishing, and CodeWorks has become a part of what we do overall. We hope you didn't even notice when we started doing that. We found that the more you do, the more you can do. Not only that, but the other work has a way of feeding us ideas for the magazine. It's real people who want real solutions to real problems, and that's what makes it fun to borrow from for CodeWorks - it makes it real, too (well, most of the time).

Football, autumn and renewal time are all at our doorstep. We hope you enjoy the autumn and the football, and that you will renew your subscriptions - and we thank you for your support.

Irv



Yes dear, I'm sure if you had your computer you could figure out what you did wrong

Beginning BASIC

All About Strings - Part 3 of 3

In the last issue we looked at the various string commands and functions. This time, we'll show just a few practical uses for those commands and functions. Keep in mind that there are many, many more than we could show here, but these will give you an idea of how they work in regular programs.

Our first example (see Listing for Beg251.Bas) shows a way to use letters as menu items instead of numbers. It also shows how to use INKEY\$ as a "hot" key, which means that you need not press Enter after your choice. Further, it shows how to trap for both upper and lower case letters, and to discriminate sufficiently so that an illegal entry is ignored.

INKEY\$ is used in line 70. Note that as long as C\$ is equal to a null string, the program will cycle on line 70. As soon as C\$ is no longer a null string (meaning you have pressed a key) control falls through to line 80. Line 80 looks for the occurrence of A, a, B, b, C, c, in C\$. It does it by using INSTR. As you will remember from the previous issues, INSTR will tell the position in C\$ where one of those letters occurs. If the letter you pressed is not in C\$ then AN will equal zero and line 90 will send control of the program flow back to line 60, where the question is asked again. This is how we can detect both upper and lower case letters. Lines 100 through 120 are simple logic statements that act on whatever value AN is. It could just as well have been an ON GOTO or ON GOSUB statement. Keep in mind that AN will hold the integer value of the *character position number* within C\$. This is one of the things that makes INSTR so powerful and useful.

```
5 REM * Beg251.Bas
10 PRINT "Menu"
20 PRINT "A - option one "
```

```
30 PRINT "B - option two"
40 PRINT "C - option three"
50 PRINT
60 PRINT "Enter your choice (upper or lower
case)"
70 C$=INKEY$ : IF C$="" THEN 70
80 AN = INSTR("AaBbCc",C$)
90 IF AN = 0 THEN 60
100 IF AN < 3 THEN PRINT "option one was
chosen" : END
110 IF AN = 3 OR AN = 4 THEN PRINT
"option two" : END
120 IF AN > 4 THEN PRINT "option three" :
END
```

AN (in lines 100 to 120) can have a value anywhere between one and six. This is because the literal string (see line 80) has six positions (AaBbCc). A quoted string is always a literal string.

Our next example (see Listing for Beg252.Bas) is similar to the first but with some differences to show different string functions at work. Again, we enter a letter choice and it can be either upper or lower case. Our choice becomes A\$ in line 60. Now instead of using INSTR as we did in the first example, we are going to use the ASC of A\$ to get the ASCII value of whatever letter we chose. But we want to get our integer number from ASC to be in the range of one to three, and the ASCII values for the letters in question are up around 65 or 97. Line 70 takes care of all that by first checking to see if the ASC of A\$ is greater than 96. If it is, it would indicate that we have entered a lower case letter because the ASCII for lower case letters is from 97 on up. In fact, lower case 'a' is ASCII 97. So if the ASC of A\$ is greater than 96 then we let X equal the number and subtract 96 from it. If the letter we pressed was lower case 'a' then ASCII 97 less 96 will equal one, which is what we want. If the ASC

of A\$ is not greater than 96 it must be less than 96, so the second part of line 70, the ELSE portion, subtracts 64 from the ASC of A\$ (because upper case 'A' is ASCII 65. This is how we get the upper/lower case discrimination.

Line 80 checks to see that our number (now in X) is within the range we want, and if not, sends us back to line 60 to try again. This keeps the input within our legal range. Line 90 is a simple ON X GOTO statement, where X can only now be one, two or three. So, you can see that this program does the very same thing that our first example program did, only it uses a different method to do it.

```

5 REM * Beg252.Bas
10 PRINT "Menu"
20 PRINT "A - option one"
30 PRINT "B - option two"
40 PRINT "C - option three"
50 PRINT
60 INPUT "Enter your choice";A$
70 IF ASC(A$) > 96 THEN X=ASC(A$)-96
ELSE X = ASC(A$)-64
80 IF X <1 OR X >3 THEN 60
90 ON X GOTO 100,110,120
100 PRINT "You selected option one" : END
110 PRINT "You selected option two" : END
120 PRINT "You selected option three" : END

```

Our next example (see Listing for Beg253.Bas) is a simple one that demonstrates the use of both VAL and MID\$. Most computers today have a way of keeping the current date. In MS DOS machines it is called DATE\$, and is set either manually or automatically upon power up. DATE\$ is made up of 10 characters, in the form: 05-26-1989, for May 26, 1989. These numbers are all in string form. Line 20 of our sample program pulls the day out of DATE\$, converts it to an integer and puts it into variable X. We use MID\$ to pull the day out of DATE\$ by looking at position 4 for two characters. Then we take the VAL of what we just pulled out. VAL, you will remember, changes a string number into an integer. VAL would also have gotten rid of a leading zero if the day was less than 10.

```

5 REM * Beg253.Bas
10 REM * if date$ = 05-26-1989
20 X = VAL(MID$(DATE$,4,2))
30 PRINT X
40 REM * X pulls integer day out of date$

```

The next program (see Listing for Beg254.Bas) asks you to input a string of assorted characters. It will then determine if there is a lower case 'x' in the string. Here again, we are using that valuable workhorse, INSTR, but we are using it slightly differently than before. Note line 20. It says IF INSTR(A\$,"x"). There is no equal sign involved. This is a simple logic statement that says that if there is a lower case 'x' in the line you have input, the statement will be true, otherwise it will be false. So if the statement is true we go to line 40, else we go to line 50, and print the appropriate remark. Line 30 shows an alternate way to do the same thing without using the logic statement. Here, we let variable P equal the position in the string where the 'x' occurs. If P is zero it means there was no 'x' and we print the appropriate remark. If P is other than zero it means there was an 'x' in the input line (A\$) and we go to the other remark and print it.

```

5 REM * Beg254.Bas
10 INPUT "Enter a string of characters";A$
20 IF INSTR(A$,"x") THEN 40 ELSE 50
30 REM * alternate line 20:
P=INSTR(A$,"x"):IF P=0 THEN 50 ELSE 40
40 PRINT "There was an x in the string" :
END
50 PRINT "There was no x in the string" :END

```

Our last example (see Listing for Beg255.Bas) shows how to take apart a BASIC line that was saved in ASCII format and put the separate parts of it into different variables. We will strip off the line number and make an integer out of it, then put the rest of the line into string variable R\$.

Line 10 asks you to enter a BASIC statement with a line number, and puts that into variable A\$. Line 20 is interesting because in it we will

look for the first space in the line we have input. BASIC always puts a space right after the line number, and that's what we are looking for. In line 20, we let variable S equal the position in A\$ where the first space (CHR\$(32)) occurs. Now we can go to line 30, where we let LN (Line Number) equal the VAL of the LEFT\$ of A\$ for S minus 1 characters. The minus one gets rid of the space itself, and leaves us with the integer value of the line number in LN. In line 50, we let R\$ equal the RIGHT\$ of A\$ for a number of characters equal to the length of A\$ less whatever position S was. In other words, we are looking at the right side of the string (A\$) but just up to the space after the line number. This puts the string, less the line number, into R\$.

```
5 REM * Beg255.Bas
10 INPUT "Enter a BASIC statement with a
line number";A$
20 S = INSTR(A$,CHR$(32)) ' or INSTR(A$,"
")
30 LN = VAL(LEFT$(A$,S-1))
40 PRINT "The line number you entered
was";LN
```

```
50 R$ = RIGHT$(A$,LEN(A$)-S)
60 PRINT "The rest of what you entered was:
";R$
```

Naturally, these examples are just a few of the ways you can use string manipulation. You can mix and match these statements and functions to do just about anything you want to do with strings. One of the things to always remember is just what form of string you are going to operate on. You always need to know this before you can write code to operate on strings. In the DATE\$ example we talked about earlier, we know that the date is always in that form, even when the day or month is less than 10, so we can write a general routine that will always operate on DATE\$ the same way and give the same results. Again, we know that BASIC always puts a space (the very first space in the line) right after the line number. Using that, we can strip off the line number with confidence that it will work every time. Any time you want to do something you generally have some kind of input, some kind of processing and some kind of output. Keep that in mind when working with strings, and they will do whatever you want from them.

Notes

Here's a little program that fits right into some of the ideas we encountered in Beginning BASIC for this and the past two issues. It's called WLC.Bas (for Word, Line, Character count).

It will read an ASCII text file and report to you how many characters, words and lines it contains.

You might notice that there are no arrays declared in this program. You don't need one. It is possible to open a sequential file and read in one line at a time until you get to the end of the file. We do just that between lines 150 and 270 of our little program. (The program is listed on page 16)

To give you something to watch while the program is working, we print a period on the screen for each line that it inputs.

Basically, we read in a line from the file, then go through that line looking for spaces. First we add the total number of characters in the line to character count. Then when we find a space, we count up one for words and subtract one from characters. Lines are the easiest to count. We just increment variable LN each time the loop goes back to read in another line.

Compile this one for much better speed.

See page 16 for the program listing.

Artificial Intelligence

Part 2 of 2 - more background and some programs

Irvin Schmidt, Editor. Last time we got a small start on some of the background concerning Artificial Intelligence, including the origin of the word "Robot" and how science fiction in some ways led the thinking about AI. We left off, last time, with a program called DENDRAL, which is used to analyze mass spectrograph data to identify chemical structure of unknown compounds.

Digital Equipment Corporation developed an expert system called R1 to design the layout and cabling of large computer systems. R1 was initially developed at Carnegie-Mellon University in 1979. Basically, you feed R1 all the information about a customer site, and it then generates a layout of the floor plan that is needed, the various interconnect components needed and the cables needed to hook everything up. R1 has constantly been revised over the years, and now is able to outperform people more than 90 percent of the time. It has the ability to make sure that all items are present and can spot incompatible items.

Another expert system from about 1971 is called MYCIN. It was designed to help medical people diagnose infectious blood diseases. It even prescribes the proper antibiotics for them! Unlike DENDRAL and R1, MYCIN asks questions and evaluates answers. You can even ask why it wants to know something and it will tell you. It gives answers in probabilities, for example, it may tell you that, "there is evidence (.8)" that a given therapy is (or is not) suggested.

It was found that by removing the knowledge base for infectious disease from MYCIN, the "inference engine" and the natural language interface left could be used for other types of knowledge. The natural language interface and the inference engine were called EMYCIN (for

essential MYCIN), and new knowledge rules were added to make two new systems, PUFF and ONCOCIN. PUFF (stands for Pulmonary Function disorder diagnosis) is used to diagnose breathing disorders. ONCOCIN is used to diagnose cancers.

Then there is PROSPECTOR, which helps geologists find likely sites of rare minerals. It can identify commercially valuable ore deposits. It was developed at the Stanford Research Institute in the late 70s. One of its "finds" was in 1981, a molybdenum deposit in eastern Washington state worth \$100 million! Like MYCIN, it too, asks questions and gives answers with a probability of certainty.

AI has come a long way since the Dartmouth conference, although most of the research has been slow in being translated into practical products. Many corporations have their own AI research organizations, but most of the research being conducted today is at universities. The three universities which stand out in this field are Carnegie-Mellon University, Massachusetts Institute of Technology (MIT), and Stanford University.

Other schools where AI is being developed are: the University of Pittsburgh, where medical expert systems are being developed; Yale Uni-

versity, where natural language research is being conducted; Purdue University, involved with research in Robotics.

The U.S. Department of Defense, through its Defense Advanced Research Projects Agency (DARPA), is involved in funding many of the university research programs. Although not interested primarily in basic research, they have allowed considerable freedom to researchers in the field with the obvious aim of developing products useful in weapons and defense.

There are seemingly insurmountable problems with most aspects of AI. Consider natural language processing: You can say, "I waited for the bus for a long time," or, "Cave-men lived on earth a long time ago." Note the difference in the image of "long time." The two are not even remotely equitable, yet a computer wouldn't be able to make that distinction as easily as a human would. Our language, itself, is full of ambiguities and must, for the most part, be considered in context. When you say, "I hit the man with the bat," do you mean you picked up a bat and hit the man or that you hit the man who was holding a bat? In fact, natural language seldom adheres to rules when you really think about it. Here is a conversation I overheard in a McDonald's recently:

"Do you accept Traveler's Checks?"

"Master Card or Visa?"

"American Express."

"Yes."

Does it make sense to you? I didn't to me, but somehow both parties got what they wanted; one the hamburger and the other the money. I couldn't help thinking about that old non-sequitur, "Do you walk to school or carry your lunch?"

When we use the language we also, without thinking about it, put things into their proper context, identify the situation to one we already know, and use a certain amount of expectation. In other words, we draw conclusions as the conversation progresses. In fact, a good joke

plays on that fact in that it will lead you to believe something will happen, and then the punch line is usually a switch of events or a play on words; something you didn't expect. Exactly why we laugh at such jokes is still a mystery. Why laugh instead of cry? Could a computer handle a joke properly?

When it comes to natural language understanding, you already know a whole lot about it. Consider your computer: If you want to see your files from the DOS prompt, you must type DIR and enter. If you want to see your BASIC files, you must type DIR *.BAS and enter. Hardly anything else will do. You can't just type in "Show me some files," or "What BASIC files do I have?" When you try that, you will get the ubiquitous "File not found."

As you can see from just a few examples, AI is a tough nut to crack. It is all the more difficult to do in a language like BASIC, which simply was not designed for that purpose. You can make a program look like it is reasonably smart, but that still falls far short of what is expected of a program which can be said to be artificially intelligent. Here, for example, are the features expected of an expert system:

The program should be useful and developed to meet a specific need where assistance is needed.

The program should be usable. It should be designed so that even a novice finds it easy to use.

The program should be educational. It should increase the expertise of its users.

The program should explain its own advice. It should be able to show its reasoning processes.

The program should be able to respond to simple questions.

The program should be able to learn new knowledge. It should be able to ask questions

and learn from them.

The program should be easy to modify, to correct errors and to enter new information.

(These points were taken from Bruce G. Buchanan and Edward H. Shortliffe, eds., *Rule-Based Expert Systems*, Reading, MA. Addison-Wesley, 1984)

With all that, we get to our two examples of what will probably be a poor pay-off for all that build-up. The first is an adaptation of the old program called "Animal." It starts out knowing one animal, a bird, and one question, whether or not it flies. If you tell it you are thinking of an animal, it will ask you if it flies. If you say it does, it will tell you that you were thinking of a bird, otherwise, it will ask you what animal you were thinking of and to enter a question that will differentiate that animal from a bird. It then builds a repertoire of animals and corresponding questions. From there, it's a simple matter of matching "yes" answers to the corresponding animal.

It's not very bright in that it must ask each question every time until you get to the right question and answer yes. In spite of that, it packs quite a bit of cleverness into just a few lines of code. Mostly, it is how the questions and responses are phrased. Someone who knows absolutely nothing about computers may think it was smarter than it is.

Our next example, *Expert4.Bas*, has a bit more utility. It can tell you why it asks a particular question, for example, and will only ask questions that are relevant to the line of reasoning. It can't learn and it cannot draw conclusions that are not already programmed into it.

Some background information is necessary before you can appreciate the problems this program is supposed to solve. The program is supposed to be a trouble shooting aid for a typesetting and film processing unit. The typesetters (there are two identical machines) are photo-

typesetters. That is, they expose silver paper film with light extruded through a spinning disk with the alphabet etched in it, through a lens system to the photo paper. There is a lens f-stop involved, similar to the f-stop on your ordinary, everyday camera. The light source is an electronic strobe flash which fires when the correct symbol is in line with the light source and the lens system. A mirror system then directs each letter or symbol across the photo-sensitive paper to create lines of type.

The film processor uses two chemicals, a developer and a fixer. In addition, it has heaters to keep the chemicals at a certain temperature and there is a water bath following these chemicals and then a dry box to dry the film after it is processed. The speed of the film through this processor is controlled by a continuously variable speed control.

The program is designed to help operationally, not to correct malfunctions, which means it will deal with controls and operator adjustable items, but not with most electrical or mechanical breakdowns. It will, however, detect an inoperative light source in the typesetters or a blown fuse in the dry box of the film processor.

The fact that there are two typesetting machines makes trouble shooting a snap in that it almost immediately isolates the problems to either the typesetters or the film processor. Eliminating just one of the typesetting machines would probably increase the size of this program ten-fold.

The program lists eight symptoms from which you can choose. This immediately limits the scope of the program; if your symptom is not there you are out of luck. The "brains" of the program are contained in lines 340 through 410, and the "brain control" is contained in lines 420 through 460.

Depending on which symptom you pick, A\$ will be equated to a series of numbers. These numbers contain all the possible paths that can

be taken for that symptom, including the NO answers. These numbers will be changed to their integer values by the VAL function in line 430, and then will be used in the ON C GOSUB statement in line 440. Note that it's ON C GOSUB and not GOTO. When we return from any subroutine (all the questions and solutions are contained in subroutines) we come back to the line following the line that called the subroutine, line 450 in our case. Here, we immediately increment the I count by two to get to the next pair of digits in A\$. In the question subroutines themselves, we already incremented I by two if the answer was YES. That was to take us over the NO response in A\$. That NO response would in most cases ask another question.

You can answer any question with a question mark to ask why the program wants to know the answer to a particular question. It was expedient to include this question mark response right along with the question that was being asked in the first place. If you do not answer with Y or N, the program will automatically fall through to print the response to the question mark and then repeat the question (via the following GOTO).

There is only one stated solution for each of the eight symptoms, and the program comes to an end after each of these.

The series of numbers contained in A\$ present a guided tour through the subroutines of the program. We always jump ahead two places in A\$, and if your response to a question was YES, we jump ahead an additional two places. Let's take symptom number 3 for an example and follow it part of the way through. When A=3 then A\$ starts with the number 1, which in our ON GOSUB will take us to line 470. Line 470 asks the question about the other typesetting machine output being similar. If we answer with a question mark it will tell us that it is trying to determine if the problem is in the typesetting machine or the film processor. The original question is then repeated.

If we answer YES, we increment I by two and

return, where I is again incremented by two. This would take us to the number 02 in A\$, which would take us to line 550 in the ON GOSUB line. Line 550 asks another question about the processor speed. If we answer YES to this one we increment I by two right there and again when we return. This would take us to the 03 in A\$ which takes us to line 640 in the ON GOSUB line. Line 640 asks yet another question about the temperature of the developer. Let's assume we answer NO to this question. Now we return and increment I by only two, which takes us to 07 in A\$ and line 890 in the ON GOSUB line. Line 890 finally gives us a solution to the problem and ends the program.

Whose expertise is in this program? Mine, of course. I wrote it based on about 10 years experience with typesetters and darkrooms. Now that I am trying to analyze just how the program works I'm finding more in it than I had originally thought was there. In spite of that, it's a simpleton program and not really worthy of the AI label. It is rigid, narrow and one-track minded. It can't learn. It can't draw conclusions other than those which are programmed into it. It could, however, get an apprentice going in the right direction if confronted with a problem, so it is not entirely worthless.

You can see what the problems would be if you wanted to add a new symptom to the program, can't you? Sure, it could be done, following the format already laid down in the program, but it wouldn't be easy. Now how about having the program modify itself? Forget it. Wrong structure, wrong language.

Those series of numbers in A\$ remind me of a description of how the brain works (synapses, and all that). Only in a real brain, those numbers would be in a three-dimensional version, with threads following through in all possible directions, using parts of this string and parts of that one. It boggles the mind at the complexity one could come up with. Add a lifetime of emotions and memories - wow!

When all is said and done, the question is still whether or not we can make machines artificially intelligent. I say we can and will make them appear to be intelligent in a lot of ways and that they will serve some very useful purposes. But I seriously doubt that we will ever have a

machine that is even half as versatile and flexible as an average four year old human. The human mind is more exquisitely complicated than any of us ever imagined, and AI will tell us more about ourselves than it will about AI.

Animal.Bas for all machines

```
100 REM * Animal.Bas * pseudo AI exercise
190 `
200 DIM A$(50),B(50),C$(50)
210 A$(1)='does it fly'
220 B(1)=1
230 C$(1)='Bird'
240 N=1
250 `
260 CLS
270 INPUT''Are you thinking of an animal (y/n)'';Y$
280 IF Y$='N' OR Y$='n' THEN END
290 CT=1
300 FOR I=1 TO N
310 PRINT A$(I);:INPUT `` (y/n)'';Y$
320 IF Y$='N' OR Y$='n' THEN 360
330 FOR J=1 TO N
340 IF B(J)=I THEN PRINT''You were thinking of a
``;C$(J):PRINT:GOTO 270
350 NEXT J
360 NEXT I
390 `
400 N1=N+1
405 PRINT
410 PRINT''I don't know that animal yet.''
420 INPUT''What animal were you thinking of ``;C$(N1)
430 B(N1)=N+1
440 PRINT''Please enter a question that will distinguish a ``;C$(N1)
450 PRINT''from a ``;C$(N)
460 INPUT A$(N1)
470 N=N1
480 GOTO 270
```

Expert4.Bas for all machines

```
100 REM * Expert4.Bas * elementary exercise in AI
110 `
120 GOTO 180
130 ` upper case converter subroutine
140 C$=LEFT$(Y$,1)
150 IF C$=>'a' AND C$=<'z' THEN C$=CHR$(ASC(C$)-32)
160 Y$=C$
170 RETURN
180 `
190 CLS
200 PRINT''Typesetter paper from processor:
210 PRINT
220 PRINT'' 1 - paper comes out white, no type shows
230 PRINT'' 2 - paper comes out all black
240 PRINT'' 3 - blurred type (not sharp)
250 PRINT'' 4 - paper comes out all wet or damp
260 PRINT'' 5 - type is too light
270 PRINT'' 6 - type is too dark
280 PRINT'' 7 - background is grey
290 PRINT'' 8 - dark spots appear at regular intervals
300 PRINT
310 INPUT''The number that describes your symptom is ``;A
320 IF A<1 OR A>8 THEN 310
330 `
340 IF A=1 THEN A$=''011112"
350 IF A=2 THEN A$=''13"
360 IF A=3 THEN A$=''01050206030708"
370 IF A=4 THEN A$=''020609"
380 IF A=5 THEN A$=''01100307020608"
390 IF A=6 THEN A$=''01050307020608"
400 IF A=7 THEN A$=''0114041615"
410 IF A=8 THEN A$=''18"
420 I=1
430 C=VAL(MID$(A$,I,2))
440 ON C GOSUB 470,550,640,720,810,850,890,940,1000,1050,1100,1140,
    1190,1240,1290,1350,1390,1440
450 I=I+2
460 GOTO 430
470 INPUT''Is the other typesetter output similar (y/n/?)'';Y$
```

```

480 GOSUB 140
490 IF Y$='Y' THEN I=I+2:RETURN
500 IF Y$='N' THEN RETURN
510 PRINT
520 PRINT''I am trying to determine if the problem is in the
530 PRINT''processor or in the typesetter.
540 GOTO 470
550 INPUT''Is the processor speed adjusted correctly (y/n/?)'';Y$
560 GOSUB 140
570 IF Y$='Y' THEN I=I+2:RETURN
580 IF Y$='N' THEN RETURN
590 PRINT
600 PRINT''Because too slow a speed will overdevelop, while
610 PRINT''too fast a speed will underdevelop the paper
620 PRINT''and leave it with insufficient drying time.
630 GOTO 550
640 INPUT''Is the developer temperature correct (y/n/?)'';Y$
650 GOSUB 140
660 IF Y$='Y' THEN I=I+2:RETURN
670 IF Y$='N' THEN RETURN
680 PRINT
690 PRINT''Because higher developer temperatures will over-
700 PRINT''develop, while lower ones will underdevelop the paper.
710 GOTO 640
720 INPUT''Is the fixer temperature correct (y/n/?)'';Y$
730 GOSUB 140
740 IF Y$='Y' THEN I=I+2:RETURN
750 IF Y$='N' THEN RETURN
760 PRINT
770 PRINT''The fixer temperature is not that critical, but if
780 PRINT''it is too high or low it will cause greyness in
790 PRINT''the background of the paper.
800 GOTO 720
810 PRINT''SOLUTION: The typesetter f-stop is open too far.
820 PRINT''Close down and try again. The film is seeing too much
    light.
830 END
840 `
850 PRINT''SOLUTION: Adjust the processor ft/min speed to the
    correct
860 PRINT''setting and try again.
870 END
880 `

```

Drill.Bas (Drill.Bas)

```
890 PRINT''SOLUTION: Adjust the temperature of the chemicals and
900 PRINT''wait for them to stabilize at the new temperature before
910 PRINT''trying again.
920 END
930 `
940 PRINT''SOLUTION: The developer solution was mixed improperly.
950 PRINT''Dump the developer and mix new solution using proper
960 PRINT''proportions of concentrate and water. Bring up to proper
970 PRINT''temperature before trying again.
980 END
990 `
1000 PRINT''SOLUTION: The drying box is inoperative. Is is probably
1010 PRINT''a blown fuse, but could be a burned out heating
    element.
1020 PRINT''Replace the fuse and if that fails, call service.
1030 END
1040 `
1050 PRINT''SOLUTION: The typesetter f-stop is closed down too far.
1060 PRINT''Open it up somewhat and try again. The film is not
    seeing
1070 PRINT''enough light.
1080 END
1090 `
1100 PRINT''SOLUTION: The light source in the typesetter is
1110 PRINT''inoperative. Call for service.
1120 END
1130 `
1140 PRINT''SOLUTION: The problem is in the processor. The
    developer
1150 PRINT''section probably contains only water from the last
    cleaning
1160 PRINT''or flushing, or, the developer solution is exhausted.
1170 END
1180 `
1190 PRINT''SOLUTION: There is a serious light leak in the system.
1200 PRINT''Did you close the processor cover? Check for obvious
1210 PRINT''sources of light.
1220 END
1230 `
1240 PRINT''SOLUTION: The fogging of the paper occurred in the
1250 PRINT''typesetter or on the way to the darkroom. Look for
1260 PRINT''obvious sources of light.
1270 END
```

```

1280 `
1290 PRINT'' SOLUTION: The fogging of the photo-sensitive paper
1300 PRINT'' occurred in the darkroom. Check the safelights. Has
1310 PRINT'' one been replaced with the wrong wattage? Is the light
1320 PRINT'' seal around doors intact?
1330 END
1340 `
1350 PRINT'' SOLUTION: Bring the fixer temperature up to normal
1360 PRINT'' and try again.
1370 END
1380 `
1390 PRINT'' SOLUTION: Dump the fixer solution and mix a new batch
1400 PRINT'' according to the instructions on the boxes. Bring up
1410 PRINT'' to proper temperature before trying again.
1420 END
1430 `
1440 PRINT'' SOLUTION: Paper in the supply portion of the typesetter
1450 PRINT'' was fogged while being loaded. If spots are getting
1460 PRINT'' smaller they will go away, otherwise load fresh roll
1470 PRINT'' of photo paper in supply of typesetter.
1480 END

```

Notes, continued from page 7

```

100 REM * WLC.Bas * counts words/lines/chars in ASCII files.*
110 CLS
120 'clear if necessary
130 LINE INPUT'' File name '';FF$
140 OPEN ''I'',1,FF$
150 IF EOF(1) THEN 280
160 LINE INPUT #1,IN$
170 PRINT ''.'';
180 LN=LN+1
190 CC=CC+LEN(IN$)
200 A=0
210 A=A+1:IF A=LEN(IN$) OR A>256 THEN 260
220 IF MID$(IN$,A,1)<>CHR$(32) THEN 210
230 IF MID$(IN$,A,1)=CHR$(32) THEN A=A+1:CC=CC-1:GOTO 230
240 WC=WC+1
250 GOTO 210
260 WC=WC+1
270 GOTO 150
280 CLOSE
290 PRINT
300 PRINT'' Number of chars '';CC
310 PRINT'' Number of words '';WC
320 PRINT'' Number of lines '';LN

```


Drill.Bas

(RE-NAMED
DRILL2.BAS)

A Vintage Program from Dettmann

Terry R. Dettmann, Associate Editor. Here's an interesting program Terry wrote almost 10 years ago for the Model I Tandy computer. It was originally designed for tape, but we have updated it a bit for disk files. We found it to be ideal for use in trivia type questions and answers.

The program, Drill.Bas, is designed to be a memory improvement device which works by forcing you to drill on short words or phrases. The program remembers your mistakes and after a particular word has come up at least five times, it will come up more or less often, depending on your answers.

Questions you fail to answer correctly will come up more often than those you get right every time so that you will be forced to drill more on those questions you don't know well. After a target percentage of correct answers is reached in each question, you will automatically be shown your score.

You may stop the drill at any time and see your score by typing .STOP (the period is important). You can also have the correct answer displayed by typing .HELP but asking for .HELP counts as a wrong answer.

You can choose to display questions and give only answers, or you may choose to mix the questions and answers so that when given the answer you must supply the question (ala Jeopardy). This makes a great trivia exercise of the program.

There is also an edit feature in the program wherein you can fix any set of questions/answers you have on file. Each quiz is a separate file on

disk and the file extension .DRL will automatically be added to your filename (so don't add an extension of your own!).

To use the program, first select option 1 to create a drill. The program will ask for a drill (filename). This will be used to identify the drill on disk later. Then it will ask for the questions and the correct responses. When you are done, type END and you will be returned to the menu. A maximum of 40 questions and answers may be input for each drill (file).

Before using a drill, it is best to save it on disk. The program uses a standard sequential file format which stores the drill name and number of questions at the beginning of the file.

When you are ready with a new drill, or one you have loaded, select option 4 of the menu to commence the drill session. After some basic instructions, the program will ask if you want the questions and answers mixed. Answer Y or N. If you answer Y, you will be given the questions sometimes and the answers other times. You must give the correct answer (or question).

Program Notes

Rather than give a complete line by line description, we will note some of the more unusual and interesting lines. In the initialization sec-

(200 - 20000) 250-1100
tion, lines 170 to 250, we find some print formatting lines which remind us of Mr. Anderson's article on PRINT USING in the last issue. Lines 200, 210 and 230, in particular, show how to embed string constants in a PRINT USING format statement.

Those of you who have machines that cannot use more than two letter variable names need not worry about the multi-lettered variables. The first two of each are not duplicated.

Line 250 is used to seed the random number generator. The statement there now is specific to MS DOS machines. Others should change this line to your particular randomize statement. The purpose of this line is to prevent the same sequence of random numbers (and consequently, questions) to come up repeatedly.

Note that the program is menu driven and uses a series of subroutines. The main line of the program actually ends at line 390, and everything following is in the form of subroutines. After returning from any subroutine, program flow will come back to line 380, which sends control back to line 240, where the title of the program and the menu are again printed.

In the scoring section, lines 430 to 490, provision is made not only to count right and wrong answers, but to tell *which* question was scored right or wrong. This section, in line 480, also calculates the percentage of correct answers. Note that after the drill, you will be given the opportunity to look at each question and get an individual score for that question only. The array, ANS(I,n) will hold those individual score numbers.

Something we haven't done that much in our normal programming in the magazine is the extensive use of "hot" keys. Hot keys respond as soon as they are pressed (without the need to press return or enter). The INKEY\$ routine at line 510 and on handles the hot keys. It is interesting to note the checks you must do when using this method of input. In line 550, for

example, a check is made to see if the enter key was pressed (ASCII 13) or if the backspace key was pressed (ASCII 8). Line 560 checks to see that the key input was within the range of acceptable characters or digits, both upper and lower case. These statements loop to each other, and the only way out is to finally press enter to return from the subroutine (in line 550).

Single key number input is handled by a one line subroutine at line 620. We talked about this type of input in Beginning BASIC in this and the past two issues. The single key yes/no is similar in construction, at line 740.

The random selection routine in lines 780-800 is where changes will need to be made if your random number generator does not select a number between 0 and 1. The way the program is shown, the random selection is for MS DOS machines (and some CP/M BASIC). For Tandy Model I and III, for example, you will need to change the RND in line 780 to: I = RND(N): etc., and line 790 to: IF (RND(150/100)) etc., and line 800 to: . . . THEN J1 = RND(2): etc.

You can both enter a drill or edit an existing drill. The section of code from line 990 to 1170 is where this happens. Note that you are asked for the file name for the exercise, which will become the name of the file on disk. Later, when you want to save the drill, you will be shown this name (in line 1210) and asked if that is the one you want to use. At that point, you can use the existing name or choose a new name. On disk (so that you will be able to identify these files) the file name will have a .DRL appended to it. When loading or saving a drill, however, do not use the extension, since the program will add it when necessary.

Now that all these things are taken care of, we can finally run a drill, starting at line 1500. First, we get instructions, then we get to choose whether or not to mix the questions and answers. The code from line 1720 on puts the questions (or answers, if you chose to mix) on the screen and prompts you for the response. We

then need to check for cries for help or to stop, which happen in line 1740 and 1750.

The scoring section, starting at line 1820, gives an overall score first, then asks if you want a question by question breakdown. If you do, it will show you each question and tell you how you did on it.

The error trap at the end of the program is for file not found. Those with BASIC prior to version 5.0 should remark line 2060 and un-remark line 2070. The trap does not fix anything, it simply

tells you that the file does not exist yet and sends you back to the main menu to make another selection or ask for a file that does exist.

This turned out to be a fun program. It's great for trivia buffs, because you can create all kinds of files to use. Be careful when answering questions; punctuation and capitalization must be exactly the way you entered the quiz or the question will be counted wrong. One way to beat that, though, is to enter everything in all upper or all lower case.

Drill.Bas (see text for conversions)

```
100 REM *****
110 REM * Drill.Bas * T.R. Dettmann
120 REM *****
130 `
140 ` ---- Initialize ----
150 `
160 ON ERROR GOTO 2060
170 `CLEAR 5000 ` only if your computer needs it
180 DEFINT A-Z:MX=50:DIM A$(MX,2),ANS(MX,3)
190 STAR$=STRING$(63,42):S1$=STRING$(28,32)
200 F1$=` ## WRONG OUT OF ##`
210 F2$=` SCORE: ##% `
220 F3$=`%'+STRING$(28,32)+'\ \'+STRING$(28,32)+'%`
230 F4$=`QUESTION ## OF ##`
240 TITLE$=`DRILL PERIOD`:GOSUB 640
250 RANDOMIZE TIMER
260 `
270 ` ---- print menu ----
280 `
290 PRINT
300 PRINT TAB(10);`1 - Enter or edit a drill
310 PRINT TAB(10);`2 - Save a drill to disk
320 PRINT TAB(10);`3 - Load a drill from disk
330 PRINT TAB(10);`4 - Run a drill
340 PRINT TAB(10);`5 - Quit this program
350 PRINT:PRINT`Your choice `:GOSUB 600
360 IF C<1 OR C>5 THEN 350
370 ON C GOSUB 990,1190,1350,1500,2000
```

```

380 GOTO 240
390 END
400 `
410 REM ---- scoring ----
420 `
430 W=0:R=0
440 FOR I=1 TO N
450   W=W+ANS(I,1)
460   R=R+ANS(I,2)
470 NEXT I
480 SC=(R/(R+W))*100+.5:SC=SC-1
490 RETURN
500 `
510 REM ---- inkey$ routine ----
520 `
530 IN$=""
540 C$=INKEY$:IF C$="" THEN 540
550 IF ASC(C$)=13 THEN RETURN ELSE IF ASC(C$)=8 THEN 580
560 IF ASC(C$)<32 OR ASC(C$)>127 THEN 540
570 IN$=IN$+C$:PRINT C$;:GOTO 540
580 IF LEN(IN$)<1 THEN 540 ELSE IN$=LEFT$(IN$,LEN(IN$)-1):PRINT C$;
   :GOTO 540
590 `
600 REM ---- single key number input ----
610 `
620 C$=INKEY$:IF C$="" THEN 620 ELSE C=VAL(C$):RETURN
630 `
640 REM ---- Heading type I ----
650 `
660 CLS:PRINT STAR$;CHR$(13);TAB(25);TITLE$;CHR$(13);STAR$;
   STRING$(2,13):RETURN
670 `
680 REM ---- Heading type II ----
690 `
700 CLS:PRINT S1$;CHR$(13);TAB(10);TITLE$;CHR$(13);S1$;STRING$(2,
   13):RETURN
710 `
720 REM ---- single key y/n ----
730 `
740 C$=INKEY$:IF C$="" OR C$="" OR C$="" OR C$="" THEN PRINT
   C$:RETURN ELSE 740
750 `
760 REM ---- select question ----
770 `
780 I=INT(RND(1)*N)+1:IF (ANS(I,1)+ANS(I,2)) <=5 THEN 800
790 IF (INT(RND(1)*150)+1)/100 > (ANS(I,1)/ANS(I,2)) THEN 780
800 IF (MIX=1) THEN J1=INT(RND(1)*2)+1:J2=INT((1/J1)+1) ELSE J1=1:
   J2=2

```

We published a program called Drill.Bas during year one. If you are collecting this type of program on one disk, consider renaming this one Drill2.Bas so that you won't write over the previous Drill.Bas.

```

810 RETURN
820 `
830 REM ---- delay loop ----
840 `
850 FOR TM=1 TO 800:NEXT TM:RETURN
860 `
870 REM ---- initialize the answer ----
880 `
890 CR=0: FOR I=1 TO N:ANS(I,1)=1:ANS(I,2)=1:NEXT I:RETURN
900 `
910 REM ---- prepare for scores ----
920 `
930 FOR I=1 TO N
940   ANS(I,1)=ANS(I,1)-1
950   ANS(I,2)=ANS(I,2)-1
960 NEXT I
970 RETURN
980 `
990 REM ---- enter or edit a drill ----
1000 `
1010 TITLE$='ENTER OR EDIT':GOSUB 640:IF H$<>' ' THEN PRINT 'The
    current title is `';H$
1020 PRINT TAB(5);'Filename for the Exercise `':GOSUB 510:A$=IN$
1030 IF A$ <>' ' THEN H$=A$
1040 FOR I=1 TO MX
1050   GOSUB 640
1060   PRINT TAB(5);'Press ENTER to keep the same keyword'
1070   PRINT TAB(5);'or response':PRINT:PRINT
1080   A$=' ':IF A$(I,1) <> ' ' THEN PRINT 'present keyword is: `';
    A$(I,1)
1090   PRINT 'Enter keyword for drill or END: `':GOSUB 510:A$=IN$:
    PRINT
1100   IF A$ <> ' ' THEN A$(I,1)=A$
1110   IF A$(I,1)='END' OR A$(I,1)='end' THEN 1160
1120   A$=' ':IF A$(I,2) <> ' ' THEN PRINT 'Present response is: `';
    A$(I,2)
1130   PRINT 'Enter correct response: `':GOSUB 510:A$=IN$:PRINT
1140   IF A$ <> ' ' THEN A$(I,2)=A$
1150 NEXT I
1160 IF A$(I,1)='END' OR A$(I,1)='end' THEN I=I+1
1170 N=I-2:RETURN
1180 `
1190 REM ---- save a drill to disk here ----
1200 `
1210 PRINT 'Filename is `';H$;', use it (y/n)';:INPUT XX$
1220 IF XX$='y' OR XX$='Y' THEN FL$=H$:GOTO 1240
1230 INPUT 'What filename do you want to use `';FL$
1240 FL$=FL$+'.drl'

```

```

1250 OPEN ``O'',1,FL$
1260 PRINT #1,H$
1270 PRINT #1,N
1280 FOR I=1 TO N
1290 PRINT #1,A$(I,1)
1300 PRINT #1,A$(I,2)
1310 NEXT I
1320 CLOSE 1
1330 GOSUB 830:RETURN
1340 `
1350 REM ---- load a drill from disk ----
1360 `
1370 INPUT``What filename do you wish to load ``;FL$
1380 FL$=FL$+``.drl``
1390 OPEN ``I'',1,FL$
1400 LINE INPUT #1, H$
1410 INPUT #1,N
1420 FOR I=1 TO N
1430 IF EOF(1) THEN 1470
1440 LINE INPUT #1,A$(I,1)
1450 LINE INPUT #1,A$(I,2)
1460 NEXT I
1470 CLOSE 1
1480 RETURN
1490 `
1500 REM ---- run a drill ----
1510 `
1520 TITLE$= ``DRILL PERIOD``:GOSUB 640:GOSUB 870
1530 PRINT TAB(5);``During the drill period, you will be shown
1540 PRINT TAB(5);``a word or phrase from the drill file at
1550 PRINT TAB(5);``random. You should give the correct answer,
1560 PRINT TAB(5);``but if you can't, you can ask for help by
1570 PRINT TAB(5);``answering `.HELP`. The period is important!
1580 PRINT TAB(5);``(Beware - asking for help counts as a wrong
    answer)
1590 PRINT TAB(5);``If you want to end the drill, answer `.STOP`
1600 PRINT TAB(5);``and your score will be shown.
1610 PRINT:PRINT ``Press any key``;
1620 IF INKEY$=```` THEN 1620
1630 GOSUB 640
1640 PRINT TAB(5);``You can have the program ask only the
1650 PRINT TAB(5);``questions, or you can have it mix questions
1660 PRINT TAB(5);``and answers.``:PRINT:PRINT
1670 PRINT TAB(5);``Do you want the drill to mix questions and
1680 PRINT TAB(5);``answers (Y/N)? ``;:GOSUB 720
1690 IF C$=``Y`` OR C$=``y`` THEN MIX=1 ELSE MIX=0
1700 TITLE$=H$
1710 GOSUB 760:GOSUB 680

```

Address.Bas

```
1720 PRINT ``What is: ``;A$(I,J1);`` ?``
1730 PRINT:GOSUB 510:A$=IN$:PRINT
1740 IF A$=``.HELP`` OR A$=``.help`` THEN PRINT ``The answer is: ``;A$(I,
    J2):ANS(I,1)=ANS(I,1)+1:GOSUB 830:GOTO 1710
1750 IF A$=``.STOP`` OR A$=``.stop`` THEN 1800
1760 IF A$=A$(I,J2) THEN PRINT:PRINT ``That is correct``:ANS(I,
    2)=ANS(I,2)+1:GOSUB 830:GOTO 1770 ELSE PRINT:PRINT ``Not
    correct``:ANS(I,1)=ANS(I,1)+1:GOSUB 830:GOTO 1710
1770 IF ((ANS(I,1)/(ANS(I,1)+ANS(I,2))) < .3) AND ((ANS(I,1)+ANS(I,
    2)) > 5) AND (ANS(I,3) <> 1) THEN CR=CR+1:ANS(I,3)=1
1780 IF CR >= N THEN 1800 ELSE 1710
1790 `
1800 REM ---- display results ----
1810 `
1820 TITLE$=``SCORE BOARD``:GOSUB 640:GOSUB 910:GOSUB 410
1830 PRINT TAB(5);USING F1$;W;R+W
1840 PRINT:PRINT TAB(5);USING F2$;SC
1850 PRINT:PRINT ``Do you want to see a complete breakdown (Y/N)? ``;
    :GOSUB 720
1860 IF C$=``N`` OR C$=``n`` THEN RETURN
1870 FOR I=1 TO N
1880   GOSUB 640
1890   PRINT TAB(5);``Prompt:   ``;A$(I,1)
1900   PRINT TAB(5);``Reply:   ``;A$(I,2)
1910   PRINT:PRINT:PRINT TAB(5);USING F1$;ANS(I,1);ANS(I,1)+ANS(I,
    2)
1920   IF ANS(I,1)+ANS(I,2)=0 THEN 1950
1930   SC=(ANS(I,2)/(ANS(I,1)+ANS(I,2)))*100+.5:SC=SC-1
1940   PRINT TAB(5);USING F2$;SC
1950   PRINT ``Press any key ``;
1960   IF INKEY$=```` THEN 1960
1970 NEXT I
1980 RETURN
1990 `
2000 REM ---- quit the program ----
2010 `
2020 CLS:END
2030 `
2040 `----- error trap for file not found -----
2050 `
2060 IF ERR <> 53 THEN ON ERROR GOTO 0
2070 ` IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0
2080 PRINT``That file does not exist yet.``
2090 PRINT
2100 INPUT``Press ENTER ``;XX
2110 GOTO 240
2120 END `of program
```

Addbook.Bas

Make address booklets from Card.Bas files

Robert H. Fuller, East Canton, Ohio. Here is a clever little program to make address booklets from Card.Bas files. It prints both the front and back of the pages in the proper order so that you can bind it with a stapler. The author first wrote this for a Tandy Color Computer and then converted it for more general use. Conversions for other machines are either obvious or not necessary.

Here is a program that prints an address booklet from a Card.Bas file. I use a version limited to five fields per record. This program uses parts of Card.Bas and part of Pages.Bas (from Issue 12, page 15).

In line 310, L1 gives the total records. This is used in line 370 to get a rough page count. Line 400 adds needed blank pages to fill out the booklet. The separation of "front" and "back" is done in lines 430 and 440. The printout in lines 480 to 500 does not show actual pages, rather the page index number used to fetch addresses.

The routine at 610 has the variables II, I and DD, D switched to switch columns on fronts to match columns on backs. Both the 610 routine and the 740 routines print six blank lines for cutting space and form feed.

Because the blanks and the blank pages need to be printed, a problem arises in that the program will print past the end of the file. Rather than pad out the data file, causing possible future problems, the program examines each index, and if greater than the last record, prints a blank for that half line.

When phone numbers are not listed, two blank lines separate addresses, with phone numbers only one blank line can be used. Line 960 gives instructions for adding phone numbers.

This program was originally written on a Tandy Color Computer, and converted to a more general form of BASIC. It should run on most machines without modification.

```
10 REM * Addbook.Bas * address book using Card.Bas
20 REM * written for CodeWorks by Robert H Fuller
30 '
40 'reads Card.Dat files (using 5 fields per record)
50 '
60 'uses fanfold paper and prints half the addresses on one side
70 '
80 'paper must be removed and turned over and printed on the back.
90 '
```


Cardconv.Bas

```
100 `sheets can be cut in half, stacked, folded and stapled.
110 `
120 `
130 CLS
140 ` clear 10000 ` only if you need to
150 V=400 ` set limit of records here
160 DIM A$(V),B$(V),C$(V),D$(V),E$(V),Q$(V),P(V)
170 DIM B(V/5),F(V/5) ` index page variables
180 INPUT`What is the file name you will use ``;Y$
190 OPEN ``I``,1,Y$
200 L1=0
210 FOR I=0 TO 1000
220   LINE INPUT #1,A$(I)
230   IF A$(I)='`ZZZ`` THEN P(I)=I:GOTO 310
240   IF EOF(1) THEN 310
250   LINE INPUT #1,B$(I)
260   LINE INPUT #1,C$(I)
270   LINE INPUT #1,D$(I)
280   LINE INPUT #1,E$(I)
290   P(I)=I
300 NEXT I
310 L1=I-1
320 CLOSE 1
330 CLS
340 INPUT`Press enter to print ``;X
350 `
360 `puts 5 addresses per page
370 A=INT(L1/5):IF 5*A<>L1 THEN A=A+1
380 `
390 `determine number of book pages
400 AA=INT(A-4*INT(A/4)):IF AA<>0 THEN A=A+1:GOTO 400
410 `
420 `store book page indices
430 FOR J=1 TO A/2 STEP 2
440   F(J)=5*(J-1):F(J+1)=5*(A-J)
450   B(J+2)=5*J:B(J+3)=5*(A-J-1)
460   `
470   `book page indices
480   PRINT`          Front          Back`
490   PRINT J;`          `;
500   PRINT F(J);F(J+1);`          `;B(J+2);B(J+3)
510 NEXT J
520 `
530 PRINT`          Position printer paper now`
540 INPUT`Select F(ronts) or B(acks)``;Z$
550 IF LEFT$(Z$,1)='`F`` THEN 610
560 IF LEFT$(Z$,1)='`B`` THEN 740
570 GOTO 540
```

```

580 `
590 `prints fronts of sheets
600 `I and II are file numbers. D and DD are page numbers
610 FOR J=1 TO A/2 STEP 2
620   FOR WW=0 TO 4
630     II=F(J)+WW:DD=F(J)/5+1
640     I=F(J+1)+WW:D=F(J+1)/5+1
650     GOSUB 830
655   NEXT WW
660   `print page numbers
670   LPRINT TAB(12);D;TAB(60);DD
680   `print six blank lines for cutting space
690   FOR Q=1 TO 6:LPRINT'' ``:NEXT Q
700 NEXT J
710 GOTO 540
720 `
730 `prints back of sheets
740 FOR J=1 TO A/2 STEP 2
750   FOR WW=0 TO 4
760     I=B(J+2)+WW:D=B(J+2)/5+1
770     II=B(J+3)+WW:DD=B(J+3)/5+1
780     GOSUB 830
785   NEXT WW
790   LPRINT TAB(12);D;TAB(60);DD
800   FOR Q=1 TO 6:LPRINT'' ``:NEXT Q
810 NEXT J
820 GOTO 540
830 `
840 `print two columns
850 `names
860 IF I<=L1 THEN LPRINT A$(P(I));ELSE LPRINT'' ``;
870 IF II<=L1 THEN LPRINT TAB(45);A$(P(II)) ELSE LPRINT `` ``
880 `
890 `address
900 IF I<=L1 THEN LPRINT B$(P(I));ELSE LPRINT `` ``;
910 IF II<=L1 THEN LPRINT TAB(45);B$(P(II)) ELSE LPRINT `` ``
920 `
930 `city state and zip
940 IF I<=L1 THEN LPRINT C$(P(I));'' ``;D$(P(I));ELSE LPRINT `` ``;
950 IF II<=L1 THEN LPRINT TAB(45);C$(P(II));'' ``;D$(P(II)) ELSE
  LPRINT `` ``
960 `to add phone numbers unremark lines 970 and 980 and remark
  line 990
970 `IF I<=L1 THEN LPRINT E$(P(I));ELSE LPRINT `` ``;
980 `IF II<=L1 THEN LPRINT TAB(45);E$(P(II)) ELSE LPRINT `` ``
990 LPRINT `` ``
1000 LPRINT `` ``
1010 RETURN

```

Cardconv.Bas

A file conversion program for Card.Dat files

Staff Project. This program is the vanguard for a new and better Card.Bas program to be presented in the next issue. You probably won't appreciate it until then, but keep in mind that it's here.

The most popular program we ever published, by far, has to be Card.Bas. It's five years old now and it may be time to take the "training wheels" off of it. This may be a little like putting the cart before the horse but that's the way the space worked out for this issue.

Without going into the whys and wherefores (we'll save them for next issue), let's just say that Card.Bas had the fields hard coded into it. This, of course, meant that you had to reprogram to change the fields and then use a different version of the program depending on the data file fields.

Card2.Bas will change all that and a few other things, too. With it, you will be able to run any of your files (your old ones, too) with just one program. To do that, we will be putting the field information into record zero of the .Dat file. But there is already something in record zero, so we need to move it to the end of the file so that we won't lose anything. While we are at it, we can get rid of that pesky ZZZ that we used for an end of file marker. BASIC supplies its own end of file when you save, and the ZZZ was there mostly as an educational device. Now that we all know how it works, let's get it out of there.

Cardconv.Bas is a short program that will do what needs to be done to the file to make it compatible with Card2.Bas. You won't want to use it, though, until you get Card2.Bas in the next issue. (Just to show that we aren't teasing you into renewing your subscription, we will include Card2.Bas on the year 4 diskette. It's done and ready to go, there just isn't space in this issue for it.)

If you have a Card.Dat file with more than 600 records, you will need to change the 600 in the

DIM statement in line 140. Don't worry about the 10 that is there now, it gets set to some other number in line 350 anyway.

We ask for the file name and how many fields each record has in lines 340 and 350. Then we open the file and read it into memory in lines 390 to 480. Next, we check to see if it was, indeed, a Card.Dat file by checking for the ZZZ or zzz in the file. Then, in lines 610 to 660, we move record zero to the end of the file. That overwrites the ZZZ sentinel and doesn't replace it with anything because we no longer want that there. You don't lose a record by doing this, because NR, the number of records, included ZZZ as the last record, into which we will now put the information from record zero. After you get the file into Card2.Bas, you will want to sort it again to get that record into the proper sequence.

Having moved record zero to the end, we now want to write a new record zero with the field names associated with the file. This happens in lines 700 to 770. Then we save the file back out to disk in lines 780 to the end of the program. Just to be on the safe side, don't name your output file the same as your input file, at least not until you are sure that the converted file will work with Card2.Bas. If you try to convert a converted file, you will get program errors.

Just for the heck of it, we did this program in QuickBASIC too, and the listing is provided for your information. There really isn't that much difference, except for the absence of line numbers. We will present Card2.Bas in the next issue as a feature, to make sure that it gets all the space it deserves.

```

100 REM * Cardconv.Bas * A file conversion program * 24 Jun 89
110 REM * (206) 475-2219 * Please do not remove these credit
    lines.
120 REM * (c)1989 80-NW Publishing Inc. & placed in public domain.
130 REM * 3838 South Warner St. Tacoma, WA 98409
140 DIM R$(600,10)
150 `
160 CLS
170 PRINT STRING$(22,45);'' The CodeWorks ``;STRING$(23,45)
180 PRINT''          C A R D F I L E C O N V E R S I O N
190 PRINT''          converts old Card files to new format
200 PRINT STRING$(60,45)
210 PRINT
220 PRINT''          This program will take files created with
230 PRINT''Card.Bas and will convert them to a new format for
240 PRINT''use with Card2.Bas. It will move record zero to a
250 PRINT''position after the last record in the file, over-
260 PRINT''writing the ZZZ there now. It will then prompt you
270 PRINT''to enter field names and put them into record zero.
280 PRINT''You can name the new file differently if you wish.
290 PRINT''Don't run this program on a file already converted
300 PRINT''by this program.
310 PRINT TAB(15);''Press any key''
320 K$=INKEY$:IF K$=''' THEN 320
330 `
340 INPUT''What is the name of the Card file to convert ``;F1$
350 INPUT''How many fields does each record have'';NF
360 `
370 ` open the file and read it in
380 `
390 OPEN ``I'',1,F1$
400   FOR I=0 TO 600
410     IF EOF(1) THEN 460
420     FOR J=1 TO NF
430       LINE INPUT #1, R$(I,J)
440     NEXT J
450   NEXT I
460   NR=I-1
470 CLOSE 1
480 PRINT''The file currently has ``;NR;'' records.
490 `
500 ` check to see if the file is already converted
510 `
520 FOR I=1 TO NR
530   IF R$(I,1)='''ZZZ'' OR R$(I,1)='''zzz'' THEN 610
540 NEXT I
550 PRINT''The file is either not a Card file or
560 PRINT''has already been converted.
570 END
580 `
590 ` move record zero to the end of the file
600 `
610 PRINT''Moving record 0 to the end of the file.''

```

```

620 PRINT "Removing the ZZZ sentinel by writing over it."
630 FOR J=1 TO NF
640   R$(NR, J)=R$(0, J)
650 NEXT J
660 PRINT
670 '
680 ' prompt for entry of field names
690 '
700 PRINT "Get ready to enter field names for the file"
710 PRINT "Press ENTER to quit adding field names."
720 PRINT "Field names can be no longer than 10 characters."
730 FOR I=1 TO 10
740   PRINT "Name of field "; I; : INPUT R$(0, I)
750   IF R$(0, I)="" THEN 770
760 NEXT I
770 PRINT
780 PRINT "If you name the output file the same as the input
790 PRINT "file you won't get another chance to do it right."
800 INPUT "What will you name the output file"; F2$
810 '
820 ' write the converted file back to disk
830 '
840 OPEN "O", 1, F2$
850   PRINT #1, NF
860   FOR I=0 TO NR
870     FOR J=1 TO NF
880       PRINT #1, R$(I, J)
890     NEXT J
900   NEXT I
910 CLOSE 1
920 PRINT "Conversion complete."
930 END 'of program

```

REM * Cardconv.Bas * written for QuickBASIC 4.5

```

DIM r$(600, 10)
CLS
PRINT STRING$(22, 45); " The CodeWorks "; STRING$(23, 45)
PRINT "          C A R D   F I L E   C O N V E R S I O N"
PRINT "          converts old Card files to new format"
PRINT STRING$(60, 45)
PRINT
PRINT "          This program will take files created with"
PRINT "'Card.Bas and will convert them to a new format for'"
PRINT "'use with Card2.Bas. It will move record zero to a'"
PRINT "'position after the last record in the file, over'"
PRINT "'writing the ZZZ there now. It will then prompt you'"
PRINT "'to enter field names and put them into record zero.'"
PRINT "'You can name the new file differently if you wish.'"
PRINT "'Don't run this program on a file already converted'"
PRINT "'by this program.'"
PRINT TAB(15); "Press any key"
DO UNTIL INKEY$ <> "": LOOP

```

```

INPUT "What is the name of the Card file to convert "; f1$
INPUT "How many fields does each record have "; nf
REM * open the file and read it in
OPEN "I", 1, f1$
FOR i = 0 TO 600
  IF EOF(1) THEN GOTO fclose
  FOR j = 1 TO nf
    LINE INPUT #1, r$(i, j)
  NEXT j
NEXT i
fclose:
nr = i - 1
CLOSE 1
PRINT "The file currently has "; nr; " records."
REM * check to see if the file is already converted
FOR i = 1 TO nr
  IF r$(i, 1) = "ZZZ" OR r$(i, 1) = "zzz" THEN GOTO ok
NEXT i
PRINT "The file is either not a Card file or"
PRINT "has already been converted."
END
ok:
PRINT "Moving record 0 to the end of the file."
PRINT "Removing the ZZZ sentinel by writing over it."
FOR j = 1 TO nf
  r$(nr, j) = r$(0, j)
NEXT j
PRINT
REM * prompt for entry of field names
PRINT "Get ready to enter field names for the file"
PRINT "Press ENTER to quit adding field names."
PRINT "Field names can be no longer than 10 characters."
FOR i = 1 TO 10
  PRINT "Name of field "; i; : INPUT r$(0, i)
  IF r$(0, i) = "" THEN GOTO ok1
NEXT i
ok1:
PRINT
PRINT "If you name the output file the same as the input"
PRINT "file you won't get another chance to do it right."
INPUT "What will you name the output file "; f2$
REM * write the converted file back to disk
OPEN "O", 1, f2$
PRINT #1, nf
FOR i = 0 TO nr
  FOR j = 1 TO nf
    PRINT #1, r$(i, j)
  NEXT j
NEXT i
CLOSE 1
PRINT "Conversion complete."

END 'of program

```

NFL89

NFL89.Bas and Stat89.Bas for the current season

After a moderately successful year last year, our Oracle is ready to try again. Don't forget that you can't make any projections until after the week 3 stats are in. From the look of things, this year's NFL schedule should be an exciting one. If you have last year's NFL88.Bas all you need to change is the data statements at the end of the program and the name, from NFL88 to NFL89.

```
100 REM ** NFL89.BAS * NFL PROJECTION PROGRAM * CODEWORKS MAGAZINE
    *
110 REM ** 3838 S. Warner St. Tacoma,WA 98409 (206)475-2219 VOICE
120 REM ** Requires a data file made
130 REM ** with the accompanying program STAT89.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 89-90 NFL schedule for weeks 4 thru 16
1440 DATA 16,18,22,28,26,21,15,23,19,17,24,25,20,9
1450 DATA 27,5,14,8,3,6,4,2,12,11,10,7,1,13
1460 DATA 18,19,22,20,21,15,27,26,23,16,28,24,25,17
1470 DATA 14,12,6,10,2,8,9,7,13,11,4,3,5,1
1480 DATA 12,18,15,22,20,21,19,26,23,6,28,25,16,14
1490 DATA 17,13,24,27,11,2,9,10,8,7,1,4,3,5
1500 DATA 17,18,19,22,6,21,26,24,20,23,2,28,25,3
1510 DATA 8,15,16,11,4,27,14,5,7,9,13,12,10,1
1520 DATA 15,18,10,22,23,21,3,26,16,19,28,20,11,17
1530 DATA 27,24,14,13,12,6,5,2,9,8,7,4,1,25
1540 DATA 18,14,22,25,21,10,20,26,9,23,19,15,24,28
1550 DATA 17,16,3,27,6,8,2,1,12,7,11,13,4,5
1560 DATA 19,18,22,23,21,24,26,28,25,27,15,17,13,16
1570 DATA 6,20,14,11,2,5,8,9,4,12,7,10,1,3
1580 DATA 18,16,9,22,28,21,26,1,25,23,17,19,15,2
1590 DATA 27,20,24,4,13,14,10,6,8,11,5,12,7,4
1600 DATA 22,18,21,9,24,26,23,28,27,19,16,25,14,17
1610 DATA 6,1,3,2,7,8,12,13,4,11,10,5,20,15
1620 DATA 18,24,22,21,26,25,23,20,19,16,15,28,17,27
1630 DATA 11,14,6,7,12,2,13,9,8,10,3,4,1,5
1640 DATA 13,18,24,22,21,19,4,26,10,23,28,8,5,25
1650 DATA 16,15,20,17,27,1,14,7,9,6,2,3,11,12
1660 DATA 18,11,23,22,7,21,26,5,15,19,27,28,16,20
1670 DATA 17,12,25,24,1,14,8,6,2,4,10,9,3,13
1680 DATA 18,17,22,7,21,23,26,27,19,13,28,15,25,4
1690 DATA 12,16,20,10,1,24,9,14,6,11,8,2,5,3
1700 END ' of 1989-90 schedule data

```

Stat89.Bas

```
100 REM * STAT89.BAS * CODEWORKS MAGAZINE * 3838 S. WARNER ST.  
    TACOMA WA.  
110 REM * 98409 (206) 475-2219 VOICE  
120 REM * Maintains the stats for NFL89.  
130 REM * If no file exists then in command mode,type OPEN''O'',1,''STAT  
    AT.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(448,5),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=448  
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 Ll=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$(23,`-`)
530 PRINT`      N F L   W E E K L Y   S T A T I S T I C S`
540 PRINT`      Maintains statistics for 1989-90 NFL Football`
550 PRINT STRING$(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.``

```

Handy Order Form

```
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

```

Handy Order Form

- RENEW SUBSCRIPTION:**
Nov/Dec 89 through Sep/Oct 90 _____ **\$24.95**

- All 4th year issues:**
Nov 88 through Sep 89 _____ **\$18.00**

- All 3rd year issues:**
Nov 87 through Sep 88 _____ **\$18.00**

- All 2nd year issues:***
Nov 86 through Sep 87 _____ **\$18.00**

- All 1st year issues:**
Sep 85 through Sep 86 _____ **\$18.00**

- DISKS (specify year and computer type) _____ \$15.00**
4th year disk will be ready Sep 1, 1989 Year(s) _____

- "Starting with MS DOS" booklet _____ \$7.00**

*Note
new
lower
prices
on back
issues
and all
disks!*

Postage and handling charges already included.

Diskettes are available for MS DOS, Tandy IV, Tandy III and most CP/M formats ***Please specify your computer type!***

*In year 2 issues, Issue 8 is out of print and will be supplied on diskette. Please specify your computer type if ordering 2nd year issues.

Computer type: _____

- Check/MO enclosed
- Charge to VISA/MC _____ Exp _____

Name _____

Address _____

City/State/Zip _____

Clip or photocopy and mail to: **CodeWorks, 3838 South Warner Street,
Tacoma, Washington 98409**

**We accept VISA & MasterCard. You may call in your order:
(206) 475-2219 Thank you.**

989

Index

and things that won't fit elsewhere

Flow.bas, correction, issue 24, page 4
Beginning BASIC, part 2 of all about strings, issue 24, page 5
Tracker.bas, main program, issue 24, page 8, mutual fund tracker
Rcard1.bas, main program, issue 24, page 25, updates Rcard.bas
Basic Techniques, article, print numbers in text lines, issue 24, page 32
Notes, preventing dupes with random, issue 24, page 35
Etax89.bas, updates to Etax88.bas, issue 24, page 36
AI, article, part 1, issue 24, page 37
Download, final notes on download, issue 24, page 40

Although this may be along the lines of our old Puzzlers, we won't count it as one. This really happened and was interesting enough to pass along.

A customer wanted labels for his parts bins to be printed on our computer. The numbers of the

parts started at one million and went from there to 1,050,000. In addition, he wanted an eighth digit appended to the end of the number. This digit was what you get when you divide the base number by seven, throw away the quotient and keep the remainder.

In other words, the number 1000000 would have the digit 1 appended (10000001) because that's the remainder when you divide 1,000,000 by seven. $1000000 \text{ MOD } 7$, in computer talk.

The whole idea for this appended digit was as a check digit, so that when people were keying in the part number, the input program would strip the last digit, divide what was left by seven and see if the check digits matched.

Well, we said, "sure, no problem, that's what computers are for." And knowing all too well that exponential numbers don't look good on parts bins, we set about to make his labels.

Did we bite off more than we could chew? Try it for yourself and see. More on this next time.

CodeWorks

3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
U.S. Postage
PAID
Permit 774
Tacoma, WA



ERICKSON, MIKE / KRJB 2910
BOX 250
MONTE RIO CA 95462

CODEWORKS



Issue 26

Nov/Dec 1989

CONTENTS



| | |
|---------------------------------|----|
| <i>Editor's Notes</i> | 2 |
| <i>Forum</i> | 3 |
| <i>Beginning BASIC</i> | 8 |
| <i>Things</i> | 10 |
| <i>Card2.Bas</i> | 11 |
| <i>Ckrite.Bas</i> | 28 |
| <i>Progest.Bas</i> | 32 |
| <i>Renewal/Order Form</i> | 39 |
| <i>Index</i> | 40 |



Editor/Publisher

Irv Schmidt

Associate Editor

Terry R. Dettmann

Circulation/Promotion

Robert P. Perez

Editorial Advisor

Cameron C. Brown

Technical Advisor

Al Mashburn

(c) 1989 80-Northwest Publishing Inc. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this publication, the publisher assumes no responsibility for errors or omissions. Unless otherwise noted, all programs presented in this publication are placed in public domain. Please address all correspondence to CodeWorks, 3838 South Warner Street, Tacoma, Washington 98409

Telephone
(206) 475-2219

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 is \$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. VISA and MasterCard orders are accepted by mail or by phone (206) 475-2219.

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in the USA. 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

So whatever happened to APL, Algol, Snobol, PL/1 and the other "high performance" languages of the past? And how come lil' ol' BASIC is still in there, slugging?

Well, it helps a language a whole lot if for the first 10 or 15 years of its life it comes free with the computer you buy. That's probably one of the most compelling reasons that BASIC is alive and well today. Not only is it alive and well, it's growing. It has been estimated that more people program in BASIC than in any other language.

BASIC is 25 years old this year. It was developed in 1964 at Dartmouth College by John Kemeny and Thomas Kurtz. In those days, it ran on college mainframes. Personal computers in those days were still a science fiction dream. A little over 10 years later, the MITS corporation in Albuquerque, New Mexico developed the first personal computer, the Altair. Popular Electronics Magazine featured it on their cover. It was just a case, because there weren't enough "guts" to go around at that time, although no one could tell the difference because no one had ever seen one before.

Bill Gates and Paul Allen were Harvard students in those days, and decided that the Altair needed a programming language. Bill Gates wrote the Altair 4K BASIC. He and Allen

left school and became the software department of MITS. Bill later said that "the 4K BASIC was the best program I ever wrote."

They left MITS sometime after that and started Microsoft in Albuquerque. In early 1979 they moved Microsoft to Bellevue, Washington. I knew Terry Dettmann in those days, and he told me he was going to visit them shortly after they moved, to get a story about them. He told me he found them in a two-room office, with no furniture except for a telephone and a secretary in blue-jeans, sitting on the floor answering calls. They grew and prospered. The times were exciting.

Just last month, Bill Gates finally came out and said the GW BASIC stood for "Gee Whiz" BASIC. It's a far cry from his early 4K BASIC. His Quick-BASIC is a much larger step into the sophisticated world of programming languages. It will probably still be around in another 25 years.

In the meantime, it has given us a lot of pleasure, solved a lot of problems, and let us write magazines about it. It's been good to us, and we thank the professors at Dartmouth and the people at Microsoft for a great language. Long live BASIC, in whatever form it decides to take!

Irv

Forum

An Open Forum for Questions and Comments

...I have been looking for references to routines in either FORTRAN or BASIC which evaluate calculated statistics or which return the critical value of a statistic when given the appropriate parameters. I want to incorporate such algorithms into existing programs and also, when they are sufficiently short, into spreadsheets. In the days of second and third generation mainframes, there were books of approximations and exact calculations; but, no one knows anything about those works now. Can you advise me of any locations or sources where I might find what I am looking for?

William L. Lieberman
Abbott Park, IL

Try your local library under Computing Science, Statistics, or Economics. Without knowing exactly what you are looking for, it's hard to tell just which one of these departments will yield the answers. The idea does sound rather interesting, however.

This is a copy of the last paragraph of a letter I mailed to you in August, 1986: "I just finished keying the Plist.Bas program from your Issue 6. As the author suggested, I also added the time and date to the headers using the DATE\$ and TIME\$ functions. The header of each page now shows the date and

time the printout started. I would prefer to use the creation date and time that is stored in the directory, but I don't know how to extract it from the diskette. Would you please show me how to do this?"

The letter I received from you said you didn't know how to do this and you would also like to know if it could be done.

Attached is a printout of how I capture time and date in BASIC.

G.W.T., FL

```
100 REM * gwt.bas *
110 REM * program from dir must be in scratch1.lst
120 '
130 OPEN"scratch1.lst"FOR INPUT AS #3
140 FOR I=1 TO 5
150 INPUT #3,DR$
160 IF I<>5 THEN 210
170 ' added for dos 4.0 header line: Volume Serial Number is
XXXX-XXXX
180 IF MID$(DR$,2,9)=" " THEN LINE INPUT #3, DR$
190 DA$=MID$(DR$,24,10)
200 TI$=DA$:MID$(TI$,1,6)=MID$(DR$,34,6)
210 NEXT I
220 CLOSE #3:KILL"scratch1.lst"
230 MID$(DA$,3,1)="":MID$(DA$,6,1)=" "
240 IF MID$(TI$,6,1)="a" THEN MID$(TI$,6,3)=" am" ELSE
MID$(TI$,6,3)=" pm"
250 PRINT DA$,
260 PRINT TI$
```

We puzzled over this for some time, and then finally realized that the program in question must somehow get into the file "scratch1.lst." It should be obvious that this will work only for MS-DOS. At the DOS prompt, you can enter the following: DIR

filename > scratch1.lst <ENTER>. This will write the filename and its associated length, time and date stamp into the scratch file. Now when you run this program it will read the time and date of the file's creation and then kill the scratch1.lst file. You don't have to be in DOS, either. From BASIC in command mode, you can type: SHELL DIR filename > scratch1.lst <ENTER>, and then type EXIT to get back to BASIC. Very interesting appli-

cation. Thank you.

I wasted a whole afternoon trying to figure this out. I know it's got to be incredibly easy but it escapes me. How do I write my program to print five labels for one person, then go on to

write five labels for the next, etc.? I can get it to write five labels, but not five for each. I don't know the loop to put inside of the first loop. Enclosed is my program. Thanks for your help.

Mrs. Ira Hynes
San Francisco, CA

```
100 DATA Betty Smith,111
Street,City St Zip
110 DATA Joe Brown, 222
Street,City St Zip
120 DATA Joe Magarac,333
Street,City St Zip
130 '
140 FOR I=1 TO 3 '3 is number
of names in data statements
150 READ A$,B$,C$
160 FOR J=1 TO 5 ' 5 is
number of times to print each
label
170 LPRINT A$
180 LPRINT B$
190 LPRINT C$
200 FOR K=1 TO 3 ' to skip
lines to next label
210 LPRINT " "
220 NEXT K
230 NEXT J
240 NEXT I
```

Assuming that you had your names and addresses in data statements, as you indicated you did, the above program will do it. Note yet another inner loop that spaces to the next label. If you want to print an "attention" line on the label as well as the names and addresses, then reduce the inner K loop by one since you will be printing four lines of address. Standard labels are one inch, or

six lines, from beginning of one label to the beginning of the next.

It is often desirable to check whether your printer is ready to accept output before sending data to it. Various combinations of printers and MS DOS computers result in different values being sent to different ports when the printer is ready to accept data. The short program listing below determines which port is being used and what value is being sent to it. The Epson FX80 sends 217 to PORT 889 when the printer is ready. Your printer/computer combination may produce different results.

Robert Hood
Bremerton, WA

```
10 CLS:LOCATE 6,15:INPUT"Prepare printer for output and
press ENTER";T$
20 DEF SEG=&H40:PORT=PEEK(8)+PEEK(9)*256+1
30 PRINT:PRINT
40 PRINT"If your printer is ready, it is sending";INP(PORT);"to
port";PORT;"."
50 PRINT
60 PRINT"You may use the statement below to check printer
status from a BASIC program."
70 PRINT:PRINT
80 PT=PORT:VL=INP(PORT):PT$=STR$(PT):VL$=STR$(VL)
90 PRINT"110 if inp(;"MID$(PT$,2);"=";MID$(VL$,2);" then
print:print"CHR$(34) "Your printer is ready."CHR$(34)
100 PRINT
110 IF INP(889)=223 THEN PRINT:PRINT "Your printer is
ready.":GOTO 140
120 IF INP(889)<>223 THEN PRINT:PRINT"Change line 110
in this program to agree with above."
130 PRINT:PRINT:EDIT 110
140 PRINT:PRINT:END
```

Thank you, Robert. As you can see, our Star SG-10 sent back something else, but it still works

fine.

In the Sep/Oct 1989 issue of CodeWorks in the NFL89.Bas program on page 34 there is an error in the schedule. On line 1590 you have entered the number 4 twice. The last four should be a three.

I think a subroutine should be developed for checking the accuracy of the data statements. This can be done by summing up the totals of data for each week of schedule, when correct the sum should be 406. I have one incorporated in my program but I program in TURBO

Pascal and I don't think it would be of much use to you. I also think you should revise your program for the changes

in 1989, with the four preseason games and the 16 regular season games. You should include in some future issue the diagram of the NFL structure describing the NFC, AFC, divisions, team names and numbers as you did in Issue 19, page 13. New subscribers as well as old subscribers would find it quite useful. I know that I refer to it quite often for both programming and in data entry.

Stephen Chubriilo
Pueblo, CO

You are right about the last four in line 1590. It should be a three. Unfortunately, all our yearly disks for all models now contain this error and should be corrected. We checked, but not quite close enough. We would have liked to include the diagram of the NFL again this year, but there just wasn't enough space in that issue. In any case, there were no changes since last year, and last year's diagram will work this year.

When we designed NFL.Bas back in 1986 we found that preseason games simply had no bearing on what the teams would actually do in the regular season. We would have liked to be able to start right off at the beginning of the season and make meaningful predictions, but it didn't work out that way.

I enjoy your publication, otherwise I would not renew my subscription. I've been with you from day one. However, I would offer a word of advice about future issues. You should expand the scope of your material to include information (hints

and examples) on the popular commercial packages such as LOTUS, dBASE IV, etc. I am not a "hacker" but I do program in BASIC if required. My main effort is using packaged software, which can be quite challenging as well. Input and guidance on their use would be most appreciated and practical. Your booklet on MS DOS is a step in the right direction...

L.V. Beckman
Mission Viejo, CA

Thank you for your comments. Do you want to know the truth? We aren't experts in any of the type of packages you mention. Nor could we afford to buy them all and become experts. Our goal with CodeWorks was, and still is, to enlighten, entertain and educate in BASIC. We picked that goal because no one else apparently had. We'll expand into QuickBASIC shortly, and Terry is currently writing another booklet on disk drives. I could write a book on using PageMaker and perhaps Wordstar, but those have already been done many times by others, and well too. Our philosophy is that if anything has a 51 percent chance (or better) of success it's worth trying. We don't see that percentage in reviewing package programs.

...As a matter of interest, I put your solution to my problem on simulating flash cards to good use. (See Issue 24, page 35). This summer I attended an Elderhostel at Vernon, Canada and one course was Sign Language. The instructor made a

statement that computers were useless in teaching sign language. I then typed out your program for her with a few changes and she was really enthusiastic and said it would be a wonderful teaching tool. I made simple changes and actually gave her two programs. One that threw the words or letters on the video every three seconds and then for sentences I made a new sentence come up after they hit any key. She intends to use it in all her classes.

James C. McCord
Fairbanks, AK

Nice to see it work, and thanks for letting us know.

I have disks 1 to 3 for TRS-80 Model III. Can those programs be converted to MS DOS with one of the conversion programs?

Hans E. Froelke
Sandy, UT

Not with any of our published conversion programs. However, you can use a program called TRSCROSS, available from Powersoft, 17060 Dallas Parkway, Suite 114, Dallas, TX 75248, to make the conversions. Or, if you still have your Model III, you can connect it with your MS DOS machine via RS-232 and send the Model III programs back to the MS DOS machine. Then, you will only need to change the change lines listed in each issue back to GW BASIC.

...I need help in finding a particular disk: The ARRANGER

II, by Dan Foy, either converted or re-written for MS DOS machines. This disk was originally written for the TRS-80 (et al) machines. I don't know if this disk does exist, but if it's out there, I would like to purchase a copy. Any help will certainly be appreciated. Rem * I don't know if your Forum column was designed to help find lost articles. If not, please overlook this, and again thanks for your valued publication.

R. H. Saunders
Epping, NH

If anyone knows the answer to this one, please let us know and we'll pass it along.

As I compose this letter, I am mindful of patience as a virtue. I have lost my Model III forever, it seems. It is replaced by an XT Clone. This means no old files to call up to start a new letter, and no idea of what is in the old files. It also means, alas, a new word processor program to get used to. You might be able to help, though. I feel that it should be possible to write a BASIC program to read directories made by DOSPLUS* 3.4 and save the file on an MS DOS disk. I have no MS DOS manual, since the computer supplier merely handed me a backup disk with it.

As this beast (we have not made friends yet) has 640K of RAM, it should be able to slurp up an old format disk totally, tell me to install a new formatted disk in drive B and strike any key. I have noticed ads for such a program in the PC

magazines, but feel reluctant to part with \$80 or more...

Thomas B. Habecker
Ridgefield, WA

It would be nice if we could do that, but the drives in your XT Clone will not read the DOSPLUS format on the disks. Computing is like that, sometimes. What it takes, probably in machine code, is a program that reads a disk in "primitive" mode and copies it a bit at a time onto another diskette. In spite of the cost, a commercial program to do it, if available, would certainly be the alternative to trying to write such a program in BASIC.

I have been using your Payroll.Bas (Pay2.Bas) program for a couple of years in my two small businesses, and it has been working superbly. However, I have one question: At the present time the program will take care of up to ten employees, which up to this time has been quite adequate. I am contemplating merging my two businesses into one, which will bring the total employment to at least 14, possibly more as time goes on. Can you suggest any change that can be made in this program that will allow more than ten employees? I tried adding more and they were ignored.

I realize that I could get one of the store-bought programs that would do somewhat more, but because I have created several other programs for use in the businesses in good old GW BASIC, I try to stay in that

language for convenience. Although BASIC is slow (comparatively), I find it fast enough for what I want, and the program can be altered to fit the situation easily and quickly...

Francis C. Williams
Honolulu, HI

It's our problem. We forgot to change one variable when we improved the PAY2.BAS program. In line 2180 it says FOR I=1 to 10, it should be FOR I=1 to NE. That will do it - now you can get more than 10 employees in. Again, this error is probably on all our yearly disks, and should be changed if you intend to use the program for more than 10 people. Thanks for bringing this to our attention.

I have converted to QuickBASIC 4.5 and it is great. Please do more with it.

David R. McCord
Redmond, WA

We intend to.

...I seem to have a problem with Cwindex.Dat and Qkey.Bas. I load Qkey and try to access Cwindex.Dat and nothing happens. I just can't seem to bring up any of the .Dat files. I suppose I need a little "one, two & three" hand holding instructions. I'd appreciate any suggestions.

David H. Smith
Edinburg, TX

We'd be glad to help, but need more information to go on. Do the .Dat files actually exist in the first place? Are you trying to

Beginning BASIC

load the default file when it doesn't yet exist? Did you type Qkey in or are you using it from our yearly diskette? Tell us more - we'll try and help.

Here are some other patches I've found for TRSDOS 1.3 (Model III):

PATCH *0
(ADD=43C3,FIND=2F,CHG=30)
PATCH *0
(ADD=4FB8,FIND=28,CHG=38)
This PATCH lets you use the decimal point instead of the slash when entering the date at bootup. That way, you can use the numeric keypad for date entry.

PATCH *0
(ADD=4EFE,FIND=215451,CHG=C32E4F)
This PATCH will disable the time prompt at bootup.

PATCH *2
(ADD=4ED4,FIND=20,CHG=18)
PATCH *5
(ADD=52EB,FIND=CB,CHG=36)
PATCH *5
(ADD=52ED,FIND=BE,CHG=00)
This PATCH will disable password checking in case you want to remove CONVERT/CMD or XFERSYS/CMD for more room on your disk or something like that. To reverse this, you only have to exchange the FIND=XX and CHG=XX numbers in the three patches. Out of curiosity, do you have any idea how many of your readers have and use TRS-80 Model 3 or 4?

Pat Chong
Las Cruces, NM

A lot less than a couple of years

ago. We are now around 75 percent MS DOS, with the remainder being mostly Model III and IV and a few CP/M machines. Although our coverage will shift to reflect these changes, we still won't give up entirely on these machines, but you can expect more MS DOS specific material in the future.

...Does there exist a subroutine in BASIC that will convert numerals to alpha? For example, in printing out a check, in addition to putting the dollar amount in numerals, is there any way to also convert those numerals into words? It seems that somewhere, in my dim past, I saw such a program, or

was I just dreaming?

Francis C. Williams
Honolulu, HI

We thought there was one, too. But we couldn't find it, and so we wrote one from scratch. At this time I don't know yet if it will fit into this issue, but expect it soon in any case. And thanks, it was a fun project.

Renewal time always brings a wealth of good letters and ideas. Thank you. To all of you whose letters were not answered, hang in there. We'll get around to them soon. Enjoy the fall weather, football and the turkey, and we'll see you again around the end of the year.

Irv



We're going to use our new computer to try and understand our old one.

Beginning BASIC

On Programming

For the past four years we have talked about the tools available in BASIC used to write programs. Those tools were the commands and functions, and how to use them. The only way to get to know them is to use them on a regular basis. What we haven't talked about, up until now, is the concept of programming. What goes into a program? How do you start? Why use one method instead of another? In short, how do you get the job done with the least code and hassle?

With this issue, we'll start a short series designed to answer those questions, and perhaps give you some insight into what it takes to put together just about any program. Most of us just start playing around and eventually end up with something reasonable, but not usually well thought out or optimized. It's fun, sometimes, to do that just to see what it leads to. But most of us do not have the time for such a haphazard approach. Like most things in life, we need it now - and it's got to work right the first time.

There are always three parts to any program. Repeat: three parts, always. Those three parts are: Input, Processing, and Output. You take something as input; you do something to it, and then you present it in some fashion as output. Try and think of a situation where this is not true. Granted, the input may come from a disk file, or from data statements in the program; but it's still input. The processing portion is usually straightforward, you do something to the input data, even if it's just moving it from one place to another. Naturally, you want to see some results when you are done, either on the screen or on your printer. Sometimes, however, the output may simply be sent to another disk file. It doesn't matter, even if it goes to a disk file, it's still a form of output. Even in simple game programs, you input your decisions, the program processes information based on your input (usually with the

random function) and the output is presented to you via the screen as a new game situation. It is always true that any program must possess these three parts. If any of the three is missing, you don't really need a program.

Input

Let's take input first, since that is the logical way things happen. Input can come from your keyboard. Your program can present you with input prompts which ask for the required information, or at least part of the required information.

Input can also come from a disk file, where you would take the data there and process it into another form. You could also have a combination of inputs where most of it came from a disk file and you supply more via the keyboard in the way of making decisions based on the data from the file.

As we have already found out, data can also be hard-coded into the program itself, in the form of DATA statements. This, too, is a form of data input to a program, although such data is rather fixed in nature and not easy to change. DATA statements are usually reserved for such uses as months of the year, column headings for printed reports, and other data which is reasonably constant.

There is yet another form of input that most of us wouldn't even guess is input. That is the RANDOM generator which BASIC provides. Yes, this is a form of input too. Instead of you picking a number between 1 and N and entering it via the keyboard, it does the job for you. It's still input, though, because it is not hard-coded into the program and provides an input that is variable, and not fixed or pre-determined.

Card2.Bas

All input to a program should be verified, which is to say that it must fall within the bounds of what the program expects to see. With a disk file or with the random generator or DATA statements, this is not difficult to do. Problems with verification come mostly with keyboard input. You never know what operator will enter what into a program. Is the input numeric when it should have been a string or the other way around? Is the value entered within the range the program will, or can, accept? On string inputs, will the program accept either upper or lower case input, or will it hang up because you entered a lower case n when it was expecting upper case Y or N? The most insidious cases are where the program accepts wrong input and goes merrily on its way - and then gives you corrupted output - and you wonder why.

Input validation is almost a necessity in any good program. Check it as soon as it is input and ask for correct input if the data is outside the accepted bounds. In the way of user-friendliness (a terrible term to use, but there is no other), you should consider allowing the user to input data in as many forms as possible, and then converting that input to the acceptable form in the program itself. One example comes to mind: When asking for input of decimal amounts, as in interest rates, you can let the user input the number either as an integer or as a decimal amount. Then if the number is less than one, leave it alone, otherwise divide the number by 100.

Don't fall into the trap of saying that you are programming only for yourself and you will remember how to enter the data. Murphy's law says that you will forget, or that a friend will see the program and want a copy, and then think a lot less of you when he can't get it to run like you said it would! Act as though every program you write will be a best seller and take all the precautions necessary to keep the complaints and gripes to a minimum. George Henry's Old Grandmother should be able to run the program without help.

Processing

This is the part the computer does. Keep in mind that it will do pretty much what you tell it to, and if you tell it wrong, the output will be wrong. Depending on the size and scope of your program, check it with sample data that is both very small and very large. There's nothing so frustrating as having a program work well on 10 items and bomb on 100, especially when your whole design has to be changed to make it work right, a total re-write in other words. Make sure that your processing is valid for the range of input you have allowed for. As in our earlier example with interest rates: leaving out the decimal point would result in some impossible payments!

Another killer in this area are parentheses. In a math function a missing pair or misplaced pair of parentheses can cause you no end of trouble. It can turn into another case of no errors reported, but output that is totally incorrect. Does $2*5-3$ equal seven or does it equal four? It's easy to see when there are actual numbers there, but not when it is in the form of: $A*B-C$, because you don't equate the variable name letters to values, normally. (If you want this expression to equal four then it should be written: $A*(B-C)$, otherwise, if you want it to equal seven then it should be: $(A*B)-C$.)

Output

Here, at least, you get something to see, either on the screen or on your printer. This is usually the first indication that something in the program is not working right. Aside from the actual values that are output, you also have the task of formatting that data on the printed page or on the screen. In addition, you may well have computed $A*(B-C)$ properly in the processing part of the program, and then told your output to print a variable that does not even represent that value. In other words, validation is again necessary. Use simple data for a test, then use a hand calculator or pencil and paper to work the simple

data out manually and see if your output agrees. If your program has several IF...THEN branches in its computational section, make sure you test each case separately. Provide the input that will force the program into each of the conditional cases, and don't rely on just one pass through all cases. Try several different values with each case, making the values hit both high and low extremes.

In addition to output to the screen or printer, you may in some cases wish to send output to a disk file. This happens more often than you might expect, as when converting one type of file to another, or in modifying the data in a file and

writing out a new, updated file. In such cases, it is wise to also print the output to the printer or screen as the new file is being written. This allows you to see what is happening, and avoids the necessity of exiting BASIC and using DOS to view the file after it has been written. (It has been our experience that every time we do this, we forget to save the BASIC program first and consequently lose the latest edits to it.)

In the next issue, we will look in detail at how to plan a program, how to decide what data structure to use, and how to structure the flow in a top-down manner.

Things which may be of interest and help fill little spaces like this one.

```

100 CLS
200 REM - PROBABILITY PROGRAM
300 PRINT "WHEN CHOOSING R ITEMS FROM A SET OF N ITEMS THE"
400 PRINT "NUMBER OF POSSIBLE UNIQUE COMBINATIONS IS GIVEN BY"
500 PRINT "THE EQUATION:
600 PRINT "
700 INPUT "PLEASE ENTER N, THE TOTAL NUMBER OF ITEMS";N
800 INPUT "PLEASE ENTER R, THE NUMBER OF ITEMS YOU ARE CHOOSING";R
900 M=1:S=1
1000 FOR I=0 TO R-1
1100 M=M*(N-I)
1200 S=S*(I+1)
1300 PRINT M,S
1400 NEXT I
1500 PRINT "NUMBER OF POSSIBILITIES: ";M/S

```

With all the interest these days in the various lottery games around the country we thought it would be interesting to show another way to look at your chances. This program allows you to see how many ways a number of items can be picked from a larger set of items.

Not very encouraging, is it? Of course, if you win you don't really care what the odds were. Our thanks to Kristi Perez for the program.

Our little thing on the back page of the last issue brought bunches of mail. Before we get into it

let us say that we solved the problem very quickly by simply using QuickBASIC. It was the most expedient thing to do at the time and it got the job done. Here is how Bob Keegan did it:

```

10 FOR I=0 TO 49999!
20 N1$=STR$(I):LN=LEN(N1$)
30 CH=(I+1)-INT((I+1)/7)*7
40 N2$=N1$+CHR$(48+CH)
50 N3$=RIGHT$(N2$,LN)
60 N4$="10000000"
70 MID$(N4$,9-LN,LN)=N3$
80 PRINT N4$
90 NEXT I

```

(more *Things* on page 38)

Card2.Bas

An Updated Version of Card.Bas

Staff Project. Now that it's five years old, Card.Bas can use some refinements and updating. Now you can use just one version of Card2.Bas and load a variety of files with a different number of fields in each. Print formatting has been improved as well.

Card.Bas is five years old this month. It's been a workhorse of a program and we use it for lots of chores around the office and at home. Judging from your letters, it's been that way for you, too.

But Card.Bas has some shortcomings. Searching and editing are just not that smooth. It uses a discrete end of file marker that keeps getting in the way and has to be taken into account whenever you want to modify the code. Originally, it had a preposterous way of initializing the file, although that was fixed with a "field change" in later issues.

But the single most annoying thing about it was that the fields were all fixed in the program, and there was a string variable attached to each field of the record. Changing the number of fields meant reprogramming and going through and changing all those string variable field names. Even then, you had to use a specific version of Card.Bas with (and only with) its associated data file. How many versions of the program did you end up with? We certainly had a few, to say the least.

Digging into that code brought back some waves of nostalgia, but we did it anyway. Card2.Bas is the result, and we ran into some surprises and a couple of interesting challenges along the way. Basically, the objective was to get rid of the ZZZ sentinel; carry the field names in record zero of the data file; use double sub-scripted arrays to prevent the old A\$ through I\$ field names; provide a much better printer output module; and improve the user interface as much as possible.

But what will happen, we asked, if we sort the file when it has all the field names in record zero? Nothing, we found out. The field names always stay in record zero because when we sort or search the file we can start at record one and go to the end of the file, instead of starting at record zero. That one was easy enough.

And what about compatibility with files created under the old Card.Bas? Well, we had to fix that, too. Cardconv.Bas (in the last issue) is a program you can use to convert your files over to Card2.Bas. Naturally, any new file created with Card2.Bas will create the file in the proper form. You can also use a word processor to massage the old files into the new form, but that is more involved, and we wanted something that would work for everyone including those who (are there any?) don't have a word processor. We used a word processor for one file, though, just to see if it could be done, and it worked fine.

In Card2.Bas, variable NR stands for Number of Records; NF stands for Number of Fields; and R\$(x,x) is a double sub-scripted array variable, and is usually counted by loop integers I and J, as in R\$(I,J), where I is usually the record number and J is the field number within the record. We will use the integer array, P(I), to keep a pointer to each record so that when we sort we can sort P(I) and not the strings themselves (just like in Card.Bas). Something new to Card2.Bas is direct cursor positioning, so we can put the whole record on the screen and let you fill the blanks, which was another thing lacking in Card.Bas.

It might be interesting to see just how the data file is laid out, so here is an example of how it would look for a file with five fields:

```
5
Name
Address
City
St
Zip
John Jones
123 Oak Street
Anytown
NY
11234
Mary Cary
213 Elm Road
Sometown
CA
92123
.
.
.
Zelda Zealot
23 Purdue St
Zanesville
OH
43444
```

(The last record just ends. There is no ZZZ, the DOS supplies the End of File marker as needed.)

Did you notice that the first record, record zero, actually has six fields in it while the rest only have five? That's because the very first thing in the file is a number indicating the number of fields. We need that so we can set up the loop properly to read in the rest of the file as you will see in a bit.

We have just about covered the first 300 program lines in the discussion above, except for date input which is used in report headings. Note line 280, where V is still the maximum number of records the file can hold. You can increase this if your data records are short. Also note that we have not yet dimensioned any arrays because we

still don't know what size to dimension to, i.e., we could be reading in a file with seven fields or one with just three. Now let's get to line 350 and read in a file.

We named the file in line 310. If we had asked for a file that didn't exist yet the error trap would have sprung and we would be down around line 1220 where we could abort or start a new file. For the moment, let's assume the file shown earlier is on disk and we are going to read it in. In line 350 we open the file for input. The very next thing we do is to read in just that one digit from the file that tells us how many fields each record in the file has. It happens in line 360, where we input #1, NF (number of fields). The very next thing we have to do at this point is to dimension the R\$ array to accept the data we are about to read in. We set V earlier, in line 280, and now we know how many fields are in each record (it's in variable NF). So now, still in line 360, we can DIM R\$(V,NF), which in our case turns out to be R\$(400,5). At the same time and in the same line, we can dimension our pointer, P, to be the same as V, or P(400).

The next thing that happens is that we set the I loop to count from zero to V. Why V? We don't yet know how many records there are in this file, so we set the loop to the maximum number and we will jump out when we reach end of file. The next line, 380, provides our escape hatch from this loop. Inside the I loop, we set the J loop to read in the fields, from one to NF. Then we simply line input through buffer #1, R\$(I,J). Note that between the execution of the two loops, which is after the J loop is finished and before the I loop goes for the next record, we set P(I) equal to the I count (in line 420). This "attaches" the same I number from the record to the pointer, P(I). If you never understood how that pointer works, follow closely, we think we are about ready to actually explain it.

At first, as loop I increments, P(I) will have the same number as the corresponding loop count. In other words, P(23) will contain the number 23 and will point to the 23rd record that was read in.

At this point you could say, so what? But when we get to sorting you will see the magic of that P number. We'll pick up on it again when we get there.

When we reach the end of the file, control jumps from line 380 to 440, where we close the file. Then we set NR (number of records) for the first time to be equal to the I count less one. It's less one because we already advanced the I count but didn't read in a record for that I count (we aborted instead because of EOF). (Programming is such fun!)

So now what do we have? Record zero, now array element zero, R\$(0,x), looks like this:

```
R$(0,1) = Name
R$(0,2) = Address
R$(0,3) = City
R$(0,4) = St
R$(0,5) = Zip (and there's a 0 in P(0))
```

Record 1, now array element 1, looks like this:

```
R$(1,1) = John Jones
R$(1,2) = 123 Oak Street
R$(1,3) = Anytown
R$(1,4) = NY
R$(1,5) = 01234 (and there's a 1 in P(1))
```

and so on ... to EOF.

And variable NF contains the number 5, the number of fields in each record. With that, we can take each menu item and examine it. We'll take them as they appear sequentially in the code, not as they occur in the menu.

ADD Records

This takes place between lines 670 and 890. We start by clearing the screen, and that's important because we want a clean screen to start with for our cursor positioning that follows. Next we print a heading, then below that, information about what the last name in the file is, what record number it is and the total records this file

can have. It happens in line 680. When we print R\$(NR,1) we will get the Name contained in the last record number in the file. (If we had printed R\$(NR,2) we would have the Address of the last person in the file). Then we print a notice on how to quit adding records to the file by making field 1 a null string.

Next we position the cursor at X=5 and Y=1, which is 5 lines down and starting at the left margin. X and Y are always our cursor positioning variables (in all of our programs, refer back to lines 220 to 260). This does not mean that when we want to step the cursor, we cannot use X or Y in a loop to do it, as we will see shortly. Then we print, for J = 1 to the number of fields (NF); first the loop number to serve as a field number; then the field name, as in R\$(0,J); then a string of dashes that is the difference between 10 and the length of the field name. The fields in our example file above would look like this:

```
1 Name-----
2 Address---
3 City-----
4 St-----
5 Zip-----
```

Now we want to position the cursor right after some space after 1 Name----- so that we can start entering information into the record. Line 740 positions us there with an X=5 and Y=18. Now we have the screen set up for data entry.

We have a big input loop between line 750 and 890. Inside this loop there are two smaller loops; one to input the data and the other to clear space before we input the next record. Also in the loop is a clear and print routine for the top of the screen, where we keep track of which record was just entered. Last but not least, we must attach a new P(I) number to the new record just created. That occurs in line 880.

Our big loop between line 750 and 890 starts counting one past the end of our array limit (NR), and presumably can go as high as V, or the maximum number of records we set for ourselves

early in the program. Inside the big loop, the first smaller loop we come to is the input loop. Its index is X because after each entry we want to move down on the screen one line to the next entry. Since our entries start on line 5 of the screen, X has to start there, too. The limit we will step X is the number of fields plus 4. Sound wrong? Yes, it does at first, but in our example case we have five fields, and 5 plus 4 is 9 and if we start at line 5 we will have 5 fields entered when we reach the count of 9. But we have a little problem. We would like to input R\$(I,J) so that the record and fields will be accounted for. But we are stepping the loop with X, and X isn't even close to 1 to 5 like our fields should be numbered. So, in line 780, we simply line input R\$(I,X-4) to get the field number back into the range we want it in.

Then we have to check to see if the user wants to quit adding names, so we check R\$(I,1) to see if it is null, and if it is, we set the new number of records (NR) and then go to the menu. If R\$(I,1) is not null we keep entering information into the array. After we have completed one record, we need to wipe the information (only what we input) off the screen so we can enter another record. Line 810 positions the cursor at the right place to do this. Then the loop from 820 to 850, similar to the one we just left, prints 40 blanks in each position. Next we clear the heading at the top of the screen (line 860), and then print the information from the record we just entered there. We then tag P(I) to this record in line 880 and then go on to start entering the next record. Note that we don't have to update NR during this I loop. The initial NR+1 is a number, just like 56 or something like that if we already have 55 records in the file when we start adding, and it will get incremented by the loop counter itself. We only had to take one away from NR in line 790 because the loop was already incremented one past the actual number of records when we decided to quit adding records.

Quick Scan the Records

This section is almost as simple as it was in the

original Card.Bas, except that there the records flew by fast and there were no field names attached. Here, we put one record on the screen at a time, with field names attached and a prompt to quit or continue by pressing any key. The code is from lines 930 to 1050. After each record is displayed and we go on, there is no need for fancy cursor positioning, so we just clear the screen and put up the next record. When we reach the end of the file (when I=NR) we skip around the "Press Q to abort, any key to continue" message and go right to the INKEY\$ routine in line 1010 which takes us back to the main menu. Note that the records are printed in P(I) order and not the straight I order of the array. (There's more on that in the next section, too.)

Save the file and END

This routine sits between lines 1090 and 1180 and is a straightforward output routine with a couple of exceptions. First off, we print the number of fields to the file all by itself. Then we loop through the remaining records and print them to the file. As in the original Card.Bas, our delete routine simply nulls the first field of the record, so we check for that here and if that field is a null string, we skip over it and don't print it to the file, which gets rid of it by discarding it into the bit bucket. One other thing should be noted here, and that is that we are printing the records out in the order that P(I) has them, not the way they are sequenced in the array. Up until now we haven't had them any other way, but the sort will most assuredly have re-ordered the numbers in P(I), and we want the file to be in the order we sorted in. Remember that P(I) is just a number, like 10 or 35 or any other number. The I numbers for the array will always start at 0 and go sequentially to the end of the file; the P(I) numbers will not necessarily be in that order, as we will see when we get to the sort routine.

Error Trap and Initialization

When you start the program you will be asked for a file name. If you give one that doesn't exist,

you will be given the opportunity to abort or to start a new file. This code is between lines 1220 and 1380. Line 1230 is for those of you who have BASIC prior to version 5.0. If you do, then move the remark from line 1230 to 1220.

If you have followed Beginning BASIC in the past few issues, you know all about INSTR. It's used in line 1270 to give the position of CC\$ in the string, "yYnN." If the position is zero, it means that you didn't press either upper or lower case Y or N, and we go back and wait for the proper input. If the position is 3 or 4, it means "no," so we end the program right there. Otherwise, we fall right through the code and create a new file.

Since at this point we don't know yet how many fields there will be, we set the loop count in line 1330 to 10. We are sort of limited to about 10 by both screen size and in the report generating section to come later. In lines 1330 to 1360 we input the name for each of the fields we want to have. Pressing enter all by itself will terminate entries. When we enter the null field by pressing enter, we set the number of fields (NF) to J less 1 and go to the main menu, where most likely, you will want to start adding records.

Search (Edit & Delete)

This is one of the longer sections in the program. It starts at line 1420 and ends at 1860. First we clear the screen, then we get record zero with a J loop and print the field names on the screen. Then, in lines 1470 to 1500, we establish which field number to search on and what the search string will be. The search string is contained in S\$, and the field number to search in is in variable SF (for Search Field). Having gotten this out of the way, we clear the screen again and get ready to do some more direct cursor manipulation.

First though, we set A equal to 1, and then search the R\$(x,x) array for our search string. We search the array from A to the number of records, and since A at this point is 1 we search

the whole array (except for record zero, which we don't want to include). The reason for variable A will become apparent shortly. In line 1540 we do an INSTR search to find what we are looking for. If we don't find it in the whole file, INSTR will be equal to zero and we drop through the next line and print that no match was found, and to press any key, which takes us back to the main menu.

If we do find a match, control jumps to line 1590 where we first clear the spaces we are going to use on the screen (in lines 1600 to 1630). Then we reposition the cursor in line 1640 and print the record in which we found the match. We print the whole record, including the field headings, and again, we are using P(I) as a pointer to the record. Then we drop down a couple of lines and print the prompt for "Next match, Previous, Delete, Edit or Quit."

The response to this prompt is an INKEY\$ response so that you get action as soon as you press the key and you don't need to press enter. There are only five responses allowed here, and line 1710 uses INSTR again to make sure that you pressed a valid upper or lower case response. If you didn't, control goes right back to the INKEY\$ line at 1700 and waits for the proper key to be pressed.

Let's take the responses in the order in which they appear in the program. If you press N, for Next match, we jump to line 1820. Here we position the cursor and wipe off the record currently on the screen. Then we readjust A to be equal to I plus 1. Do you see the reason for variable A now? If we didn't advance A by one, we would be stuck on the same record and couldn't get off of it. The A=I+1 advanced us to the *next* record after the one where we found the match, so we can keep on looking through the array for more matches. And that's what happens in line 1860, we go back to 1530 with our new A and keep right on looking.

When you press P for previous, we subtract one from A and send control back to line 1530. First we have to make sure that A is not going to

be less than one because we can't look at records prior to record 1. You get an error if you try to do that. It turns out that if you are searching on the sorted field in a file you can step back with the previous option all the way to the first match. If you are searching on an unsorted field, however, you can only back up one previous match. If you try to go farther, it just recycles on that record. It will also recycle on record 1 if you get back that far.

If we pressed D for Delete, line 1730 will change the first field of the record to a null string. Then control goes to 1820, just like it did for Next match. This is so you can do a whole string of deletes in a row, without having to start at the beginning. (You're not locked into this, we'll get to the Quit option shortly.)

If you press E for Edit, we have a bunch of fancy stuff to do. First, in line 1750, we ask for the number of the line to edit. This is another "hot" key which uses INKEY\$. But since INKEY\$ demands a string, we need to take the VAL (value) of the number so it will work in line 1780, where we input the correct line and put it into R\$(I,WF). I, of course, is the record we are currently looking at and editing. WF (which field) is the field number we are editing. The dialog which just took place on the screen happened on screen lines that correspond to NF plus 6 and NF plus 7. We want to get rid of that dialog without disturbing the remainder of the screen, and we do it just like that in lines 1790 and 1800. Then we go back to line 1600 (from line 1810) where we clear out the record and re-write it, showing the editing change we just made. After this, we are presented again with the same "Next match, Previous, Delete, Edit or Quit" prompt, so we can make more edits if we like or do whatever we want.

If you press Q for Quit, control simply sends us back to the main menu.

Did you notice throughout the entire program that we never refer to the number of fields directly, but only by referring to NF? That's

because we are never sure when we start the program which data file we are going to be using, and the number of fields can vary from data file to data file.

The Shell Sort Routine

Frankly, I am not very happy about my ability to describe the Shell sort to you. Perhaps I don't know it well enough myself. I do know that if you see the way it works you will understand it, so I'll take some time out here and see if I can write a little demo sidebar to this article that will do that.

(Two days later ...) Well, I'm back. It was an interesting side trip and I still haven't written the side bar. But I know what has to go into it and how we'll do it, and that's half the problem solved. During the process of getting to know the Shell sort, I found that the original Card.Bas had a flaw in the sort routine. Don't ask how it got there, or why - I don't know. At any rate, it's fixed now, we removed about three lines of code in the sort and increased its speed a whopping 40 percent! It's not that the original Card didn't sort right - it did, it just wasn't as fast as it could have been.

The sort routine (see the sidebar, also) is what is called a "modified Shell sort." On a 10 Mhz CPU it sorted 217 names and addresses in 15 seconds flat, and that was even with time out to print a period on the screen for each swap that it did. For comparison, on the same machine, with a bubble sort, it took roughly 35 minutes. In both cases the list of names was sorted in reverse order first, and then timed to see how long it would take to put back into ascending order by name.

We can't possibly leave this section without getting into our old friend, P(I) - we promised, remember? Let's start by saying that the file you read into the memory array from disk is in memory in the same order that it was on disk and as long as it is in memory *it never changes*. It stays that way, even when you sort. Record three

from disk is still the third record in the array and it stays that way the whole time it is in memory. When we read in the array from disk, we assigned P(I) to be equal to I (the loop counter). So, P(I) and I will be identical for any record. If the loop count (I) is 15, so will P(I) be 15.

When we sort, however, we just *look* at the information we want to sort on, and if the item (I) we are looking at is greater than item the next item (I+1), then we *switch the P(I) numbers only*. Look at lines 2050 and 2060, and you can see where this happens. We don't have to switch each field either, because when we are looking at a record, the I count tells us which record it is and that same I count applies to each of the fields in that record. When we swap P(I) numbers, we are effectively swapping for each of the fields as well.

P(I) is an array that was "synchronized" to the I count of the array. Each number in P(I) pointed to a number identical with the I count. After we sort, the P(I) array numbers will no longer be synchronized to the I count numbers, instead they will point to the *sorted order* of the R array. Nothing in the array changed places - each record is still where it always was. But now, when we search, or scan or print, we do so in the order of P(I), and we see the list in the sorted order that we wanted. Sneaky, huh? That's why the sort is so fast. It's because P(I) is an integer number, and that's all we really changed when we sorted. Strings just never get moved with this method.

When we are finally done, and end the session and save the file, then the records are sent to the disk in the order of P(I), and they actually change places then, compared to what they were when we read them from disk.

If you still don't get it, look at it this way, with just five records. Loop counter I reads them from disk and attaches P(I) to each. I reads from 1 to 5 and P(I) also reads from 1 to 5. The names, let's say, are:

| Before the sort: | | After the sort: | |
|------------------|---------|-----------------|---------|
| (I) | P(I) | (I) | P(I) |
| 1 | Foxtrot | 1 | Foxtrot |
| 2 | Delta | 2 | Delta |
| 3 | Charlie | 3 | Charlie |
| 4 | Baker | 4 | Baker |
| 5 | Able | 5 | Able |

Now instead of reading R\$(I), we read R\$(P(I)), and we see the list in sorted order instead of the other way around. And the names never changed their places in the array at all. Pointers, you'll have to admit, are clever devices.

The Print Routines - Report Format

We had a problem with this. How do you set up a generalized report or label format when you don't even know how many fields there will be or what's in which field? Well, for years I have been waiting for a good reason to explore the idea of tabs that are programmed, in other words, tabs that change depending on the data. Here, at last, was an opportunity to do it. Not only do the tabs set themselves, but they adjust to the length of the largest field for each tab. In addition, they take into account situations where the field name is longer than the data in that field, as when the field name might be "Overdue" and the file contains a "yes" or "no" for that field. What's more, the program totals the tabs and adds the last field length and if it's more than 80 characters it automatically shifts your printer into 132 character mode (and back to 80 characters when it's done). If the tabs total more than 132 we just have to live with wrap-around. Let's take the output selection section first.

At line 2210 we clear the screen and print the printed output selection heading. We're using hot keys here again, and line 2320 checks for legal input with our old friend, INSTR. You can select L for standard sticky labels, R for report format or Q if you change your mind about being at this section in the first place.

The report section starts at line 2340 and ends at line 2800. The first thing we do (at line 2370)

is to find the maximum length of each of the fields we are dealing with. The loop at 2370 goes through each of the fields in the file and lets A(J) equal the highest length it finds. At the same time, we have to cruise through the field names themselves, in record zero, to see if any of them are longer than the actual data in the field. It would seem that all this would take a long time, but on our 10 Mhz machine, with our 217 name test file, it only took a few seconds. So when we are done, A(J) (for however many fields J is) holds the longest length of each field.

You can't just take the information in A(J) and set tabs. In fact, the first print position isn't a tab at all, it starts at the left margin. Then, each successive tab has to be added to all previous tabs to get the actual tab position. It wasn't as easy as it first seemed. In addition, we have to provide for at least ten columns of print, even though there may not be that many.

In lines 2460 to 2480 we simply go through the A(J) array and add one to provide for at least one space between the tabbed columns. In lines 2510 to 2530 we get the total of all the tabs and put that value into TB.

Lines 2550 to 2580 is where you will probably have to make some changes depending on your printer. The values in the program now are for a Micronics Star SG-10 in IBM mode. It is in these lines where we set the printer to 132 column mode and back. Also, if the printer is set to 132 columns, MS DOS (and some CP/M) people will have to set the width of lprint. That occurs in line 2580. Remark line 2580 if you don't have to set width.

Line 2610 is where we finally calculate the actual tabs and put them into array T(x). Note that each one, after the first two, are equal to the previous T value plus the A(J) value. On my machine tab(1) and tab(0) are the same, so T(1) is equal to 1, T(2) is equal to the longest line in the first field plus one, T(3) equals T(2) plus the longest length in the second field plus one, and so on. It turns out that variable NF (number of

fields) will tell us how many tabs to use, and we don't even have to worry about it.

From line 2640 to 2800 we print the report. In line 2640, we set PG (page) to one and LC (line count) to zero. Then we go through all the records, from one to NR (number of records), and from field one to the number of fields (NF), we LPRINT at TAB(T(J)), the information in that particular field. Before that happens, however, at line 2660, we check to see if this is the first line of the report or the 61st line, etc., and if it is, we go to the subroutine at 2770 to print the page header first. We then print 60 lines of information, incrementing LC (line count) each time we print a line. When LC reaches 60, we set it back to zero, increment the page count and issue a page eject to the printer so that it will go to the top of the next page. This all happens in line 2700. In line 2720 we are at a point where all of the report has been printed, but the last page might not be full, so we issue another page eject with the CHR\$(12). Then in the next line (2730) we set the printer back to normal mode, regardless of what it was at. Who knows? You might want to print labels next and don't want them to be all in condensed print. It's just a housekeeping thing - that's all. Watch it run first, then come back and read this again, and yes, this and the next section of code were a real kick to write. It's so nice when things work!

The Print Routines - Label Format

We had the same problems with label format as with report format. How do you know what goes where and how many fields will the user want to put on one line (like City, State and Zip)? What if he has First name in one field and Last name in another? How are we going to let him hook them together the right way? We're happy to say that the solution to all that was even simpler than the report formatting. It starts at line 2840 and runs to the end of the program. Here, we print the field headings on the screen for you to see which is which (in lines 2870 to 2890). Then we put some instructions on the screen: Press ENTER for any blank lines, and to

put more than one field on one line, just put the field numbers, one right after the other, like 12 for fields 1 and 2 on the same line, or you could even put 21 if you wanted them in that order.

Starting at line 2950, we go through a loop six times and ask what goes on each line. Why six? Well, we normally print six lines per inch, and the labels are exactly one inch from top of one to top of next. By forcing you to enter six lines (even if they are blank) we have automatically set the correct spacing between labels. Not only that, but it allows you to have a two, three, four or even five line label if you like. (You can't print six lines, it would print in the crack between the labels!) In the loop at line 2970, we check to see if the value of the field number you just input is more than nine. If it is, you obviously want more

than one field on one line, and we go to the subroutine at line 3110 to "take apart" your number. Can you see why we used strings here instead of integers? Strings are so much easier to take apart, and you can always use VAL to get them back to an integer later.

When we finally get to line 3020 to print the labels, we check for a value of zero (which means you pressed ENTER without giving a field number) and if so, we print a blank line. Otherwise, if you designated a field or fields, we print them in the order that you mentioned them - up to four of them per line. But what happens when you don't want four fields on one line? Nothing happens. If I\$, J\$, K\$ or L\$ is a null string, nothing prints, and we are home free. Try it, you'll see.

Figure 1 gives a graphic look at how the Shell sort works. A little study of this figure goes a long way in understanding what's going on. The symmetry generated in this figure is due to the fact that we started with a reverse ordered list. If the list were not in reverse order such symmetry would not be apparent.

As you can see, the list is first divided into two (line 220 of the program is what does that). A comparison is then made of the first item in each half, and if the first item in the upper half is smaller than the first item in the bottom half, a switch is made. Note that this is a wholesale move that takes the item at least half way to where it should be. In our case the 7 is only one away from where it should be, but is moved way to the left to the lowest position. It is put into its proper place in one later move, when it is switched with the 1. Some numbers happen to end up exactly where they belong after just one move - the 10, for example.

After the first cut in half is over, the remaining sections are again cut in half and the process is repeated on each of the remaining parts. Variable M in line 220 gets cut in half each time a cut

is made until it is finally zero, indicating that we are done (see line 230).

Just for comparison, a Bubble Sort would have taken 78 moves to do the same thing. For those interested, the Bubble Sort formula is the number of items squared less the number of items, all divided by 2. It's difficult to come up with a formula for the Shell Sort, suffice it to say it's faster by far, especially when a larger number of items are to be sorted. It's that square term in the Bubble Sort formula that gets you.

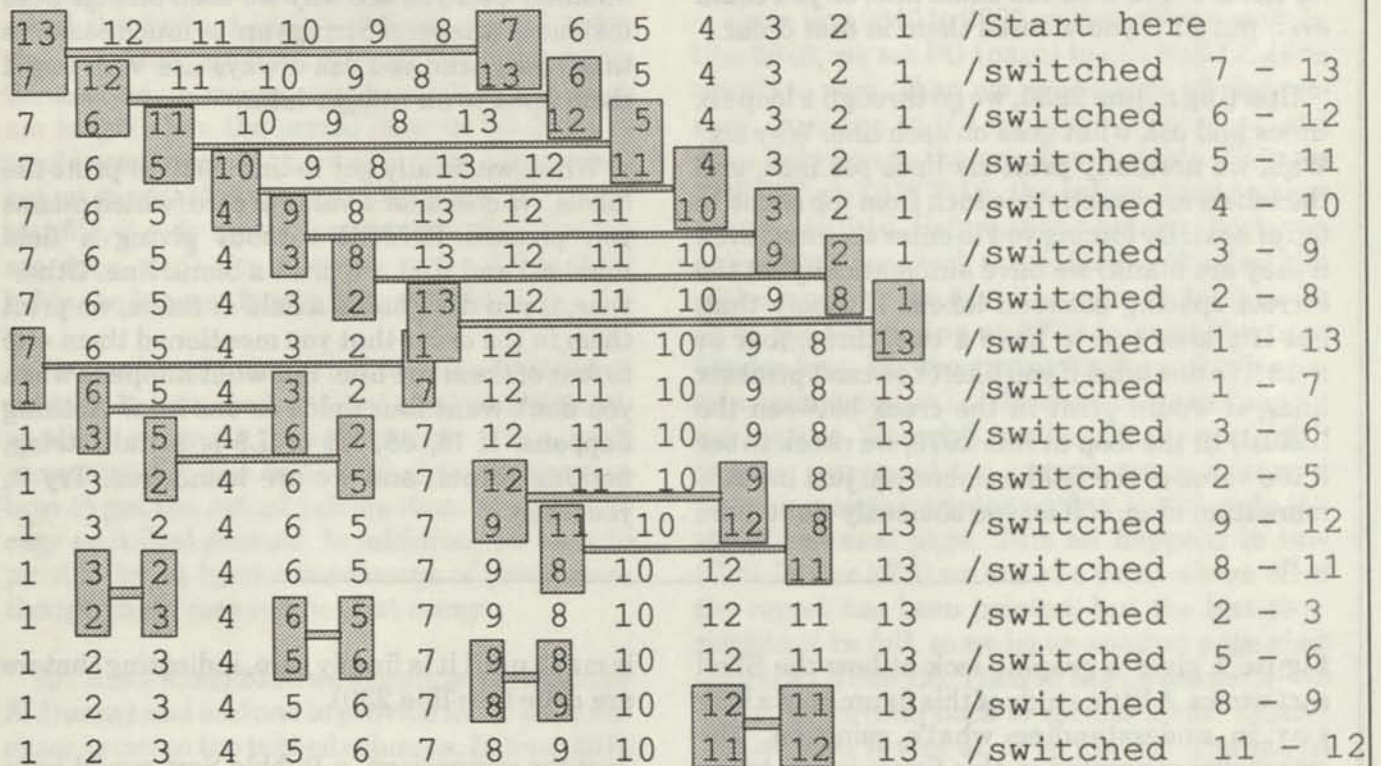
If you want to play around with other than reverse ordered numbers, you can remark line 140 and un-remark both line 150 and 160. This will get you a set of random numbers. When you do this, give the program the same number of items to sort several times and watch the number of swaps required. It all depends on how unsorted the original list is.

Naturally, lines 180 and 300 are in the program only to print out the results after each swap. You wouldn't need them if you incorporate this routine into some other program. Nor would you need the beginning lines, up to line 200. And,

incidentally, we used 13 items because that was just about the most we could use that would fit across a page without wrapping.

the Shell Sort, that being the Quick Sort. It's just a wee bit faster, we're told, but we haven't been able to find the routine in any of our books. We'll keep looking though, and when we do we'll explore it, too.

There is only one sort routine that is faster than



Sorted after 16 swaps.

Figure 1

```

100 REM * Shell.bas * a shell sort demo
110 DIM P(101)
120 INPUT "How many items in the array ";NR
130 FOR I=1 TO NR
140   P(I)=(NR+1)-I
150   'P(I)=INT(RND(0)*98)+1
160   'RANDOMIZE TIMER
170 NEXT I
180 FOR Q=1 TO NR:PRINT P(Q);:NEXT Q:PRINT " /Start here"
190 'The actual sort begins here
200 N=NR
210 M=N
220 M=INT(M/2)
230 IF M=0 THEN 370
240 J=1
250 K=N-M
260 I=J

```

```

270 L=I+M
280 IF P(I)=<P(L) THEN 340
290 SWAP P(I),P(L):COUNT=COUNT+1
300 FOR Q=1 TO NR:PRINT P(Q);:NEXT Q:PRINT `` /switched ``P(I);''-'';P(L)
310 I=I-M
320 IF I<1 THEN 340
330 GOTO 270
340 J=J+1
350 IF J>K THEN 220
360 GOTO 260
370 PRINT''Sorted after ``;COUNT;'' swaps.''
380 END

```

Card2.Bas program listing

```

100 REM * Card2.Bas * an improved Card.Bas program 19 Jun 89 ims
110 REM * CodeWorks Magazine 3838 S. Warner St. Tacoma WA 98409
120 REM * (206) 475-2219 * Please leave these credit lines.
130 REM * (c)1989 80-NW Publishing Inc. & placed in public domain.
140 ON ERROR GOTO 1220
150 `NR = number of records in the file
160 `NF = number of fields in a record
170 `R$(x,x) = data records
180 `P(I) = integer pointer to data records
190 `
200 `Generalized Locate/Print@ subroutine. Unremark as needed.
210 GOTO 270
220 LOCATE X,Y:RETURN `MS-DOS, GW-BASIC
230 `PRINT@((X-1)*64)+(Y-1),,:RETURN `Tandy Models I/III
240 `PRINT@((X-1),(Y-1)),,:RETURN `Tandy Models II/IV
250 `PRINT@(X,Y),,:RETURN `Some MBASIC machines.
260 `PRINT CHR$(27)+'Y'+CHR$(31+X)+CHR$(31+Y):RETURN `CP/M
270 ` Initialization
280 V=400 ` set max number of records file can hold here
290 CLS
300 IF DATE$<>' ' THEN ELSE INPUT''What is today's date'';DATE$
310 INPUT''What file name do you wish to use ``;F1$
320 `
330 REM * Open the file and read it
340 `
350 OPEN ``I'',1,F1$
360 INPUT #1,NF:DIM R$(V,NF),P(V)
370 FOR I=0 TO V

```

```

380 IF EOF(1) THEN 440
390 FOR J=1 TO NF
400 LINE INPUT #1, R$(I,J)
410 NEXT J
420 P(I)=I
430 NEXT I
440 CLOSE 1
450 NR=I-1
460 `
470 CLS
480 PRINT STRING$(22,45);'' The CodeWorks '';STRING$(23,45)
490 PRINT'' CARD FILE PROGRAM Version 2
500 PRINT'' an in-memory replacement for 3 x 5 cards
510 PRINT STRING$(60,45)
520 PRINT'' You are currently working with '';F1$
530 PRINT
540 PRINT TAB(15);''1 - ADD records
550 PRINT TAB(15);''2 - SEARCH (Edit & Delete)
560 PRINT TAB(15);''3 - Quick SCAN all records
570 PRINT TAB(15);''4 - SORT
580 PRINT TAB(15);''5 - PRINT
590 PRINT TAB(15);''6 - SAVE file and END
600 PRINT
610 INPUT''The number of your choice '';XX
620 ON XX GOTO 670,1420,930,1900,2210,1090
630 GOTO 610
640 `
650 REM * ADD records routine
660 `
670 CLS
680 PRINT TAB(20) ``ADD Records'':PRINT''Last record was: '';R$(NR,1);''
Number '';NR;'' of '';V
690 PRINT''To QUIT adding records, press enter on 1st field.''
700 X=5:Y=1:GOSUB 220
710 FOR J=1 TO NF
720 PRINT J;R$(0,J);STRING$(10-LEN(R$(0,J)),45)
730 NEXT J
740 X=5:Y=18:GOSUB 220
750 FOR I=NR+1 TO V
760 FOR X=5 TO 4+NF
770 Y=18:GOSUB 220
780 LINE INPUT R$(I,X-4)
790 IF R$(I,1)='' THEN NR=I-1:GOTO 470
800 NEXT X
810 X=5:Y=18:GOSUB 220
820 FOR X=5 TO 4+NF
830 Y=18:GOSUB 220
840 PRINT STRING$(40,32)

```

```

850 NEXT X
860 X=2:Y=1:GOSUB 220:PRINT STRING$(60,32)
870 X=2:Y=1:GOSUB 220:PRINT"Last record was: ";R$(I,1);" Number ";
      I;" of ";V
880 P(I)=I
890 NEXT I
900 `
910 REM * Quick scan all the records
920 `
930 CLS
940 FOR I=1 TO NR
950   FOR J=1 TO NF
960     PRINT J;R$(0,J);STRING$(10-LEN(R$(0,J)),45);" ";R$(P(I),J)
970   NEXT J
980   PRINT
990   IF I=NR THEN PRINT"***** END OF FILE ***** press any key":GOTO
      1010
1000 PRINT"Press Q to abort, any other key to continue"
1010 K$=INKEY$:IF K$="" THEN 1010
1020 IF K$="Q" OR K$="q" THEN 470
1030 CLS
1040 NEXT I
1050 GOTO 470
1060 `
1070 REM * save the file and end routine
1080 `
1090 OPEN"O",1,F1$
1100 PRINT #1,NF
1110 FOR I=0 TO NR
1120   FOR J=1 TO NF
1130     IF R$(P(I),1)="" THEN 1160
1140     PRINT #1,R$(P(I),J)
1150   NEXT J
1160 NEXT I
1170 CLOSE 1
1180 END
1190 `
1200 REM * error trap for file not found & file initialization
1210 `
1220 IF ERR <> 53 THEN ON ERROR GOTO 0
1230 `IF (ERR/2)+1 <> 54 THEN ON ERROR GOTO 0
1240 PRINT"There is no file called ";F1$
1250 PRINT"Do you wish to create one (y/n) "
1260 CC$=INKEY$:IF CC$="" THEN 1260
1270 AN=INSTR("yYnN",CC$)
1280 IF AN=0 THEN 1260 ELSE IF AN=3 OR AN=4 THEN END
1290 PRINT"Prepare to create a new data file."
1300 PRINT

```

```

1310 PRINT "Simply press ENTER to quit adding fields"
1320 PRINT "Field names can be no more than 10 characters long."
1330 FOR J=1 TO 10
1340 PRINT "Name of field ";J;:INPUT R$(0,J)
1350 IF R$(0,J)="" THEN 1370
1360 NEXT J
1370 NF=J-1
1380 RESUME 470
1390 `
1400 REM * Search (Edit & Delete) routine
1410 `
1420 CLS
1430 FOR J=1 TO NF
1440 PRINT J;R$(0,J)
1450 NEXT J
1460 PRINT
1470 INPUT "What field number do you want to search on ";SF
1480 IF SF<1 OR SF>NF THEN 1470
1490 PRINT "Enter the item (or part of the item) you wish to find."
1500 INPUT S$
1510 CLS
1520 A=1
1530 FOR I=A TO NR
1540 IF INSTR(R$(P(I),SF),S$) <> 0 THEN 1590
1550 NEXT I
1560 CLS:PRINT "No MATCH was found - Press any key"
1570 K$=INKEY$:IF K$="" THEN 1570
1580 GOTO 470
1590 PRINT
1600 X=4:Y=1:GOSUB 220
1610 FOR J=1 TO NF
1620 PRINT STRING$(60,32)
1630 NEXT J
1640 X=4:Y=1:GOSUB 220
1650 FOR J=1 TO NF
1660 PRINT J;R$(0,J);STRING$(10-LEN(R$(0,J)),45);" ";R$(P(I),J)
1670 NEXT J
1680 PRINT
1690 PRINT "(N)ext match (P)revious (D)elete (E)dit (Q)uit"
1700 K$=INKEY$:IF K$="" THEN 1700
1710 IF INSTR("NnPpDdQqEe",K$)=0 THEN 1700
1720 IF K$="N" OR K$="n" THEN 1830
1730 IF K$="P" OR K$="p" THEN A=A-1:IF A<=1 THEN A=1:GOTO 1530 ELSE
1530
1740 IF K$="D" OR K$="d" THEN R$(P(I),1)="":GOTO 1830
1750 IF K$="Q" OR K$="q" THEN 470
1760 IF K$="E" OR K$="e" THEN PRINT "Edit which field number "
1770 WF$=INKEY$:IF WF$="" THEN 1770

```



```

1780 WF=VAL(WF$)
1790 INPUT''Enter correction ``;R$(P(I),WF)
1800 X=NF+6:Y=1:GOSUB 220:PRINT STRING$(60,32)
1810 X=NF+7:Y=1:GOSUB 220:PRINT STRING$(60,32)
1820 GOTO 1600
1830 FOR X=4 TO NF+4
1840   Y=1:GOSUB 220:PRINT STRING$(60,32)
1850 NEXT X
1860 A=I+1:GOTO 1530
1870 `
1880 REM * sort the data routine (Shell sort)
1890 `
1900 CLS
1910 FOR J=1 TO NF
1920   PRINT J;R$(0,J)
1930 NEXT J
1940 INPUT''What field number to sort on ``;Q
1950 IF Q<1 OR Q>NF THEN 1940
1960 PRINT ``Sorting - each dot represents a swap''
1970 N=NR
1980 M=N
1990 M=INT(M/2)
2000 IF M=0 THEN 2150
2010 J=1
2020 K=N-M
2030 I=J
2040 L=I+M
2050 IF R$(P(I),Q)=<R$(P(L),Q) THEN 2120
2060 SWAP P(I),P(L)
2070 `T=P(I):P(I)=P(L):P(L)=T `for you who can't swap
2080 PRINT''.``; `dots are to watch on the screen
2090 I=I-M
2100 IF I<1 THEN 2120
2110 GOTO 2040
2120 J=J+1
2130 IF J>K THEN 1990
2140 GOTO 2030
2150 PRINT:PRINT''Sorted - press any key''
2160 K$=INKEY$:IF K$=`` THEN 2160
2170 GOTO 470
2180 `
2190 REM * the PRINT routines
2200 `
2210 CLS
2220 PRINT TAB(15);''Printed Output Selection''
2230 PRINT
2240 PRINT TAB(10);''Press L for Label format
2250 PRINT TAB(10);''Press R for Report format

```

```

2260 PRINT TAB(10);''Press Q if you got here by mistake
2270 PRINT TAB(5);''(Turn on your printer and align to top of form)''
2280 K$=INKEY$:IF K$=''' THEN 2280
2290 IF K$='R' OR K$='r' THEN 2350
2300 IF K$='L' OR K$='l' THEN 2840
2310 IF K$='Q' OR K$='q' THEN 470
2320 IF INSTR('RrLlQq',K$)=0 THEN 2280
2330 '
2340 REM * the REPORT routine
2350 '
2360 ' first find maximum field or heading lengths
2370 FOR I=1 TO NR
2380   FOR J=1 TO NF
2390     IF LEN(R$(I,J))>A(J) THEN A(J)=LEN(R$(I,J))
2400     IF LEN(R$(0,J))>A(J) THEN A(J)=LEN(R$(0,J))
2410   NEXT J
2420 NEXT I
2430 A(1)=A(1)+1
2440 '
2450 ' add one for space between columns
2460 FOR J=1 TO NF
2470   A(J)=A(J)+1
2480 NEXT J
2490 '
2500 ' find total of all tabs
2510 FOR J=1 TO NF
2520   TB=TB+A(J)
2530 NEXT J
2540 '
2550 ' set the printer to condensed mode if more than 80 cols - reset
    if not
2560 IF TB>81 THEN LPRINT CHR$(15) ELSE LPRINT CHR$(18)
2570 ' set printer width too if you need to
2580 IF TB>81 THEN WIDTH LPRINT 132 ELSE WIDTH LPRINT 80
2590 '
2600 ' format the tabs in T( )
2610 T(1)=1:T(2)=A(1):T(3)=T(2)+A(2):T(4)=T(3)+A(3):T(5)=T(4)+A(4):
    T(6)=T(5)+A(5):T(7)=T(6)+A(6):T(8)=T(7)+A(7):
    T(9)=T(8)+A(8):T(10)=T(9)+A(9)
2620 '
2630 ' print out the report
2640 PG=1:LC=0
2650 FOR I=1 TO NR
2660   IF I MOD 60=1 THEN GOSUB 2770
2670   FOR J=1 TO NF
2680     LPRINT TAB(T(J));R$(P(I),J);
2690   NEXT J
2700   LC=LC+1:IF LC=60 THEN LC=0:PG=PG+1:LPRINT CHR$(12)

```

Write.Bas

```
2710 NEXT I
2720 LPRINT CHR$(12) ` to page eject partial or last page
2730 LPRINT CHR$(18) ` set the printer back to normal mode
2740 GOTO 470
2750 `
2760 ` subroutine to print page header
2770 LPRINT F1$;' ' `;DATE$;' ' `;' 'Page ` `;PG
2780 FOR K=1 TO NF:LPRINT TAB(T(K));R$(0,K);:NEXT K
2790 LPRINT STRING$(TB-2,45)
2800 RETURN
2810 `
2820 ` REM * print in label format routine
2830 `
2840 CLS
2850 PRINT TAB(20);' 'Label format' '
2860 PRINT TAB(10);' 'Align labels to first print position' '
2870 FOR J=1 TO NF
2880   PRINT J;' ' `;R$(0,J)
2890 NEXT J
2900 PRINT
2910 PRINT' 'Press ENTER for any blank lines on the label.' '
2920 PRINT' '(To enter successive fields on one line simply enter
2930 PRINT' 'the field numbers, for example: 34 for fields 3 and 4)
2940 PRINT
2950 FOR I=1 TO 6
2960   PRINT' 'What field number(s) on line' `;I;:INPUT L$(I)
2970   IF VAL(L$(I))>9 THEN GOSUB 3110
2980 NEXT I
2990 `
3000 ` print the labels
3010 `
3020 FOR I=1 TO NR
3030   FOR J=1 TO 6
3040     IF VAL(L$(J))=0 THEN LPRINT' ' ` ELSE LPRINT R$(P(I),
      VAL(L$(J)));' ' `;R$(P(I),VAL(K$(J)));' ' `;R$(P(I),
      VAL(J$(J)));' ' `;R$(P(I),VAL(I$(J)))
3050   NEXT J
3060 NEXT I
3070 GOTO 470
3080 `
3090 ` multiple fields per line `take-apart` subroutine
3100 `
3110 I$(I)=MID$(L$(I),4,1)
3120 J$(I)=MID$(L$(I),3,1)
3130 K$(I)=MID$(L$(I),2,1)
3140 L$(I)=MID$(L$(I),1,1)
3150 RETURN
3160 END `of program
```

Ckrite.Bas

Convert Numerals into Words

Irv Schmidt, Editor. A fun program to design and write. Here is another example of a reader-requested program. It's presented as a stand-alone, but can easily be converted to a subroutine - as well it should be.

Ckrite.Bas is presented here as a stand-alone program that shows how it works, but as we will explain later, it will have much more utility as a subroutine, perhaps in a payroll program. Basically, the program takes an integer dollar amount and changes it into the words you would use when writing a check. It will handle amounts from one dollar up to \$99,999.99, which should be enough for most of us. We don't suppose that people who write checks routinely for over \$100,000 would be reading this article in any case.

This was another of those reader-suggested programs that turned out to be both a challenge and fun. Yes, we have seen such routines before, but couldn't find one and decided to write it from scratch. Looking at it now, we wish it had come up during the time we were talking about strings in Beginning BASIC. It could have made a perfect example of string manipulation techniques.

It turns out that just 29 words will suffice to make up any dollar amount up to, but not including, a million. These, of course, must be put into data statements. In our case, we killed two birds with one stone and put the space following each word right there in the data statements as well. That way, we need not worry about that again. For this reason, all the data elements are enclosed in quotes, with the space following the word. Actually, we only used 27 words in the data statements, it being easier to put the

Hundred and Thousand in two or three places where they were needed.

Because this may be used as a subroutine, and to prevent any clashes with your variable names, we chose to use only variables beginning with Z. The Z loop in lines 200 to 220 read the data into the Z\$(Z) array. Once they are in there we can effectively forget them. They only need to be read once.

The integer amount we will work with is input in line 240 as ZA. Line 250 checks for the limits that ZA can have and ends if we exceed those limits. As a subroutine, you would want to change line 250 to return to the calling routine and print an error message, or at least an "out of limits" message.

Now we run into all sorts of fun. Mostly because of the way GW BASIC deals with decimals. You get some awful screwy numbers when you take a decimal number and subtract the integer from it. If you want to see it, print $9.98 - \text{int}(9.98)$ and you will get .9799996. In a normal program that prints number amounts, you can always use PRINT USING to make things come out right, but we couldn't find an easy way to use PRINT USING here, so we had to program around it. In line 260 we make a double precision number out of ZA to help us out of this mess.

It would seem that all we would have to do to

pull the cents off of a number would be to make a string out of all of it and then look at the right two places. But it doesn't work that way either - for \$9.98 you again get that .9799996 decimal amount.

By convention, we always want to print cents on a check as "O2" or "20" or, if there are no cents, "NO" cents. Problem is though, that the computer might send ZA in as a number with no decimal point, with just a decimal point or with no trailing zero when there is a decimal amount (like 102.40 would show as 102.4).

To make the rounding work right, in line 270, we add .005 to the decimal part of the number only when there is a decimal part. If the integer of ZA is equal to ZA there is no decimal part, otherwise there is and we add the .005 to it. In line 280 we make a string out of ZA. If there was no decimal part to the number, in line 290, we add the point and the two zeros. Keep in mind that the number ZA could have had no decimal part, could have had one decimal place or two. Next, in line 300, we find the position of the decimal point in string ZA\$. By now we are sure it has one, since if it didn't, we added one in line 290.

In line 310 we pull off ZC\$. We do it with a MID\$ that only looks at two places after the decimal, and because we added the .005 earlier, it always comes out right in spite of GW BASICs efforts to mess it up. In line 320 we go through ZC\$ (the cents) and change any zero to a capital letter Oh. We didn't really have to do this, but they just seem to look better on a check.

We still might have just one digit in the cents, so line 330 checks for that and if ZC\$ is only one digit long, it adds an Oh to the string. This makes the cents value of .2 come out as "20" like we want it to. Having done all this, if the ZC\$ is equal to "OO" we change it to "NO" and then in line 350 we finally change ZC\$ to an "&" a space and the cents and a "/100".

A lot of trouble to go through for just a few

cents, wouldn't you say?

Now all we need to worry about is the whole dollar amount. In line 370 we make a string out of the integer of ZA#. But STR\$ puts that leading space in front of the number, so in line 380 we get rid of the space. (In QuickBASIC we could have used LTRIM\$(ZB\$) here and done the same thing. LTRIM\$ and RTRIM\$ trim left and right spaces from a string, respectively.)

Next, we find the length of the dollar amount and put it into variable ZL. Then we use a little loop to put each digit in the dollar amount into the array Z(Z). Because the Z loop counter is one past the end when we are done, we back it up one in line 430. It gets tricky here, so let's use an example.

Let's say the number in question is 61203. The Z(Z) array would look like this:

1. Z(1) = 6
2. Z(2) = 1
3. Z(3) = 2
4. Z(4) = 0
5. Z(5) = 3

The Z loop count, because we backed it up one in line 430, is now 5, so Z = 5. Let's go back up to lines 200 to 220 for a minute and look at something first. The Z\$(Z) array contains the words we want. Z\$(2) contains TWO, right? And Z\$(2+10) would contain TWELVE, and Z\$(2+18) contains TWENTY. As you can see, we can always add 10 to get the teens, and 18 to get twenty, thirty, forty, etc. We are going to start with the least significant digit first and work back towards the most significant. Let's go now, to line 440.

When we look at the least significant digit we see it is a three. But how do we know that it's not part of 13? It makes a big difference what the preceding number is. It would also make a difference if the preceding number was larger than one, because in that case it might be 23 or 33, and if it was a one, it would have to be 13. It could

also be a zero, as it is in our example, in which case we can simply go and get the word "THREE" and use it as is. So, in line 440 we first check the next-to-most significant digit to see if it is zero, one or larger than one. We do that by looking at Z(Z-1). Remember that Z and Z(Z) are two entirely different variables. Right now, Z is 5, Z-1 is 4, and Z(Z-1) contains the digit zero. That being the case, we start building our ZZ\$ (which will be our final output string in the end). We say, in line 440 that ZZ\$ is equal to Z\$(Z(Z)). Z(Z) now contains a 3, and so Z\$(Z(Z)) will get us the word, "THREE" from the Z\$(Z) array.

In line 450, had the next-to-most significant digit been a one, we would have got Z\$(Z(Z)+10) from the Z\$(Z) array, and it would have been a "THIRTEEN." Next, if the length of ZB\$ (our integer dollar amount) was only one character long, we would be done and line 460 would send us to the output, where we would attach the cents portion and be done. Line 470 takes care of the case where the next-to-most significant digit is greater than one. Here, we have to do just a little more than before. If that digit is greater than one we want to say "TWENTY" (or THIRTY or whatever) plus the least significant digit. So in line 470, we look at Z\$(Z(Z-1)+18) to get the "TWENTY" or "NINETY" or whatever it happens to be, plus Z\$(Z(Z)), which would get us the "THREE" in our case or whatever it happens to be - within the limits of one to nine.

In line 480 we again check to see if the length of ZB\$ is 2. If it is, we are done and go to the output section again. If it isn't, we have to check for hundreds. This one is easy. If the hundreds digit is zero we can just simply forget it. Otherwise, we look at Z(Z-2), the hundreds digit, and go to the Z\$(Z) array and get the number, add the word "HUNDRED" and then add what we have so far in ZZ\$. In our case, it would now say: TWO HUNDRED THREE."

Next, we check the length of ZB\$ again to see if it is three characters long. If it is, we are again done. If not, we have to check for thousands. We have the same situation here as we had with the

units and tens, earlier. Only this time we append the word "THOUSAND" to the word captured from the Z\$(Z) array, then append it to the front of whatever we have so far in ZZ\$. The result, as per our example, would be: SIXTY ONE THOUSAND TWO HUNDRED THREE. Then, in line 540, we add leading asterisks to ZZ\$ and add the cents from ZC\$. The final result would be: **** SIXTY ONE THOUSAND TWO HUNDRED THREE & NO/100. The check blank itself will add the word DOLLARS. Line 550 simply prints the line on the screen for you to see.

To make a subroutine of this:

First off, you can eliminate the remarks at the beginning. Then, put the DIM statement and the data statements, as well as the loop that reads the data statements, at the beginning of your main program. All that stuff needs only to be executed once. In fact, you'll get an error if you try to read the data statements twice (Out of DATA error).

You can start your subroutine with line 250. You don't need 240 any longer either; just make sure that you send the dollar amount to the subroutine as integer ZA. In line 250, you can change the END to a RETURN, or a message that the ZA dollar amount was out of limits.

The remainder of the code remains intact as a subroutine, except for line 550, where you don't want to print the answer, but return to the calling routine and let it handle it. So change line 550 to RETURN. Just remember that variable ZZ\$ contains the answer. In addition to all this, you would most certainly want to renumber the subroutine somewhere in the high numbers, to fit into your calling program. If you type this program in, pay special attention to the zeros and the Oh's. I'll try and remember to list this for these pages on a printer that slashes the zeros.

I don't ever remember writing so much explanation for such a small amount of code. Perhaps it's because so much is happening there. In any case, all these ZZZ's are making me sleepy - maybe it's time to go cop a few.

Ckrite.Bas for all models.

```
100 REM * Ckrite.Bas * changes numerals into words
110 REM * can be modified as a subroutine to write checks
120 REM * Input ZA as an integer and get ZZ$ out in words.
130 DIM Z$(27)
140 '
150 DATA "ONE ", "TWO ", "THREE ", "FOUR ", "FIVE ", "SIX ", "SEVEN "
160 DATA "EIGHT ", "NINE ", "TEN ", "ELEVEN ", "TWELVE ", "THIRTEEN "
170 DATA "FOURTEEN ", "FIFTEEN ", "SIXTEEN ", "SEVENTEEN "
180 DATA "EIGHTEEN ", "NINETEEN ", "TWENTY ", "THIRTY ", "FORTY "
190 DATA "FIFTY ", "SIXTY ", "SEVENTY ", "EIGHTY ", "NINETY "
200 FOR Z = 1 TO 27
210   READ Z$(Z)
220 NEXT Z
230 '
240 INPUT "What is your amount "; ZA
250 IF ZA>99999.99 OR ZA<1 THEN END
260 ZA#=ZA
270 IF INT(ZA#)<>ZA# THEN ZA#=ZA#+.005
280 ZA$=STR$(ZA#)
290 IF INT(ZA#)=ZA# THEN ZA$=ZA$+".00"
300 ZX=INSTR(ZA$,".")
310 ZC$=MID$(ZA$,ZX+1,2)
320 ZF=INSTR(ZC$,"0"):IF ZF<>0 THEN MID$(ZC$,ZF,1)="0":GOTO 320
330 IF LEN(ZC$)=1 THEN ZC$=ZC$+"0"
340 IF ZC$="00" THEN ZC$="NO"
350 ZC$="& "+ZC$+"/100"
360 '
370 ZB$=STR$(INT(ZA#))
380 ZB$=RIGHT$(ZB$,LEN(ZB$)-1)
390 ZL=LEN(ZB$)
400 FOR Z= 1 TO ZL
410   Z(Z)=VAL(MID$(ZB$,Z,1))
420 NEXT Z
430 Z=Z-1
440 IF Z(Z-1)=0 THEN ZZ$=Z$(Z(Z))
450 IF Z(Z-1)=1 THEN ZZ$=Z$(Z(Z)+10)
460 IF ZL=1 THEN 540
470 IF Z(Z-1)>1 THEN ZZ$=Z$(Z(Z-1)+18)+Z$(Z(Z))
480 IF ZL=2 THEN 540
490 IF Z(Z-2)<>0 THEN ZZ$=Z$(Z(Z-2))+"HUNDRED "+ZZ$
500 IF ZL=3 THEN 540
510 IF Z(Z-4)=0 THEN ZZ$=Z$(Z(Z-3))+"THOUSAND "+ZZ$
520 IF Z(Z-4)=1 THEN ZZ$=Z$(Z(Z-3)+10)+"THOUSAND "+ZZ$
530 IF Z(Z-4)>1 THEN ZZ$=Z$(Z(Z-4)+18)+Z$(Z(Z-3))+"THOUSAND "+ZZ$
540 ZZ$="**** "+ZZ$+ZC$
550 PRINT ZZ$
```

Progest.Bas

A Program Estimating Program

Terry Dettmann, Associate Editor. This is another vintage program by Terry. It was originally published in 80-U.S. Journal, in January 1981, but what it does still holds up today. There is nothing unusual about the code and little or no modification is necessary for the various computer models.

Believe it or not, programming is a business too. You would be surprised at how many people who are doing it for money aren't treating it like a business.

Most of the business computer tools are pretty standard. Receivables, payables, ledger, etc., even for a programming business. But how do you estimate a job?

Have you ever gone to a business, been given detailed specifications on a job you want to do, and then been asked for an estimate? You probably have if you are trying to program for pay. What's more, you probably wound up taking a figure off the top of your head and saying, "Here's my estimate."

It almost seems as if it's a national pastime for businessmen to put programmers on the spot. Most occupations that require job estimates have some sort of detailed program estimating technique. Carpenters, bricklayers, all of them, have estimating aids. Now it's time for the programmer to have such an aid.

The program with this article grew out of a reading (and multiple re-readings) of the book *The Program Development Process - Part 1, The Individual Programmer* by Joel D. Aron. You probably can't find this book outside of a major computing center library, but it is part of a well-known series of books, called the *System Pro-*

gramming Series, published by Addison-Wesley.

The series is sponsored by IBM and is aimed mostly at large system programmers. It includes volumes on programs, programming, databases, compilers, interactive graphics, sorting and recursive programming. In all, there are 11 books in the series.

In the program development process, Aron goes over the entire process of developing programs, emphasizing what the individual programmer should do in order to work most efficiently. Chapter 3 deals with problem analysis and planning.

When I first read the book, I was quite impressed by the planning chapter, in particular by the attempt to provide the programmer with some real numbers for estimating the time it would take to do a job.

As the book points out, there is no real agreement about what these numbers really should be, but several tables are provided that are taken from published studies of the program development process. These tables form the basis for the program listing with this article.

Estimating a Job

In order to estimate a job, it is necessary for the programmer to first estimate the difficulty of

various parts of the job. The book (and our program) first breaks the job down into its attributes, starting with input/output.

Four characteristics of the input and output are measured and assigned appropriate weights. They are:

1. The number of record types of **fixed format**. That is, the number of different types of records of information that do not change in format.

2. The number of **variable format** records. These are records whose character or size changes during the running of the program.

3. The number of **commands, messages or inquiries** that the program will have to handle. For example, how many commands can be entered from the keyboard to control execution?

4. The number of **special devices** used for the program. By special devices, we mean things which are not normal connections to the standard computer system (burglar alarms, etc.)

Next the program gets information about storage requirement for the program:

1. The number of arrays that will be used by the program.

2. The number of files that will be used during the operation of the program.

3. The number of files that will be used that have a special structure. For example, how many files are stored as linked lists?

4. The number of multiple file relationships that the program will have to handle. For example, how many files will have information moved from one to another during program execution.

Now we have to find out what the processing objectives are for the program:

1. The program asks for a number from zero to 10 to indicate the real-time performance objectives. A program that would be expected to give immediate displays of any item in an inventory would rate a higher degree of difficulty than one that allows a search for the item before displaying it.

2. The objectives for data communications are requirements for computer to computer or computer to terminal information transfer. Normally, this will be a zero.

3. The importance of graphics displays is next. Obviously, graphics are very important to a game, but far less important to most business programs.

Still under the processing information section, the book assigns a whole table to determining the effect of the choice of a language on the program. Clearly, if you choose to program in Assembly Language, it will take you longer to code the program than if you chose to do it in a high level language (unless you are a super-programmer!).

The degree of difficulty though, will depend on the nature of the program and the assembler or interpreter you use. To account for that, the book (and the program) assigns weights based on whether the program is prepared in Assembly Language, a high-level language or from existing modules. In addition, weights are assigned based on how difficult that language is to use.

Next the program asks for your own qualifications as a programmer and your knowledge of the specific job you are going to program. The length of time needed to complete the job is obviously much greater for a trainee than for a senior programmer (6 times greater by the weights given in the book).

Once all of the questions are asked, the program will display the total time in man-days to complete the job and the suggested breakdown

of days to complete it. The breakdown is based on the complexity of the program. For a very complex program, more time is assigned to design on a percentage basis.

Some items in the book's tables were left out of the program intentionally (such as shared arrays) since they aren't used that much in micro-computer programming. This doesn't affect the final results since the weights for these items would have been zero for a typical project.

I have compared the output from the program to the results of actual jobs (I won't say what kind of programmer I was!). Amazingly, even without modifying the weights from the book, the estimates I got came out close to the actual time required to do a project.

In one case, I completed a project with approximately two man-months of effort that the program estimated 60 man-days for! I can't argue with that.

Despite the agreement in my own tests, this program can hardly be considered a really accurate estimate for all cases. It does, however, give the programmer a definite feel for the length of time needed to complete a program and forms the basis for a reasonable estimate for estimating a job. It can also make an impressive display for a customer when you use his system to estimate the program development time in front of his very eyes.

As a check to see if you have entered this program correctly, here is a check: Answer every question with a 1 and you should get an estimate of 14.5 man-days to complete the job. It should be split up to 5 days for design, 3.6 days for coding and developing test data, 5 days to debug and 0.75 days to prepare documentation.

(Editors note: Having written numerous programs and written articles describing them over the past few years, it appears that the time allowed for documentation for a 5 man-day program is inadequate. Five days to debug a 5 man-day program sure is realistic, however.)

Progest.Bas for all models

```
100 REM *****
110 REM          Program Job Estimating  Progest.Bas
120 REM          Version 1.0
130 REM          Terry R. Dettmann
140 REM *****
150 `
160 REM initialization
170 `CLEAR 1000
180 DEFINT A-Z: DIM PD! (3,4), PEW! (14), LC! (3,4,6), EL! (4), UN! (3,5)
190 REM arrays:
200 REM          PD!      program development activities
210 REM          PEW!     program unit estimating weights
220 REM          LC!     language capability weights
230 REM          EL!     programmer experience weights
240 REM          UN!     job uniqueness weights
```

```

250 REM
260 REM read the weights from data statements
270 FOR I=1 TO 3
280   FOR J=1 TO 4
290     READ PD!(I,J)
300   NEXT J
310 NEXT I
320 `
330 FOR I=1 TO 14
340   READ PEW!(I)
350 NEXT I
360 `
370 FOR I=1 TO 3
380   FOR J=1 TO 4
390     FOR K=1 TO 6
400       READ LC!(I,J,K)
410     NEXT K
420   NEXT J
430 NEXT I
440 `
450 FOR I=1 TO 4
460   READ EL!(I)
470 NEXT I
480 `
490 FOR I=1 TO 3
500   FOR J=1 TO 5
510     READ UN!(I,J)
520   NEXT J
530 NEXT I
540 `
550 REM initialize the days to zero and define graphics line
560 `
570 DAYS!=0
580 HDR$=STRING$(60,45)
590 `
600 REM ---- begin estimate ----
610 REM what are the I/O characteristics of the program?
620 `
630 CLS:PRINT HDR$:PRINT TAB(25);"Estimating":PRINT HDR$
640 PRINT TAB(5);"I/O Characteristics"
650 PRINT TAB(10);:INPUT "Number of fixed format records";N
660 DAYS!=PEW!(1)*N + DAYS!
670 PRINT TAB(10);:INPUT "Number of variable format records ";N
680 DAYS!=DAYS!+PEW!(2)*N
690 PRINT TAB(10);:INPUT "Number of commands, messages and inquiries ";N
700 DAYS!=DAYS! + PEW!(3)*N
710 PRINT TAB(10);:INPUT "Number of special devices ";N
720 DAYS!=DAYS! + N*PEW!(4)

```

```

730 `
740 REM what are the storage requirements for the program?
750 `
760 CLS:PRINT HDR$:PRINT TAB(25);''Estimating'':PRINT HDR$
770 PRINT TAB(5);''Storage''
780 PRINT TAB(10);:INPUT ``Number of arrays used ``;N
790 DAYS!=DAYS! + N*PEW!(5)
800 PRINT TAB(10);:INPUT ``Number of files to be used ``;N
810 DAYS!=DAYS! + N*PEW!(6)
820 PRINT TAB(10);:INPUT ``Number of files with list structures or overflows
``;N
830 DAYS!=DAYS! + N*PEW!(7)
840 PRINT TAB(10);:INPUT ``Number of multiple file relationships ``;N
850 DAYS!=DAYS! + N*PEW(8)
860 `
870 REM what level of processing difficulty will the program have?
880 `
890 CLS:PRINT HDR$:PRINT TAB(25);''Estimating'':PRINT HDR$
900 PRINT TAB(5);''Processing''
910 PRINT TAB(10);''Enter a number in the range requested for each''
920 PRINT TAB(10);''question asked ``:PRINT
930 PRINT TAB(10);:INPUT ``Real-time performance objectives (0-10) ``;N
940 DAYS!=DAYS! + N
950 PRINT TAB(10);:INPUT ``Data communications (0-5) ``;N
960 DAYS!=DAYS! + N
970 PRINT TAB(10);:INPUT ``Graphic displays (0-10) ``;N
980 DAYS!=DAYS! + N
990 `
1000 REM what are the capabilities of the programming language being used?
1010 `
1020 CLS:PRINT HDR$:PRINT TAB(25);''Estimating'':PRINT HDR$
1030 PRINT TAB(5);''Language Capability''
1040 PRINT TAB(10);:PRINT ``(1) Assembly language or (2) high level language''
1050 PRINT TAB(10);:INPUT ``or Packages ``;L
1060 IF L=3 THEN 1190
1070 PRINT:PRINT TAB(10);''For each of the following questions'':PRINT
TAB(10);''answer (1) Easy (2) Medium (3) Hard (4) Very hard''
1080 PRINT TAB(15);:INPUT ``Restructuring data'';N
1090 DAYS!=DAYS! + LC!(L,N,1)
1100 PRINT TAB(15);:INPUT ``Monitor status ``;N
1110 DAYS!=DAYS! + LC!(L,N,2)
1120 PRINT TAB(15);:INPUT ``Retrieve and present data ``;N
1130 DAYS!=DAYS! + LC!(L,N,3)
1140 PRINT TAB(15);:INPUT ``Calculations ``;N
1150 DAYS!=DAYS! + LC!(L,N,4)
1160 PRINT TAB(15);:INPUT ``Program linkage ``;N
1170 DAYS!=DAYS! + LC!(L,N,5)

```

Handy Order Form

```
1180 GOTO 1280
1190 PRINT TAB(5);''Package Programming''
1200 PRINT TAB(10);''Will you use a (1) Program package, (2) Utility
1210 PRINT TAB(10);''program or (3) a report program generator ``;:INPUT N1
1220 PRINT:PRINT TAB(1);''Will you (1) Prepare control cards only''
1230 PRINT TAB(10);''or (2) add your own code ``;:INPUT N2
1240 PRINT TAB(10);''Is this project (1) easy (2) medium (3) hard''
1250 PRINT TAB(10);''or (4) very hard ``;:INPUT N3
1260 DAYS!=DAYS! + LC!(L,N3,2*(N1-1)+N2)
1270 `
1280 REM ---- know how -----
1290 REM how good is the programmer?
1300 `
1310 CLS:PRINT HDR$:PRINT TAB(25);''Estimating'':PRINT HDR$
1320 PRINT TAB(5);''Know how''
1330 PRINT TAB(10);''What is your experience level?''
1340 PRINT
1350 PRINT TAB(15);'' 1 - Senior programmer
1360 PRINT TAB(15);'' 2 - Programmer
1370 PRINT TAB(15);'' 3 - Apprentice
1380 PRINT TAB(15);'' 4 - Trainee
1390 PRINT:PRINT TAB(10);:INPUT ``Level ``;EX
1400 `
1410 REM ---- how unique is the job ----
1420 `
1430 CLS:PRINT HDR$:PRINT TAB(25);''Estimating'':PRINT HDR$
1440 PRINT TAB(5);''Job Uniqueness''
1450 PRINT TAB(10);''How much job knowledge is required?''
1460 PRINT TAB(10);''(1) much (2) some (3) none''::INPUT N1
1470 PRINT TAB(10);''How much do you have available?''
1480 PRINT TAB(15);'' 1 - Detailed knowledge''
1490 PRINT TAB(15);'' 2 - Good general with fragmentary detailed''
1500 PRINT TAB(15);'' 3 - Fair general with little or no detailed''
1510 PRINT TAB(15);'' 4 - No detailed but general knowledge''
1520 PRINT TAB(15);'' 5 - No detailed or general knowledge''
1530 PRINT TAB(10);:INPUT ``Knowledge level ``;N2
1540 EW!=EL!(EX) + UN!(N1,N2)
1550 MD!=EW!*DAYS!
1560 PRINT:PRINT TAB(10);''What kind of logic is involved in the program''
1570 PRINT TAB(10);:INPUT ``(1) average (2) complex (3) complex control ``;LG
1580 `
1590 REM ---- display estimate ----
1600 `
1610 CLS:PRINT HDR$:PRINT TAB(25);''Job estimate ``;:PRINT HDR$
1620 PRINT TAB(10);''The estimate for this job is ``;MD!;'' man days''
1630 PRINT
1640 PRINT TAB(10);''These days will be split up as follows:''
```

```

1650 PRINT TAB(15);MD!*PD!(LG,1);'' days for design''
1660 PRINT TAB(15);MD!*PD!(LG,2);'' days for coding and developing test data''
1670 PRINT TAB(15);MD!*PD!(LG,3);'' days for debugging''
1680 PRINT TAB(15);MD!*PD!(LG,4);'' days preparing documentation''
1690 PRINT
1700 PRINT''Estimate Completed.''
1710 `
1720 END
1730 REM ---- estimating weight data ----
1740 DATA .35, .25, .35, .05, .4, .2, .35, .05, .35, .2, .4, .05
1750 DATA 1,2,1,1,1,3,5,2,0,0,0,0,1,2
1760 DATA 3,3,2,2,2,0,4,5,4,3,3,0,5,7,6,5,4,0,6,9,8,7,5,0
1770 DATA 1,1,1,1,1,0,2,2,2,2,2,0,3,4,4,3,3,0,4,6,6,4,4,0
1780 DATA 2,2,1,2,1,2,3,4,1,4,1,4,3,8,1,4,1,8,3,16,1,4,1,8
1790 DATA .5, 1, 1.5, 3
1800 DATA 0, .25, .5, .75, 1, 0, 0, .25, .5, .75, 0, 0, 0, .25, .25

```

Things, continued from page 10

In his letter, Bob said that he suspected we would get a great deal of variety on this one and he was right, we did.

Robert B Franke sent in an interesting set of programs using For..Next and While..Wend to do the job. It seems that some of the other BASICs are integer limited at 32,767 according to Mr Franke, and he had to play games with his For..Next loops by adding to the loop counter.

David Leithauser, whose articles and programs have appeared in past issues, sent in the following:

"Enclosed is a program that solves the problem of making labels for the parts bins. The secret is to not actually create a number containing the check digit, but simply print the check digit after the number. Line 20 computes the check digit. I used this equation because my computer does not like to use the MOD function on large numbers. Line 30 prints the number followed by the check digit. The STR\$ function is used to remove spaces before and after the numbers. The extra LPRINT statements in line 40 are used to advance the printer to the next label."

His program follows:

```

10 FOR X=1000000! TO 1050000!
20 Y=INT((X/7-INT(X/7))*7+.5)
30 LPRINT STR$(X);MID$(STR$(Y),2)
40 LPRINT:LPRINT:LPRINT
50 NEXT X

```

Which is surprisingly close to what we came up with in QuickBASIC:

```

CLS
FOR i = 1000000! TO 1050000!
  a = i MOD 7
  LPRINT STR$(i); LTRIM$(STR$(a))
  FOR j = 1 TO 5: LPRINT '' ': NEXT j
NEXT i

```

LTRIM\$ is a neat feature in QuickBASIC. It will strip off spaces to the left of a string. RTRIM\$ does the same for the right side of a string. As in Mr. Leithauser's program, the J loop is to advance to the next label.

Bill Seugling suggested that the problem seems trivial compared to some of our old puzzlers. We agree, now that we've had a good look at it.

On to a new subject. Robert Hood sent us a cute little program that shows you all the possible color combinations (assuming you have a color monitor and MS-DOS).

(more *Things* on page 40)

Handy Order Form

- RENEW SUBSCRIPTION:**
Nov/Dec 89 through Sep/Oct 90 _____ \$24.95
- All 4th year issues:
Nov 88 through Sep 89 _____ \$18.00
- All 3rd year issues:
Nov 87 through Sep 88 _____ \$18.00
- All 2nd year issues:*
Nov 86 through Sep 87 _____ \$18.00
- All 1st year issues:
Sep 85 through Sep 86 _____ \$18.00
- DISKS (specify year and computer type) _____ \$15.00
4th year disk will be ready Sep 1, 1989 Year(s) _____
- "Starting with MS DOS" booklet _____ \$7.00

*Note
new
lower
prices
on back
issues
and all
disks!*

Postage and handling charges already included.

Diskettes are available for MS DOS, Tandy IV, Tandy III and most CP/M formats ***Please specify your computer type!***

*In year 2 issues, Issue 8 is out of print and will be supplied on diskette. Please specify your computer type if ordering 2nd year issues.

Computer type: _____

- Check/MO enclosed
- Charge to VISA/MC _____ Exp _____

Name _____

Address _____

City/State/Zip _____

Clip or photocopy and mail to: **CodeWorks, 3838 South Warner Street,
Tacoma, Washington 98409**

**We accept VISA & MasterCard. You may call in your order:
(206) 475-2219 Thank you.**

1189

Index

And things that won't fit elsewhere

Misc, program, eliminate random doubles, issue 25, page 3

Beginning BASIC, part 3 of All About Strings, issue 25, page 5

Misc, program counts words, lines and characters, issue 25, page 7

Animal.bas, main program, issue 25, page 12, AI demo program

Expert4.bas, main program, issue 25, page 13, AI demo program

Drill.bas, main program, issue 25, page 19, makes and runs drills

Addbook.bas, main program, issue 25, page 24, prints address booklet

Cardconv.bas, main program, issue 25, page 28, converts Card.Dat files

Cardconv.bas, main program, issue 25, page 29, QuickBASIC version

NFL89.bas, main program, issue 25, page 31, NFL predictor for 1989-90

Stat89.bas, main program, issue 25, page 35, keeps stats for NFL89.bas

Things, about how to print large numbers, issue 25, page 40

Here is his program. Don't be surprised when your screen starts flashing. The higher color numbers cause blinking. When the program is done, it will continue to blink. To get rid of the blink, type in: SCREEN 0,0,0 and it will go away.

```
5 REM * ctest.bas * by Robert Hood
10 CLS
20 FOR J=0 TO 255
30 DEF SEG:POKE 78,J
40 PRINT USING"###";J;:PRINT" Poke 78
Color Test"
50 FOR K=0 TO 1000:NEXT K
60 NEXT J
70 POKE 78,7
80 FOR K=0 TO 1000:NEXT K
90 CLS
```

Meanwhile, we recently watched a NOVA program on PBS concerning the order in chaos. It suggested a little program to prove that there is order in apparently random happenings. Whether it works or not is one subject for the next issue's Things.

CodeWorks

3838 South Warner Street
Tacoma, Washington 98409

Bulk Rate
U.S. Postage
PAID
Permit 774
Tacoma, WA

ERICKSON, MIKE / KRJB 2910
BOX 250
MONTE RIO CA 95462

NOTICE

This is the first issue of the new subscription year. If you have received this notice, it means we have not yet received your renewal order.

We are sending this issue now because we know that you had *not* intended for your subscription to run out — but it has ...

Send your personal check or credit card information **TODAY** so you may continue to enjoy and benefit from a subscription to CodeWorks.

Please use the order form on page 39 of this issue, and include your comments so that we may plan an exciting year of issues for you.

Thank you,

The CodeWorkers

```

{ calc/pcl }
  { from Pascal for Basic Programmers
    by C. Seiter and R. Weiss, 1983 }
program calc;
  { declare global objects }
const  namelen = 4;  varlen = 8;  errlen = 12;
type   alfa = array [1..errlen] of CHAR;
       Valfa = array [1..varlen] of CHAR;
       funcnam = (ABSnam, SQRTnam, EXPnam,
                  SINnam, COSnam, TANnam,
                  ATANnam, ASINnam, ACOSnam, LNnam,
                  LOGnam, INTnam, FACnam, SINHnam,
                  COSHnam);
       V_ptr   = @V_item;
       V_item  = RECORD
                 nextvar : V_ptr;
                 Vname   : Valfa;
                 value   : real;
               end;
       N_Array = ARRAY [funcnam] OF ARRAY [1..namelen] OF
                 CHAR;
       {
function LEN(S:string): integer; external;
function MID$(S:string; POS,LENGTH:integer): string; external;
function DECODER(S:string) :real; external;
function CHARACTER(S:string; POS:integer):CHAR; external;
function CPYSTR(S:string):string; external;
function CONC(S1,S2:string):string; external;
function DELETE(S:string; POS,LENGTH:integer):string; external;
function FIND(SUBS, S:string): integer; external;
function INSERT(SUB,S:string; POS:integer):string; external;
function REPLACE(OLDS,NEWES,S:string):string; external;
       { }

function EVAL(EXPR:string; var BooBoo:boolean;
  first: V_ptr; var names: N_Array): real;
var   ch, token : CHAR;
      num : string;
      x : real;
      ptr : V_ptr;
      P, P0, t, offset,length : integer;
      uppercase, lowercase, digit : SET OF CHAR;
procedure error(a: alfa); FORWARD;
       {
function EOS : boolean;
begin
  if P > LENGTH then EOS := TRUE
  else EOS := FALSE;
end;
       {
procedure SKIP;
  { skip blanks }
begin
  WHILE CHARACTER(EXPR,P)=' ' AND NOT(EOS) DO
    P := P + 1;
  if NOT(EOS) then
    token := CHARACTER(EXPR,P);
end;
       { end skip }
       {
function expression : real;
var op : CHAR;  oldsum, newnum : real;
function term : real;
var op : CHAR;  oldsum, newnum : real;
function power : real;

```

```

var oldsum, newnum : real;
function factor : real;
var sign : real; i : integer;
function func : real;
CONST piby2 = 1.570796;
var x, t : real; funcID : funcnam;
found : boolean;
function name(var found:boolean): funcnam;
var j,pl: integer; I : funcnam;
done : boolean; tok : char;
chars : ARRAY [1..4] OF CHAR;
begin
pl := p; tok := token;
if tok IN uppercase + lowercase then
FOR j := 1 TO namelen DO
if tok IN uppercase + lowercase then
begin
chars[j] := tok;
Pl:=Pl+1; tok :=CHARACTER(EXPR,pl);
if chars[j] in lowercase then
chars[j] := CHR(ORD(chars[j])+offset)
end
else chars[j] := ' ';
{ look up name in array 'names' }
found := FALSE;
done := FALSE;
i := ABSnam;
WHILE NOT found AND NOT done DO
if chars = names[i] then found := TRUE
else if i < COSHnam then i := SUCC(i)
else done := TRUE;
if found then begin
P := pl; { update pointer if found }
token := tok;

end;
name := I;
end; { end name }
{ ** }
function variable : real;
var j : integer; ptr : V_ptr;
done : boolean;
chars : ARRAY [1..varlen] OF CHAR;
finished : boolean;
begin
if token IN uppercase + lowercase then
FOR j := 1 TO varlen DO
if token IN uppercase + lowercase + digit then
begin
chars[j] := token;
P:=P+1; token :=CHARACTER(EXPR,p);
if chars[j] in lowercase then
chars[j] := CHR(ORD(chars[j])+offset)
end
else chars[j] := ' ';
{ look up name in variable list }
ptr := first; finished := FALSE;
WHILE ptr<>NIL AND NOT(finished) DO
if chars = ptr@.Vname then finished := TRUE
else ptr := ptr@.nextvar;
if NOT(finished) then
ERROR('unknown name')
else
variable := ptr@.value; end; {end variable}

```

```

function FAC(x:real): real;
var i,up : integer; T : real;
begin
UP := ROUND(X); T := 1;
if UP>0 then
FOR I := 1 TO UP DO
T := T * I;
FAC := T;
end;
begin { begin func }
funcID := name(found);
if found then
begin
if BooBoo then ESCAPE;
skip;
{ the function argument must appear in
parentheses}
if token = '(' then
begin
p := p + 1; token := CHARACTER(EXPR,P);
skip;
x := expression;
if BooBoo then ESCAPE;
if token = ')' then
begin
P := P + 1;
token := CHARACTER(EXPR,P);
end
else ERROR('missing RPAR')
end
else
ERROR('missing LPAR');
{ evaluate arithmetic function --
the argument has been calculated and

```

placed in the variable 'x'. For some functions, check for out-of-range arguments or special cases. }

```

CASE funcID OF
ABSnam: func := ABS(x);
SQRTnam: if x >= 0 then func := SQRT(x)
else ERROR('arg range ');
EXPnam: func := EXP(x);
SINnam: func := SIN(x);
COSnam: func := COS(x);
TANnam: if cos(x)<>0 then func:=SIN(x)/COS(x)
else ERROR ('tangent arg ');
ATANnam: func := ARCTAN(x);
ASINnam: if ABS(x)>1 then ERROR('asin range ')
else if ABS(x)=1 then func:=pi*2*x
else
func := ARCTAN(x/SQRT(1-SQR(x)));
ACOSnam: if ABS(x)>1 then
ERROR('acos range ')
else if x=0 OR ABS(x)=1
then func := pi*2*(1-x)
else begin
T := ARCTAN(SQRT(1-SQR(x))/X);
if x>0 then func := T
else
func := 2 * pi*2 * T
end;
LNnam: if x>0 then func := LN(x)
else ERROR('LN arg ');
LOGnam: if x>0 then func := LN(x)/2.302585093

```

```

INTnam: func := ROUND(x);
FACnam: func := FAC(x);
SINHnam: func := (EXP(x)-EXP(-x))/2;
COSHnam: func := (EXP(x)+EXP(-x))/2;
end { end CASE }
end { end if found }
else { variable? }
  func := variable;
end; { end function func }
begin { begin function factor }
  { check for leading + or - sign }
  if token = '-' then begin
    sign := -1;
    P:=P+1; token := CHARACTER(EXPR,P);
  end else begin
    if token = '+' then
      begin
        P:=P+1; token := CHARACTER(EXPR,P);
      end;
    sign := 1;
  end;
skip;
{ evaluate parenthesized subexpression }
if token = '(' then begin
  P := P + 1; token := CHARACTER(EXPR,P);
  skip;
  x := expression;
  if BooBoo then ESCAPE;
  if token = ')' then
    begin
      P := P + 1; token := CHARACTER(EXPR,P);
    end
  else ERROR('missing RPAR');
end

end
else if token in uppercase + lowercase then
  begin x := func;
  if BooBoo then ESCAPE
  end
else if token in digit then
  begin { get a number }
    P0 := P;
    WHILE token in digit AND NOT(P>=LENGTH) DO
      begin
        P := P + 1;
        token := CHARACTER(EXPR,P);
      end;
    num := MID$(EXPR,P0,P-P0+1);
    x := DECODER(num);
  end
  else if NOT (EOS) then ERROR('bad factor ');
skip;
  { apply earlier determined sign }
  factor := sign * x;
end; { end function factor }
begin { begin function power }
  oldsum := factor;
  if BooBoo then ESCAPE;
  WHILE token = '[' DO begin
    P := P + 1; token := CHARACTER(EXPR,P);
    skip;
    newnum := factor;
    if BooBoo then ESCAPE;
    if oldsum > 0 then oldsum := EXP(newnum * LN(oldsum))

```

else ERROR('LOG arg '); 4.

```

else if newnum = 0 then oldsum := 1
else ERROR('bad exponent');
end;
power := oldsum;
end; { end power }
begin { begin term }
oldsum := power;
if BooBoo then ESCAPE;
WHILE token IN ['*', '/'] DO begin
op := token;
P := P + 1; token := CHARACTER(EXPR,P);
skip;
newnum := power;
if BooBoo then ESCAPE;
CASE op OF
'*': oldsum := oldsum * newnum;
'/': if newnum=0 then ERROR('divide by 0 ')
else oldsum := oldsum / newnum;
end;
end;
term := oldsum;
end; { end term }
begin { begin expression }
oldsum := term;
if BooBoo then ESCAPE;
WHILE token IN ['+', '-'] DO begin
op := token;
P := P + 1; token := CHARACTER(EXPR,P);
skip;
newnum := term;
if BooBoo then ESCAPE;
CASE op OF
'+': oldsum := oldsum + newnum;
'-': oldsum := oldsum - newnum;
end;
end;
expression := oldsum;
end;
procedure ERROR;
begin
BooBoo := TRUE;
WRITELN; WRITELN('ERROR! ',A);
end;
{ }
begin { function EVAL }
{ for translation from lower to upper case }
offset := ORD('A')-ord('a');
uppercase := ['A'..'Z', '##']; lowercase := ['a'..'z'];
digit := ['0'..'9', '.'];
BooBoo := FALSE;
P := 1;
length := LEN(EXPR);
skip;
eval := expression;
{ }
end;
begin
{$NULLBODY}
end.

```