CodeWorks 1st Year Programs for TRS-80 Models I/III

The diskettes provided contain the following programs. The number after the file name indicates the size of the program in bytes.

We have made every effort to make these programs run on the Models I/III "as is". MUMP BAS will not work on the Models I/III. Refer to the appropriate issue for programs that require initialization.

Please do not remove our identification lines in the programs, they serve to show where these programs come from and will insure that we stay in business and continue to produce more good and economical software.

ADD	BAS	1573	Issue 7 Addition program works with Drill.Bas.
ADUMP	BAS	1011	Issue 6 An ASCII memory dump program.
BUSMOD	BAS	3328	Issue 5 A business "what if" model.
CAL	BAS	6998	Issue 2 Perpetual (almost) calendar maker program.
CARD	BAS	13280	Issue 2 In memory, mini-database, requires initialization.
CHECK	BAS	6601	Issue 4 Card.Bas adapted to keeping check records.
CONV	BAS	2888	Issue 5 Convert DATA statements to seq. file and back.
DIV	BAS	1549	Issue 7 Division program works with Drill.Bas.
DRILL	BAS	1220	Issue 7 Collection of Instructor's drill programs.
DSTAT	BAS	4768	Issue 6 A descriptive statistics program.
EXTR	BAS	2857	Issue 1 Extracts program lines for later use.
FMAKER	BAS	1558	Issue 7 A general purpose sequential file maker pgm.
HALLEY	BAS	4108	Issue 2 Plotted the position of Halley's Comet.
MADA	BAS	906	Issue 7 Mental addition pgm works with Drill.Bas.
MADAS	BAS	1174	Issue 8 Mental Add/Sub pgm works with Drill.Bas.
MAKER	BAS	2432	Issue 1 Makes DATA statements the easy way.
MATH	BAS	4992	Issue 4 Math formulae on screen that change.
MGSORT	BAS	5007	Issue 5 Merge/sort files too big for ram memory.
MULT	BAS	1727	Issue 7 Multiplication program works with Drill.Bas.
MUMP	BAS	755	Issue 7 An HEX dump pgm for MS-DOS. Dumps up to 640K.
MURRAY	BAS	5752	Issue 3 Program to improve calculating accuracy.
NETWORK	BAS	10359	Issue 6 A game for up to 4 players. Buy/Sell stock.
NFL86	BAS	6859	Issue 7 An NFL football projection program.
NFLSTAT	BAS	3900	Issue 7 Maintains NFL statistics for NFL86.
NORRIS	BAS	4088	Issues 2&4 A collection of file conversion routines.
PAY	BAS	13568	Issue 4 Payroll program, requires initialization.
PLIST	BAS	2698	Issue 6 A neat page-listing utility program.
PLOT	BAS	5883	Issue 1 Prints histograms with 3 period projection.
PRECOMP	BAS	3320	Issue 4 A BASIC precompiler w/o line #, with labels.
PROG1	BAS	1298	Issue 4 Sequential file tutorial demo program.
PROG2	BAS	893	Issue 4 Sequential file tutorial demo program.
RANDEMO	BAS	3037	Issue 6 A random file tutorial demo program.
RANDEMO2	BAS	2020	Issue 7 A random file tutorial demo program (#2).
SEARCH	BAS	2644	Issue 1 Finds selected variables in target program.
SPELL	BAS	2564	Issue 7 Spelling program works with Drill.Bas.
SUB	BAS	1562	Issue 7 Subtraction program works with Drill.Bas.
UPDATE	BAS	1574	Issue 7 Updates spelling for Drill.Bas.
VXREF	BAS	14464	Issue 5 A BASIC cross reference program.
WOOD	BAS	11135	Issue 3 Best fit program for cutting plywood, etc.
WRITER	BAS	10629	Issue 1 Analyze your writing (for 80 col screens.)

CodeWorks 1st Year Programs for TRS-80 Models I/III

The diskettes provided contain the following programs. The number after the file name indicates the size of the program in bytes.

We have made every effort to make these programs run on the Models I/III "as is". MUMP BAS will not work on the Models I/III. Refer to the appropriate issue for programs that require initialization.

Please do not remove our identification lines in the programs, they serve to show where these programs come from and will insure that we stay in business and continue to produce more good and economical software.

ADD	BAS	1573	Issue 7 Addition program works with Drill.Bas.
ADUMP	BAS	1011	Issue 6 An ASCII memory dump program.
BUSMOD	BAS	3328	Issue 5 A business "what if" model.
CAL	BAS	6998	Issue 2 Perpetual (almost) calendar maker program.
CARD	BAS	13280	Issue 2 In memory, mini-database, requires initialization.
CHECK	BAS	6601	Issue 4 Card.Bas adapted to keeping check records.
CONV	BAS	2888	Issue 5 Convert DATA statements to seq. file and back.
DIV	BAS	1549	Issue 7 Division program works with Drill.Bas.
DRILL	BAS	1220	Issue 7 Collection of Instructor's drill programs.
DSTAT	BAS	4768	Issue 6 A descriptive statistics program.
EXTR	BAS	2857	Issue 1 Extracts program lines for later use.
FMAKER	BAS	1558	Issue 7 A general purpose sequential file maker pgm.
HALLEY	BAS	4108	Issue 2 Plotted the position of Halley's Comet.
MADA	BAS	906	Issue 7 Mental addition pgm works with Drill.Bas.
MADAS	BAS	1174	Issue 8 Mental Add/Sub pgm works with Drill.Bas.
MAKER	BAS	2432	Issue 1 Makes DATA statements the easy way.
MATH	BAS	4992	Issue 4 Math formulae on screen that change.
MGSORT	BAS	5007	Issue 5 Merge/sort files too big for ram memory.
MULT	BAS	1727	Issue 7 Multiplication program works with Drill.Bas.
MUMP	BAS	755	Issue 7 An HEX dump pgm for MS-DOS. Dumps up to 640K.
MURRAY	BAS	5752	Issue 3 Program to improve calculating accuracy.
NETWORK	BAS	10359	Issue 6 A game for up to 4 players. Buy/Sell stock.
NFL86	BAS	6859	Issue 7 An NFL football projection program.
NFLSTAT	BAS	3900	Issue 7 Maintains NFL statistics for NFL86.
NORRIS	BAS	4088	Issues 2&4 A collection of file conversion routines.
PAY	BAS	13568	Issue 4 Payroll program, requires initialization.
PLIST	BAS	2698	Issue 6 A neat page-listing utility program.
PLOT	BAS	5883	Issue 1 Prints histograms with 3 period projection.
PRECOMP	BAS	3320	Issue 4 A BASIC precompiler w/o line #, with labels.
PROG1	BAS	1298	Issue 4 Sequential file tutorial demo program.
PROG2	BAS	893	Issue 4 Sequential file tutorial demo program.
RANDEMO	BAS	3037	Issue 6 A random file tutorial demo program.
RANDEMO2	BAS	2020	Issue 7 A random file tutorial demo program (#2).
SEARCH	BAS	2644	Issue 1 Finds selected variables in target program.
SPELL	BAS	2564	Issue 7 Spelling program works with Drill.Bas.
SUB	BAS	1562	Issue 7 Subtraction program works with Drill.Bas.
UPDATE	BAS	1574	Issue 7 Updates spelling for Drill.Bas.
UXREF	BAS	14464	Issue 5 A BASIC cross reference program.
WOOD	BAS	11135	Issue 3 Best fit program for cutting plywood, etc.
WRITER	BAS	10629	Issue 1 Analyze your writing (for 80 col screens.)

CodeWorks 1st Year Programs for PC/MS-DOS GW-BASIC

The diskette provided contains the following programs. The number after the file name indicates the size of the program in bytes.

Refer to the appropriate issue for programs that require initialization.

Please do not remove our identification lines in the programs, they serve to show where these programs come from and will insure that we stay in business and continue to produce more good and economical software.

ADD	BAS	1573	Issue 7 Addition program works with Drill.Bas.
ADUMP	BAS	1011	Issue 6 An ASCII memory dump program.
BUSMOD	BAS	3328	Issue 5 A business "what if" model.
CAL	BAS	6998	Issue 2 Perpetual (almost) calendar maker program.
CARD	BAS	13280	Issue 2 In memory, mini-database, requires initialization.
CHECK	BAS	6601	Issue 4 Card.Bas adapted to keeping check records.
CONV	BAS	2888	Issue 5 Convert DATA statements to seq. file and back.
DIV	BAS	1549	Issue 7 Division program works with Drill.Bas.
DRILL	BAS	1220	Issue 7 Collection of Instructor's drill programs.
DSTAT	BAS	4768	Issue 6 A descriptive statistics program.
EXTR	BAS	2857	Issue 1 Extracts program lines for later use.
FMAKER	BAS	1558	Issue 7 A general purpose sequential file maker pgm.
HALLEY	BAS	4108	Issue 2 Plotted the position of Halley's Comet.
MADA	BAS	906	Issue 7 Mental addition pgm works with Drill.Bas.
MADAS	BAS	1174	Issue 8 Mental Add/Sub pgm works with Drill.Bas.
MAKER	BAS	2432	Issue 1 Makes DATA statements the easy way.
MATH	BAS	4992	Issue 4 Math formulae on screen that change.
MGSORT	BAS	5007	Issue 5 Merge/sort files too big for ram memory.
MULT	BAS	1727	Issue 7 Multiplication program works with Drill.Bas.
MUMP	BAS	755	Issue 7 An HEX dump pgm for MS-DOS. Dumps up to 640K.
MURRAY	BAS	5752	Issue 3 Program to improve calculating accuracy.
NETWORK	BAS	10359	Issue 6 A game for up to 4 players. Buy/Sell stock.
NFL86	BAS	6859	Issue 7 An NFL football projection program.
NFLSTAT	BAS	3900	Issue 7 Maintains NFL statistics for NFL86.
NORRIS	BAS	4088	Issues 2&4 A collection of file conversion routines.
PAY	BAS	13568	Issue 4 Payroll program, requires initialization.
PLIST	BAS	2698	Issue 6 A neat page-listing utility program.
PLOT	BAS	5883	Issue 1 Prints histograms with 3 period projection.
PRECOMP	BAS	3320	Issue 4 A BASIC precompiler w/o line #, with labels.
PROG1	BAS	1298	Issue 4 Sequential file tutorial demo program.
PROG2	BAS	893	Issue 4 Sequential file tutorial demo program.
RANDEMO	BAS	3037	Issue 6 A random file tutorial demo program.
RANDEMO2	BAS	2020	Issue 7 A random file tutorial demo program (#2).
SEARCH	BAS	2644	Issue 1 Finds selected variables in target program.
SPELL	BAS	2564	Issue 7 Spelling program works with Drill.Bas.
SUB	BAS	1562	Issue 7 Subtraction program works with Drill.Bas.
UPDATE	BAS	1574	Issue 7 Updates spelling for Drill.Bas.
VXREF	BAS	14464	Issue 5 A BASIC cross reference program.
WOOD	BAS	11135	Issue 3 Best fit program for cutting plywood, etc.
WRITER	BAS	10629	Issue 1 Analyze your writing (for 80 col screens.)

CodeWorks

CONTENTS

CODEWORKS

Premier Issue Sep/Oct 1985

Editor/Publisher
Irv Schmidt
Associate Editors
Terry R Dettmann
Greg Sheppard
Jay Marshall
Circulation/Promotion
Robert P Perez
Editorial Advisor
Cameron C Brown
Technical Advisor
Al Mashburn

Produced by 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. Neither is any liability assumed for damages resulting from the use of any information contained herein. Please address 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 please) and allow 4 to 6 weeks for editorial review. Do not send diskettes, rather send a hard copy listing of programs. Media will be returned if return postage is provided. Cartoons and photographs are welcome. 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), one year only. Not available outside United States Zip codes. **VISA and Master Card orders are accepted by mail or telephone.**

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in the United States of America. Bulk rate postage 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 (at no cost).

Point of View

The joy of success - the agony of defeat! Both are wrapped up in programming a computer. To some, it is a disgusting task. Others find in it a sense of accomplishment that is difficult to describe.

To program you must have imagination, be logical, be able to accurately describe a problem, be knowledgeable and most of all, be disciplined. Few things like programming demand the most minute care and attention to what you are doing. Computers are very fussy about how you talk to them. The ubiquitous "syntax error" constantly stares out of the video screen at you. It has other brothers, sisters and cousins that do the same and are less descriptive of who they really are. You get to know them by their effects.

As the age of personal computers progresses, more of us find uses for computing. Although most of the major programs today are available as "load and run": spreadsheets word processors and the like, there exists a whole world of applications waiting to be written by someone. Using canned programs on your computer is fine - they do a great job, but in some ways it is like buying that new car and finding that it can only travel down freeways.

Programming your own has many advantages. First, you learn something and there is always satisfaction in that. Didn't someone once say that knowledge is power? Next is the ability to create a program that fits your problem exactly, not *just about* or *maybe*. It also gives you the confidence and freedom to go into your program later and make improvements, adjustments and updates. If it's your program, you know best how it works and how to fix it if it doesn't. Programming your own also lets you dream up interesting and unique applications previously unheard of.

We believe that there is so much fun and excitement in taking an idea and making it a programming reality that once you get into it, you will too. That is why we are publishing this magazine. We fully intend for it to be an interchange between computerists. To that end, we will bring you an interesting and challenging mix of articles and programs in each issue. Some will be reader-written, so you have a chance to show off your latest brilliance. Each issue will contain an in-depth programming project, complete with explanations of why the program was designed the way it was and how it was developed. The idea of the project is not only to give you a working program that does something useful, but to delve into the whole philosophy of how to get from the *idea* to the working program.

We should also mention that all programs presented in this magazine are yours to do with what you like. You may use them for fun, profit or whatever.

Right now, we are putting together a multi-user UNIX-like system that we intend to use for downloading programs to you. This service is intended to be a part of the subscription price, however, you will need to pay for the phone call. Actually, you learn more by typing the programs into your computer, but if you have a modem and want to download them directly, that's up to you.

This magazine is not intended to be a vehicle for commercial advertising. Nor will it be available on newsstands. It is dedicated exclusively to you. Consequently, we need as many subscribers as we can get to launch this project. We are very excited about it, and hope you enjoy this first issue. Please take a few minutes and fill out the questionaire with the order form. We will be able to serve you much better when we know your equipment and your programming needs. ●

Forum

An Open Forum for Questions & Comments

Obviously, since this is the first issue of this magazine, you are probably asking: "How can you have comments and questions from readers before you even have any?"

It's a neat trick and it's done with mirrors — almost. You see, we needed to have input for this column and so we asked everyone in the office, the proofreaders and even the kid who comes in to play with the spare computer, to ask all the right questions. Anything they wanted to know about the birth of this new baby was fair game. It's not at all surprising that there were no negative comments from that group (actually, there were some, but we sneaked in at night and fixed them.)

Overall though, the questions they asked are very likely some of the same you would ask, so here they are:

Why another computer magazine?
There are magazines and then there are magazines. Most of them cater more to advertisers and their products than they do to the reader. Readers for them are just numbers to hold up their ad rates. We think there is a great deal of plain fun in computing, and so we are directing this publication to the reader.

What is the main focus of the magazine?
Our primary focus is problem solving through programming. To that end, if you have a special (and interesting) programming problem, we will be happy to work on a solution for you and publish it for all to see. We intend to demonstrate the use of programming tools and how to mix and match them to get any job done efficiently and quickly.

Will there always be program listings in the magazine?
Always.

Will the magazine always be 32 pages?
No, we intend to grow slowly (or quickly). Our first goal is to go up to 64 pages in easy steps. When we approach that goal, we'll go after a new one.

Why isn't it slick and four-color?
Slick and four-color is used when you want to impress advertisers and newsstand dealers. We don't care about them. What we do care about is the information we impart. After all, that's the meat of the whole thing, isn't it?

My computer cost me $49.95 and runs on tape, you don't even mention tape.
Get yourself a computer.

Who *can* use your programs?
Disk based machines. Primarily those running MS-DOS, CP/M or TRSDOS. The BASIC language which usually is a part of those machines is very compatible (we'll point out differences as we come to them), and is very likely a product of *Microsoft Inc.* This doesn't mean we will be exclusively on those machines, but they cover a very large portion of the computing realm these days. There are dozens of CP/M machines using MBASIC, probably more yet are IBM PC work-alikes and then there is the whole Tandy crowd. Apples running the Z80 SoftCard use CP/M also, so they are included too.

What about your download system?
We have a XENIX system, Microsoft's version of UNIX, that we have been testing for some time now. It's a multi-user dial-up system, and aside from the operator at the console (which we will have to keep free), we can put two auto-answer modems on it and they can both be in use at the same time doing different things. It isn't ready yet. But it is coming along nicely and we hope to have it fully operational by late fall. Being realists, we are working on an alternate backup that would be even better should ours not come up on schedule. Your subscription price includes the free use of that download system.

Why are you putting your programs into public domain?
To keep people from going to hell for stealing.

Computing is such a dry subject. Why talk about it?
It's not dry if you like it, and if we present it properly it will be loads of fun. ●

Your comments and questions are welcome. We can't guarantee an answer to each and every one, but will surely do our best in giving everyone a fair shake. Don't be afraid to criticize — we have been publishing long enough to be used to it. (Constructive only please!). Address yours to us at ***CodeWorks, 3838 South Warner St., Tacoma, Washington 98409***

Trend Analysis Plotter

For 80 or 132 Column Line Printers

Staff Project. Using a line printer to print graphic data has been with us since the days of Teletype (TTY) input and output. Creating this program was not as difficult as it first appeared. The trick was to take it one step at a time and work down.

The purpose of this project is to create a short utility program that can handle various sets of data and print that data out on a line printer in graph form. To make it more useful, a computation of the trend line should be added along with a three period projection of the data. It would also be nice if it could print on an 80 column or 132 column line printer. In addition, it should show the actual data value somewhere on the graph, along with appropriate labels. To make it handle varying magnitudes of data, it should also have some sort of self-scaling feature.

This sounds like a rather tall order. But, surprisingly, it worked out to be only about 60 lines of code.

The data for this program is handled in DATA statements at the end of the code. This makes it easy to use this program on various sets of different data.

Elsewhere in this issue, we presented a program called "Maker.Bas". That program is an excellent way to create the various data sets that this program can use.

Lines 100 to 270 are preliminary code used to print a neat heading on the screen, clear some string space and give a few instructions.

Lines 280 through 310 ask for information about the number of columns on your particular line printer. They also ask for a name for your graph, which will be printed along with the other information that is output. You get a nicer presentation if you have a 132 column line printer (or if your 80 column line printer can select 132 columns). The 80 column version is still rather presentable however. You can also change all the LPRINT statements to PRINT if you have an 80 column video screen and see your graph on the video instead of the line printer.

Line 320 prints the graph headings. Since the scale must be selected by the program, the headings are simply set to go from 0 through 9 and back to 1. The tabs are computed, based upon the information you gave for 80 or 132 columns. Variable TS is either 5 or 10 depending on your answer to the question in line 280.

Lines 330 through 360 make the printer come along under the heading numbers and add tick marks (using the exclamation mark). If you selected 132 columns, then the tick marks are put every five spaces, otherwise, they are set every 10 spaces. This is because in 80 column mode the marks are closer together, and putting one more mark in between clutters it up. Aside from that, it is difficult to get them exactly half way between. This is handled by TI, which is the step in the FOR..NEXT loop. Note that it is legal to compute the start and end of a loop as in line 340.

Lines 380 through 430 are going to define the scale factor for our graph. In them, we set variable D2 to zero before we enter the loop. D2 will hold the highest data number after we have read the data. We set the J loop to read to 1000. This is just some big arbitrary number, since we will jump out of the loop when we reach the sentinel. Obviously, if you have more than 1000 data elements to read, this number would need to be increased.

In line 400 we read the data label as A$, and then the actual data value, which is variable D. At this point, we immediately check to see if the sentinel has been reached, and if so, jump out of the loop.

In line 410 we check to see if the actual data value is larger than D2, which was previously set to zero, and if so, make D2 equal to D. This way, as we go through the entire list of data, anytime that D is larger than D2, D2 will be set to that value and we will have the largest number in the data set.

Line 420 is where we get the scale factor based on the value of D2. The variable used to hold the scale factor is D1. This line says that if the highest number in the data set was 100 or less, then the scale factor will be 1; less than 1000, scale factor of 10; and so on.

Since we have now once read the data, we need to restore the data pointer. The RESTORE command in line 440 does that for us.

Now that we have the scale factor and restored

the data pointer, it's time to read the data again and print out the graph, one line at a time. Lines 450 through 520 do this. Again, we read the data and in line 490 we make variable L the length of the bar we will print on the graph. Well, not quite. It depends on whether we are in 80 or 132 column mode, doesn't it? The plus 1 in both lines 490 and 500 was necessary to tweek the line to fit exactly under the scale headings on the graph. Line 500 says that if we selected 80 column mode, then we should cut the length of the graph bar in half.

Line 510 prints an entire line on the graph. First, it prints the label, then tabs to TS times 2 to start the row of capital X's for the bar. The for..next loop in line 510 prints a row of those X's starting at the tab position and extending to L, which is the actual data value, scaled to fit the graph with the tweek added to make it come out right. Still in line 510, we then tab way over to the right at TS times 12 plus 2 and print the actual data value. It does this for each data item we have in our list.

Now that this is done, we restore the data again. Yes, we need to read it again to establish the trend line. Up in line 480, we finally got tired of reading to 1000 and actually counted the number of data items and put them into variable C. This could have easily been done earlier, say up after line 400, but we just didn't think of it. Now, in lines 540 through 580 we read the data again and get the sum of the squares and all that for the trend line. Notice that we don't need the label A$ for this, but we have to read it in any case, otherwise we may be reading string data as numeric and that will not work. So we simply read it and ignore it. In line 590 we calculate the values for the regression line.

All this time our printer was sitting on the next line after printing the graph of the data. Now we want to do the same with the three period progression to finish off the graph. This happens in lines 600 through 650. It's very much the same as it was for the data, except now in line 610 we check for negative numbers (you can't print in the negative direction on a line printer). Also, our label for these projected values will simply be PROJ, and instead of using the X as we did earlier, we will use the small letter "o" or anything like that to set it off from the other data. When the graph is done, we will probably want to manually draw a straight line through the last three projection lines to see the actual trend line. The lowercase "o" is ideal for this since it is easy to find the center of this character.

Now that the graph is completed, why not print out the actual values for the regression line? Okay, we already have the values, so in line 680 we print them out, and that's that.

By the way, the only limit to the number of data lines you have is the limit of your computer memory. Using continuous roll paper, you can create a strip chart many feet long if you like. If you do that though, don't forget to set your printer so that the number of lines on the page and the number of lines to print are equal. If you don't, you will have gaps in your charts.

You will find that because of the self-scaling feature of the program, a single large value with many smaller values will not show very good resolution. For example, several values in between 1 and 10 with one that is over 100 will leave those below 100 looking rather compressed.

If you have various sets of data to chart you can save just the data (in ASCII) and merge them into the program starting at line 700. Don't forget to put the proper sentinel at the end of each data set.

There it is, and it does just what we wanted it to. Now you can impress anyone with flashy charts with trend lines and projections. Now let's see... using only a standard line printer, how can we get the computer to print the trend line too? Shall we leave that as an exercise for the reader? We think so. ■

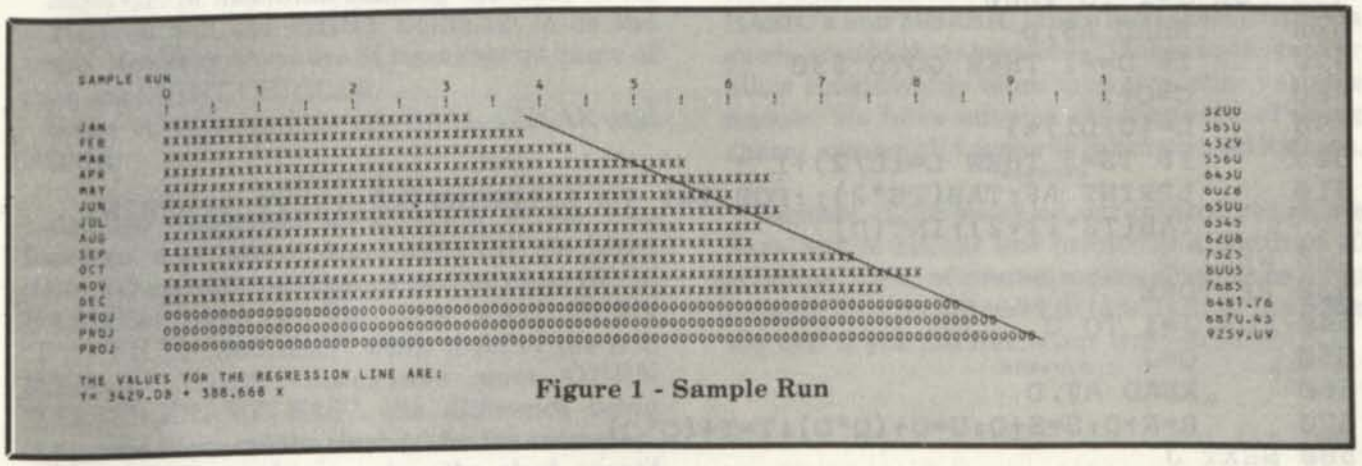


Figure 1 - Sample Run

```
100 REM ** PLOT.BAS  * CREATED FOR CODEWORKS MAGAZINE **
110 PRINT CHR$(12):REM ** CLEAR THE SCREEN **
120 CLEAR 1000
130 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
140 PRINT"          T R E N D  A N A L Y S I S  P L O T T E R"
150 PRINT"                With a three-period projection"
160 PRINT STRING$(60,"-")
170 PRINT
180 PRINT"A utility plotter with 3 period projection for use with"
190 PRINT"an 80 or 132 character line printer."
200 PRINT
210 PRINT"The data for this program must be entered as DATA lines at"
220 PRINT"the end of the program. The form of these DATA statements"
230 PRINT"is: DATA LABEL,#### where LABEL is a name and #### is a"
240 PRINT"numeric value."
250 PRINT"The very last DATA statement should contain a dummy label"
260 PRINT"and a numeric value of -1, like this: DATA ,-1"
270 PRINT
280 INPUT"IS YOUR PRINTER 80 OR 132 COLUMNS - ENTER 80 OR 132";CP
290 IF CP<>80 AND CP<>132 THEN GOTO 280
300 IF CP=80 THEN TS=5 ELSE TS=10
310 INPUT"What will you name this graph";Z$
320 LPRINT Z$:LPRINT TAB(TS*2);"0";TAB(TS*3);"1";TAB(TS*4);"2";
    TAB(TS*5);"3";TAB(TS*6);"4";TAB(TS*7);"5";TAB(TS*8);"6";TAB(TS*9);
    "7";TAB(TS*10);"8";TAB(TS*11);"9";TAB(TS*12);"1"
330 IF TS=5 THEN TI=TS ELSE TI=TS/2
340 FOR I=TS*2 TO TS*12 STEP TI
350     LPRINT TAB(I);"!";
360 NEXT I
370 LPRINT" "
380 D2=0
390 FOR J=1 TO 1000
400     READ A$,D:IF D=-1 THEN GOTO 440
410     IF D>D2 THEN D2=D
420     IF D2<101 THEN D1=1 ELSE IF D2<1001 THEN D1=10 ELSE IF D2<10001
        THEN D1=100 ELSE IF D2<100001  THEN D1=1000 ELSE IF D2<1E+06
        THEN D1=10000
430 NEXT J
440 RESTORE
450 FOR J=1 TO 1000
460     READ A$,D
470     IF D=-1 THEN GOTO 530
480     C=C+1
490     L=(D/D1)+1
500     IF TS=5 THEN L=(L/2)+1
510     LPRINT A$;TAB(TS*2);:FOR I=1 TO L:LPRINT"X";:NEXT I:LPRINT
        TAB(TS*12+2);INT(D)
520 NEXT J
530 RESTORE
540 FOR J=1 TO C
550     Q=J
560     READ A$,D
570     R=R+D:S=S+Q:U=U+(Q*D):T=T+(Q*Q)
580 NEXT J
```

```
590 Y=(T*R-S*U)/(C*T-S*S):X=(C*U-S*R)/(C*T-S*S)
600 FOR F=C+1 TO C+3
610     F1=Y+X*F:IF F1<=0 THEN PRINT"CAN'T PRINT NEG. #'S":GOTO 690
620     L=(F1/D1)+1
630     IF TS=5 THEN L=(L/2)+1
640     LPRINT"PROJ";TAB(TS*2);:FOR I=1 TO L:LPRINT"o";:NEXT I:LPRINT
        TAB(TS*12+2);INT(F1)
650 NEXT F
660 LPRINT" "
670 LPRINT"THE VALUES FOR THE REGRESSION LINE ARE:"
680 LPRINT "Y=";Y;"+";X;"X"
690 END
700 DATA JAN,3200
710 DATA FEB,3850
720 DATA MAR,4329
730 DATA APR,5560
740 DATA MAY,6430
750 DATA JUN,6028
760 DATA JUL,6500
770 DATA AUG,6345
780 DATA SEP,6208
790 DATA OCT,7325
800 DATA NOV,8005
810 DATA DEC,7685
820 DATA ,-1
```

"It's for you."

Programming Notes

Our first programming note must of necessity concern itself with our listings and some of the common differences in the various implementations of *Microsoft* BASIC.

The first we usually run into is the CLEAR SCREEN command. Most MS-DOS BASIC's and just about all the Tandy BASIC's will use CLS to clear the screen and home the cursor. We've heard they are trying to make this the ANSI standard.

Most CP/M machines running MBASIC or its variations will use PRINT CHR$(12) to do the same. However, there are at least two we know of that use PRINT CHR$(26).

Some other variations are CALL CLEAR and HOME.

****

Another difference in BASIC's is the way filespecs are named. Most MS-DOS machines (IBM-PC and work-alikes) as well as most CP/M BASIC's, open a file this way: OPEN "I",1,"FILENAME.BAS" while most of the non MS-DOS Tandy machines use: OPEN "I",1,"FILENAME/BAS". the difference being the period (dot) and the slash (/) for the extension. Those Tandy machines using the slash would recognize the dot after the name as a three letter password for the file and not as an extension. The example above, by the way, was a command to open a sequential file for input and assigns buffer number 1 to it.

****

Spacing around keywords in a BASIC program doesn't make any difference on some machines. In general, those same machines will also only allow two-letter variable designators. Most MS-DOS BASIC's and MBASIC (and a few others) require a space around their keywords. This is because they allow considerably more than two-letter variable names. We have adopted the standard of using spaces around all keywords in our printed listings.

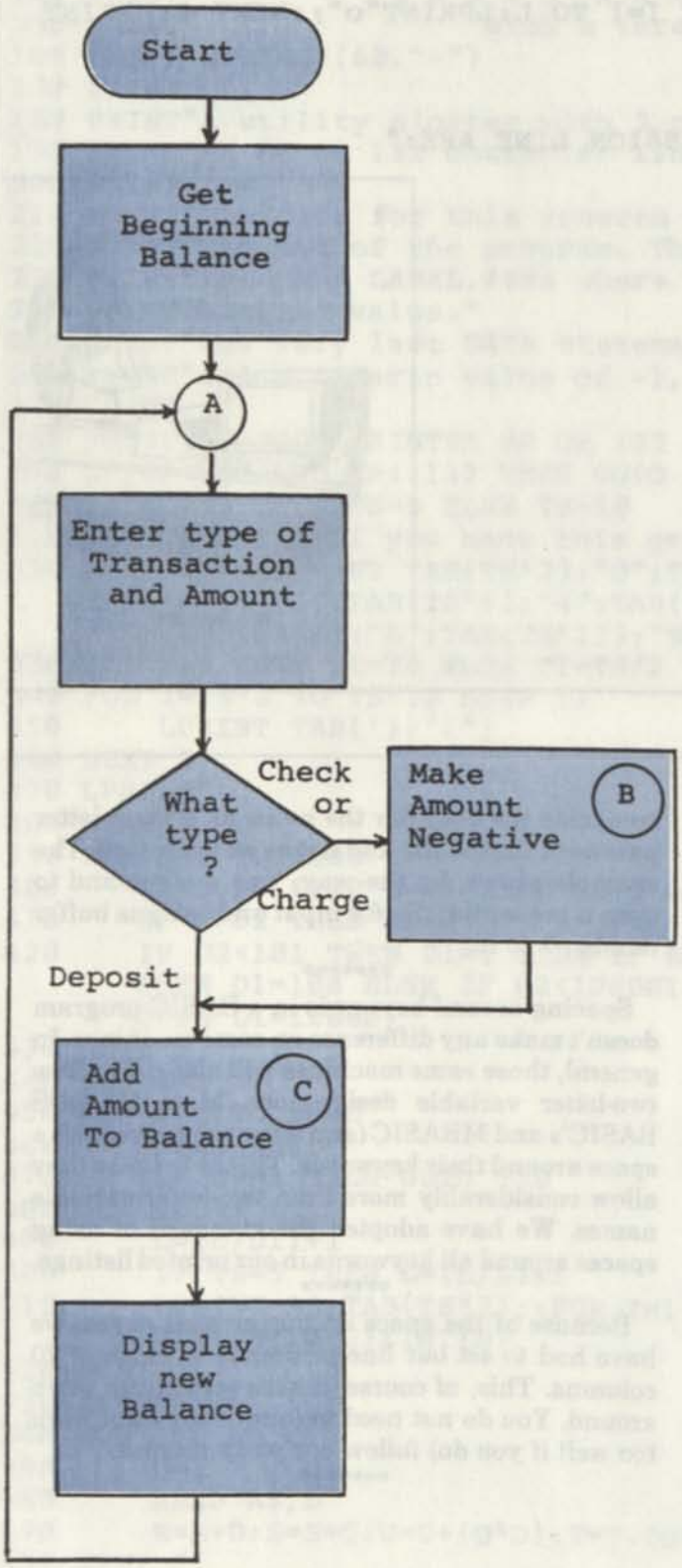
****

Because of the space on our printed pages, we have had to set our line printer to a width of 70 columns. This, of course, makes some lines wrap around. You do not need to (and it may not work too well if you do) follow our wrap around.

****

Programming made Easy

A Program to balance your Checkbook



Terry Dettmann, Associate Editor. This program is intended to show that it only takes four BASIC commands to write a program. They also happen to be the four most used commands. We've all seen enough checkbook balancing programs, so that's no big deal, but that is not the point of the program.

```
10 PRINT "ENTER THE BEGINNING
   BALANCE"
20 INPUT BA
30 PRINT"ENTER (CK) CHECK, (DP)
   DEPOSIT OR (CH) CHARGE"
40 INPUT C$
50 PRINT"ENTER AMOUNT"
60 INPUT AM
70 IF C$="CK" THEN GOTO 120
80 IF C$="DP" THEN GOTO 130
90 IF C$="CH" THEN GOTO 120
100 PRINT"ERROR IN DATA ENTRY —
    TRY AGAIN"
110 GOTO 30
120 AM =—AM
130 BA = BA+AM
140 PRINT"AMOUNT", AM,
    "NEW BALANCE", BA
150 GOTO 30
```

In programming, we often lose sight of the forest for the trees. We have a tendency to confuse good programming with very specialized techniques. We forget that it is more than an ability to program random files, faster sorts, or tricky input routines. Good programming is the development of simple solutions to a given problem.

In reading magazines, many beginners will try a new technique before they have mastered the simpler tasks. Even the seasoned professional will find he can accomplish a great deal with the simplest BASIC commands and statements.

The four BASIC statements we will use in this article are: PRINT, INPUT, IF-THEN and GOTO. By learning how to use these statements effectively we can expand our programming skills

a great deal. By adding one or two statements at a time, we can come up with some amazing capabilities.

INPUT: We will use the INPUT statement in its very elementary form. At this point we will not allow it to do anything but enter information (data). The form of the statement is:

INPUT *variable-name*

where the variable name can be either an alphanumeric string or a number. Strings are defined by using the dollar sign ($) after the variable name. Those variables without the string declarator are numeric variables.

PRINT: The PRINT statement is used to get information from the computer. It prints data to the video screen. The form we will use is:

PRINT *variable-name list*

where the variable name list can be one or more variable names consisting of numeric or string variables or constants. The only variable name list separator we will use in this article is the comma.

IF-THEN: The IF-THEN statement is used to make decisions. The form we will use is:

IF*(condition to be tested is true)* THEN *(do this)*

GOTO: The GOTO statement is used to jump or branch to a new place in the program. We will use it with the IF-THEN statment in complex situations. Its form is:

GOTO *line number*

where the line number is the number of the program line to be jumped or branched to.

Program Design: The first step in any program is the design state. Unfortunately, many BASIC programmers just sit down at the keyboard and make up little programs as they desire. For serious programming, nothing beats prior planning.

The standard method for design is the flow chart. It is a series of symbols, connected by lines to show the flow of the program. In its greatest detail, every minute step of the program is illustrated. In its simplest form, it isn't much more than a block diagram of the program concepts. There are arguments for both ends of the spectrum of the flow chart. We will show you a variation which is gaining wide-spread acceptance among computerists.

This variation uses English intermixed with BASIC statements and commands. As the program is prepared and refined, more and more of the English text is converted into BASIC. For foreign programmers, a foreign language text could just as well be used, even though the BASIC portion is still in English.

Before we get into our sample program, there is one more thing to understand about BASIC. We talk about statements and commands. These are very much like English sentences. In BASIC each sentence must be assigned a line number. The computer will numerically follow these line numbers unless instructed to do otherwise by the program itself.

Now let's design our program.

The first step in any program design is to state or define the problem we wish to solve, which is widely referred to as the program objective. Many programs have been lost by ignoring this step. If the problem or objective is inadequately defined, the program will almost always require redesign.

For our problem, we state:

```
  PRINT prompt for the beginning balance.
  INPUT the beginning balance.
A PRINT prompt for check, deposit or charge.
  INPUT transaction type.
  PRINT prompt for amount.
  INPUT amount.
  IF check then goto B.
  IF deposit then goto C.
  IF charge then goto B.
  PRINT error message.
  GOTO A.
B Make amount negative.
C Balance = balance + amount.
  PRINT amount, new balance.
  GOTO A
```

Next we assign variable names which are similar to our English names in the program flow: BA for balance and AM for amount. See how closely our finished program matches the flow. ■

Search

A Program to find Specific Variables

Randolph Townsend, Riverside, California. Save your eyes and your temper when your program goes awry. This is a short but very useful program that not only will find specific variables, but will find any search string. This means that you can search through text files as well as program listings. This, and other programs in this issue, rely on some knowledge of what ASCII is all about. We have put together a short discussion of ASCII on page 30, and suggest you look at it if you must ASCII.

Although the messages are often cryptic, the error routines on many computers help enormously in BASIC programming. All too often however, the reason for the error is not apparent, or even worse, the computer finds no error but the output is wrong. Problems of this sort may have many different sources. But they often have their roots in fuzzy thinking, inattention or even stupidity on the part of the programmer. Most of us have had the experience, when confronted with one of these knotty dilemmas, of spending hours reading each line of a long program in an attempt to track the problem to its lair. Catastrophic consequences have resulted only because we have used the same variable for different things.

It is much easier to let the computer do the reading. This can be arranged in a variety of ways. Many software firms will be delighted to sell you programs that will search out every occurrence of a given string in your program. Rapid and effective assembly language programs to accomplish this have been published. However, it is quite easy to devise a BASIC program which searches through your program for any string you specify and have it tell you where it was found. The only shortcomings to working in BASIC, rather than in assembly language, are slightly longer times in accomplishing the search and the need to store your target program in ASCII format first.

One simple program which does this chore quite well is shown in the listing accompanying this article. Operation of the program is very easy. The problem (target) program is stored on disk in ASCII format (SAVE"filename",A). The search program shown in the listing is then run. The name of the program being searched and the string to be searched for, are entered in response to the queries. The program being searched is read in with the LINE INPUT# command, examined using the INSTR command, and the string is printed when it is found. You have options in line 300 of continuing the search, printing out the line on paper, or stopping. These options are denoted with the commands C, P, and S, respectively. The printout also gives you the name of the program and the string for which you have searched.

When all the lines have been read, you can re-examine for another string, look at another program or quit. As a check, run this program on itself and you should find PN$ to be present in lines 210, 220, 240 and 370.

Don't forget to save your target program in ASCII and enter BASIC with at least one file allocated. Also, various computers use different syntax in the OPEN statement in line 240. Adjust accordingly for this command and for the INSTR and LINE INPUT# commands.

This program, although designed primarily for BASIC program listings, will work well on any ASCII text file. ●

```
100 REM ** SEARCH.BAS *
110 CLEAR 1000
120 PRINT CHR$(12) :REM ** CLEAR SCREEN COMMAND **
130 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
140 PRINT"                    S T R I N G  S E A R C H"
150 PRINT"                Search for Strings in other Programs"
160 PRINT STRING$(60,"-")
170 PRINT
180 PRINT" Program to search for strings in other programs."
190 PRINT" The Target program must be stored in ASCII format."
200 PRINT
210 INPUT "ENTER TARGET PROGRAM NAME ";PN$
220 IF PN$ = "END" THEN END
230 INPUT "ENTER STRING TO BE SEARCHED FOR ";SS$
240 OPEN "I",1,PN$
250 IF EOF(1) THEN GOTO 330
260 LINE INPUT #1,TX$
270 IF INSTR(TX$,SS$)=0 THEN GOTO 250
280 PRINT TAB(10) SS$;" FOUND IN "
290 PRINT TX$
300 PRINT "Continue/Stop/Print, C/S/P ";:GOSUB 430: PRINT Y$
310 PRINT " "
320 IF Y$ = "C" THEN GOTO 250 ELSE IF Y$ = "P" THEN GOSUB 360:GOTO 250
330 CLOSE 1
340 PRINT "FINISHED WITH PROGRAM":PRINT" ANOTHER PROGRAM Y/N ";
   :GOSUB 430
350 IF Y$ = "Y" THEN GOTO 110 ELSE IF Y$ = "N" THEN END
360 IF FL=0 THEN GOTO 370 ELSE GOTO 380
370 LPRINT "SEARCH OF PROGRAM ";PN$;" FOR ";SS$
380 LPRINT
390 LPRINT TX$
400 LPRINT
410 FL = 1
420 RETURN
430 Y$ = INKEY$:IF Y$ = "" THEN GOTO 430 ELSE RETURN
```

Programming Notes

The relational operators AND and OR sometimes seem to play tricks on us. Let's say we want the user to enter either the number 10 or 20. If a number other than 10 or 20 is entered, we want to go back and ask the question again, otherwise, we should go on to the next normal line of code. For example:

10 INPUT"ENTER THE NUMBER 10 OR 20";A
20 IF A<>10 OR A<>20 THEN GOTO 10
30 REM Program lines follow here.

It seems logical to say that if A does not equal 10 or 20 to go back and ask the question again. However, this section of code will go back to line 10 in *every* case. You could write line 20 to say: 20 IF A = 10 OR A = 20 THEN GOTO 30 ELSE GOTO 20. That works. But there is something strange about asking the program to go to a line it would go to anyway. Another way to do it is to change the OR in line 20 to an AND. This seems to defy logic but it works. Those of you familiar with Boolean logic will recognize it. When changing equalities to inequalities, we need to change the relational operator. In this case from OR to AND. Now if we say:

20 IF A <> 10 AND A <> 20 THEN GOTO 10

it works just fine.

****

Extract

A program to Extract lines of code

Dexter Walker, Birmingham, Alabama. Unlike the previous program, "Search", this program extracts sets of lines out of existing code and allows you to accumulate them for automatic insertion into another program. Very handy if you are a subroutine builder. You may never have to write code again if you extract, then mix and match sections from other programs. Again, this one relies on some knowledge of ASCII, see page 30. Also along these lines you should find use for R C Bahn's "Strings and ASCII" following this article. Both this, and the previous programs, should be valuable additions to your utility library.

Everyone who has done any programming has a treasury of good routines buried in his programs — good logic that can be used later in another program. After all, there really is no use in re-inventing the wheel. We all know that we can load up that old program, delete everything before and everything after the nugget of code that we want and merge it into our new masterpiece. But, if you are like me, you probably don't do it.

I recently did this on a rather long, previously written program and even though it worked, I didn't enjoy it very much. I finally figured out that there was something destructive about killing a part of the good code which I had written, even though the end result was going to be good. It then occured to me that it would be better to extract the lines I wanted from the old program rather than deleting the code I didn't want.

The Extract program shown here will run on most disk-based machines with only slight, or no, modification. Be sure to enter BASIC with 2 files open.

The source (target) file must be saved in ASCII format (SAVE"filename",A) and the file created by this program will also automatically be in ASCII format, ready to be merged with whatever else you are writing.

The extracted file will allow you to pick up to ten different sections from your source file for extraction. If you need to extract more than ten sections, you will need to dimension the array SL(J) to the number you want. That dimension command does not exist in the program now, but can be added as line 115, e.g., 115 DIM SL(20). The reason it is not now dimensioned is that most BASIC interpreters allow up to ten items in any array without dimensioning.

This is a useful routine that will take just a few minutes to key in and will save you hours of recoding. You might start a file of useful utility programs like this one and add this one to it.

Now, especially if you are not a touch typist, you can extract and merge rather than delete and type in sections of code from other programs. ●

```
100 REM ** EXTR.BAS * BY DEXTER WALKER **
110 CLEAR 5000
120 PRINT CHR$(12): REM ** CLEAR THE SCREEN **
130 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
140 PRINT"                          E X T R A C T"
150 PRINT"              Extract and file portions of a program"
160 PRINT STRING$(60,"-")
170 PRINT"This program extracts portions of an existing file"
180 PRINT"and creates a new file containing the selected lines."
190 PRINT"The source file must be saved in ASCII format."
200 PRINT
210 LINE INPUT "Enter the name of the SOURCE file :";FS$
220 LINE INPUT "Enter the name of the EXTRACT file :";FD$
230 PRINT CHR$(12): REM ** CLEAR THE SCREEN **
240 PRINT "Enter line numbers to extract (enter E to end)"
250 PRINT
260 J = 1
270 INPUT "Starting line: ";SL$
280 IF SL$ = "E" THEN GOTO 330 ELSE SL(J) = VAL(SL$)
290 INPUT "Last line (may be same as start line) ";EL(J)
300 J = J+1
310 PRINT
320 GOTO 270
330 JT = J-1
340 PRINT CHR$(12): REM ** CLEAR THE SCREEN **
350 PRINT "DISPLAY OF EXTRACTED LINES"
360 PRINT
370 OPEN "I",1,FS$
380 OPEN "O",2,FD$
390 IF EOF(1) THEN GOTO 450 ELSE LINE INPUT # 1,A$
400 V = VAL(A$)
410 FOR J = 1 TO JT
420 IF V >=SL(J) AND V <=EL(J) THEN PRINT A$:PRINT #2,A$
430 NEXT J
440 GOTO 390
450 CLOSE
460 PRINT
470 PRINT "FILE EXTRACT COMPLETE"
```

Programming Notes

Many BASIC's will have a command called MOD. It is used like this: PRINT 7 MOD 3 (or A MOD B) and the machine will return the remainder left when dividing 7 by 3 (or B into A). This is called the Modulus Operator, and the remainder is called the Modulo value. For those of you who do not have this operator, you can simulate it with this: X = INT(A—B*INT(A/B)).

******

Interpreter BASIC is interactive BASIC. Unlike compiled BASIC, where you cannot try any section of code until it is all compiled, interactive BASIC lets you do calculations at the READY or OK prompt or run small sections of code and print the results to see how it worked. This has to be one of the greatest advantages of interactive BASIC, since it lets you test and debug a program in no time at all compared to compiled BASIC. Of course, compiled BASIC runs at a much higher speed and once compiled cannot easily be changed or decoded without having the source code. (Source code is the code from which the compiled version comes.)

*****

Strings and ASCII

An Exercise in Strings and ASCII Codes

String constants or variables are single or groups of alphabetic, numeric or graphics characters. String variables are designated in BASIC by the appearance of the dollar sign as the last character of the name, such as A$.

The object of the following exercises is to learn to manipulate "strings" and the ASCII code for simple computer graphics. The major BASIC statements which are concerned with strings include ASC(string), CHR$(code exp.), LEN(string), LEFT$(string, n), MID$(string, p,n), RIGHT$(string, n), STR$(numeric exp.), VAL(string) and STRING$(n, char.). A single operation, concatenation, can be performed on strings. In this operation, a string may be appended to another string by use of the plus (+) sign. Complete descriptions of the foregoing statements can be found in the reference manual for your computer.

The first part of this article concerns methods for building strings and examining their appearance prior to utilization. The last part of the article describes a demonstration program for the tabular and graphic display of the sine function.

Counting in a loop from 32 to 191: Type and run the following program:

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 32
30 PRINT 31 + (32*(I-1))+J;
40 NEXT J
50 PRINT : PRINT
60 NEXT I
70 END
```

Later we will want to automatically build some strings and have the computer do the counting. This nested loop performs the task. The outer I loop (lines 10 and 60) directs program flow through the inner J loop five times. The inner J loop (lines 20 and 40) counts each of the five times from one to 32. The sequential numbers (32 to 191) are computed and printed in line 30. To assure yourself that the statement operates correctly, compute by hand the value for I = 1, J = 1; I = 5, J = 32 and several intermediate numbers. The range of numbers from 32 to 191 corresponds to the ASCII codes for characters and graphics symbols. Line 50 separates the video screen output into five blocks of 32 numbers. The first PRINT completes the line. The second PRINT skips a line. Note that if the terminal number in a block occupies the last position of the video line, an automatic line feed occurs. Thus, the number of blank lines separating blocks may not be constant for all blocks.

Building strings and testing the video screen and printer: Type and run the following program:

```
100 CLEAR 100
110 B$ = "ASCII CODES"
120 FOR I = 1 TO 5
130 A$ = ""
140 FOR J = 1 TO 32
150 N = 31 + 32*(I-1)+J
160 A$ = A$ + CHR$(N)
170 NEXT J
180 PRINT B$; N-31;"TO";N
190 PRINT A$
200 LPRINT B$;N-31;"TO";N
210 LPRINT A$
220 NEXT I
230 END
```

This program displays all the ASCII characters on the video screen and printer. If you have no printer delete lines 200 and 210. You can identify the counting loop consisting of lines 120, 140, 170 and 220. The calculation of the sequence number (N) occurs in line 150. A string of 32 characters (A$) is built by concatenation in line 160. A$ is used in the output in lines 190 and 210. Another string (B$) is defined in line 110 and used as a label in lines 180 and 200.

This program shows you all the characters available for building strings. The options, particularly for ASCII codes greater than 95, will vary depending upon your type of computer, the presence of lower case capability and your printer. Many printers will not be able to interpret the graphics codes (128 through 191). Subsequent programs in this article should be modified to conform to your own system's capabilities.

Remember that the graphics codes for some computers only extend from 127 to 159. Change line 310 accordingly.

Organized inspection of groups of characters:

```
300 CLEAR 1000
310 FOR K = 32 TO 191
320 FOR N = 1 TO 23 STEP 2
330 IF N = 1 THEN
      PRINT "ASCII CODE = ";K
340 A$ = STRING$(N,CHR$(K))
```

```
350 PRINT A$
360 NEXT N
370 FOR T = 1 TO 200:NEXT T
380 NEXT K
390 END
```

This program demonstrates the STRING$ statement and, in an organized fashion, allows you to inspect the appearance of the characters in lines and in sheets. While there will be few surprises with the alphabetical, numeric and special characters, the graphics characters will provide interesting patterns. The exact pattern will depend upon the model of your computer.

Lines 310 and 380 define a loop running from 32 to 191. You might want to confine this study to the graphics range of 128 to 191. Lines 320 and 360 define a loop which will count twelve odd numbers between and including one and 23. This number is used in line 340 to produce twelve lengthening representations of A$, each of which is printed on the video by the statements of line 350.

Line 330 prints a label for the video screen page. Line 340 forms A$. The STRING$ statement automatically concatenates a string of characters of length N composed of characters defined by CHR$(K).

Line 370 is a timing loop which will enable you to quickly inspect the page. Depending upon the speed of the clock in your computer, you may want to change the duration of the loop by replacing the 200 to 2000 or more. If the pattern interests you, press BREAK and look at it. Record the ASCII code for future reference. Type CONT to continue. Note that when you stop the program sixteen lines of the screen are occupied by the display and systems messages. The label does not scroll off the screen.

Tabulating and plotting a function: Listed with this article is a program which uses the prior concepts to tabulate and plot the sine function (SIN(T)). To accomplish these objectives four primary tasks must be performed: (1) initialization of the program, (2) computation of data, (3) tabulation of the data, and (4) graphing the data.

In most numerical graphing problems the range of the numbers to be plotted will not conform to the dimensions of your video screen or printer. Thus, the computed values must be scaled in the direction of both X and Y axes.

For programming ease, we have chosen to display the X axis in the vertical direction and the Y axis in the horizontal direction. The STRING$ statement will be used to build a string of an appropriate length in the horizontal direction of the Y axis (line 380). The number of intervals in the designated range of X will determine the angular

```
  X             Y
 -3            -.14112
 -2.5          -.598472
 -2            -.909298
 -1.5          -.997495
 -1            -.841471
 -.5           -.479425
 0              0
  .5            .479426
  1             .841471
  1.5           .997495
  2             .909298
  2.5           .598472
  3             .14112
```

```
RANGE OF X=-3 TO 3
RANGE OF Y=-.997495 TO .997495
```

Figure 1 Sample output of listing 1

scaling (lines 60, 130 and 140). Each interval will occupy one line of output.

Note that scaling in both X and Y directions demands knowledge of the maximum and minimum values of X and Y. The range of X is defined in line 90. The range of Y is found in the course of computing values in lines 120, 190 and 200.

Remember that the sine function utilizes angles in radians instead of degrees. The input in line 90 for the sine function should therefore be in radians. Try the folowing limits in line 90: 0,3.14159; -3.14159, 3.14159; 0, 6.28318. The formula for conversion of degrees to radians is:

Radians = (3.14159) * (degrees/180)

The above limits in degrees are 0, 180; -180, 180; and 0, 360.

Discussion. The function plotting program illustrates the fundamental problems of scaling and use of ASCII characters. It is relatively short and serviceable. The next logical improvement would be the introduction of labeled axes.

The width of the video screen graph was limited to 32 spaces to accommodate computers with that

size screen. The variable WS can be redefined in line 350 for wider screens. Similarily the variable WP can be redefined in line 350 for wider printers. The precision of printer plots can be further increased by increasing the number of lines (NL) in line 60.

Note that the program flow in line 460 returns to line 50 and avoids the CLEAR statement in line 30. Thus, after the first pass, the program will repeat by merely pressing RETURN.

Finally, returning to strings, you should experiment with the graphics "fill" STRING$(DI,CHR$(132)) and the graphics symbol (CHR$(157)) of line 380. The numbers 132 and 157 were chosen arbitrarily. The previous exercises probably have generated different choices for you. For bar graphs you may want these two symbols to be identical. Note that in line 390 an alternate, safe and quick way of plotting without "fill" is the use of the TAB statement. ■

Listing 1

```
10 REM ** DEMO PLOTTING ROUTINE **
20 REM ** INITIALIZE **
30 CLEAR 1000
40 DIM Y(15), X(15)
50 PRINT CHR$(12): REM ** CLEAR THE SCREEN **
60 NL = 12
70 INPUT "PRINTER OUTPUT? (Y/N)";P$
80 REM ** COMPUTE RESULTS WITHIN DESIGNATED RANGE **
90 INPUT "ENTER MIN, MAX VALUES OF X"; DN,UP
100 RN = UP - DN
110 N = 0
120 BG = -99999 : SM = 99999
130 ST = RN/NL
140 FOR T = DN TO UP STEP ST
150 N = N + 1
160 X(N) = T
170 REM ** FUNCTION IS DEFINED IN NEXT STATEMENT **
180 Y(N) = SIN(T)
190 IF Y(N) > BG THEN BG = Y(N)
200 IF Y(N) < SM THEN SM = Y(N)
210 NEXT T
220 YR = BG - SM
230 REM ** DISPLAY NUMERICAL RESULTS **
240 PRINT CHR$(12): REM ** CLEAR THE SCREEN **
250 PRINT "   X", "   Y"
260 IF P$ = "Y" THEN LPRINT"  X","  Y"
270 FOR K = 1 TO N
280 PRINT X(K), Y(K)
290 IF P$ = "Y" THEN LPRINT X(K), Y(K)
300 NEXT K
310 INPUT "PRESS RETURN TO CONTINUE";A$
320 REM ** GRAPH RESULTS **
330 PRINT CHR$(12): REM ** CLEAR THE SCREEN AGAIN **
340 FOR NN = 1 TO N
350 WS = 31: WP = 31
360 DI = (WS -1)*(Y(NN) - SM)/YR
370 PD = (WP -1)*(Y(NN) - SM)/YR
380 PRINT STRING$(DI,CHR$(132));CHR$(157)
390 IF P$ = "Y" THEN LPRINT TAB(PD)"*"
400 NEXT NN
410 PRINT "RANGE OF X="; DN; "TO"; UP
420 IF P$ = "Y" THEN LPRINT "RANGE OF X=";DN; "TO"; UP
430 PRINT "RANGE OF Y="; SM; "TO"; BG
440 IF P$ = "Y" THEN LPRINT "RANGE OF Y="; SM; "TO"; BG
450 INPUT "(N)EW PLOT OR (E)ND"; A$
460 IF A$ = "N" THEN GOTO 50
```

Writer

Evaluates your Writing Mechanics

Staff Project. This is the "biggie" for this issue. It may seem rather complex, but remember that it is just a bunch of short routines all hooked together to do the job. Naturally, it will be available on the download when it becomes operative. If you type it in, pay special attention to lines 1850 and 1860. Don't make the lines wrap where we did. We had to because we are limited to 70 characters on the printed page. Those lines will fit nicely on an 80 column screen.

Writer is a program that takes a critical look at the mechanics of your writing. It knows nothing about the context of your writing, but can look at sentence length, word length, use of punctuation, unique word usage and other quantifyable attributes. With it, you can check your own or anyone else's writing mechanics.

Although appearing to be slightly verbose the program flows in a somewhat top-down structure. Text to be analyzed may be entered directly from the keyboard or from a previously saved disk file.

To simplify the code, there are several "do not's" which are listed on the screen when entering text. Perhaps the most important of these is not to use periods in abbreviations, since the period followed by a space is what the program looks for to determine the end of a sentence.

Since most authorities agree that samples of less than 100 words are insufficient to make any intelligent test, the program asks for more than 100 and less than 300 words to be examined. It turns out that most screens will hold somewhere between 100 and 300 words, so it works out nicely and one can see the entire sample text on one screen.

After entering text or loading a file from disk, the program counts sentences and words. This actually occurs as the file is being loaded and happens in lines 710 through 780. It then tells you how many words and sentences, and that it will take approximately two minutes to analyze the text. This time is rather approximate and depends upon the speed of your particular computer. During this time, it also counts the occurance of the various punctuation marks, which will later be removed because of the need to get accurate word length counts.

After counting punctuation marks we want to remove them and at the same time change upper case to lower case. This happens in lines 850 through 980. We removed the punctuation because later we will want to compare words, and obviously, the word "end!" will not compare with the word "end". We remove the capitalization for the same reason. The little routine from line 930 through 970 does the trick of removing capital letters. It simply checks each letter to see if it is a capital, and then if it is, it adds 32 to its ASCII value and puts it back. Years ago when the ASCII standard was originated, someone was apparently thinking, and made lower case letters exactly 32 higher than their upper case counterparts.

Now that the words are all stripped of their punctuation and capitalization we want to sort them into alphabetical order. One reason for doing this is that it will make it much easier to find the frequency of repetitious words later. The sort is a modified Shell-Metzner sort, and in line 1030, we print a plus sign on the screen after each pass through the list. This is mostly to let you know that something is happening and that your machine has not hung up on itself.

For those of you who have the SWAP command in your BASIC, you can change line 1100 to read: SWAP P(I),P(L). It may speed up the sort somewhat.

Most text evaluation schemes depend on the use of small, medium and large words. They determine that by the number of syllables in the word. But, a

computer doesn't know doodly-squat about syllables, so we had to come up with a close approximation in lines 1170 through 1270.

The problem came down to words with three or more syllables. Less than three syllable words are considered "easy" words, and getting them was easy as well. Most nine character words have three syllables, but some do not. At the same time many eight character words have only two syllables, but some do not. By checking a few thousand words from a standard dictionary, we found that about 80% of nine character words have three syllables and that about 45% of eight character words have three syllables. This is accounted for in line 1240, after we have found the number of eight, nine and greater character words.

In lines 1250 through 1280 we run through our sorted list of words to find how many unique words were used. Since the list is sorted, all the words "the", for example, will appear together. Our loop then simply runs down the list and compares each word to the next one in the list. If it is the same, nothing happens, but if it is different, then variable VC is incremented by one. We already know (from much earlier in the program) how many words there were. Now we know how many were unique and what the percentage of unique words is.

The next thing we want to do is to find out how many times each word was used. To do this, we build an array, using W$(U,#) where # can be 1, 2 or 3. W$(U,1) will hold the word, W$(U,2) will tell how many times it was used and W$(U,3) will temporarily be stuffed with the number 50. Let's digress a bit and then find out more about that number 50.

One of the objectives of this program is to compare our writing mechanics against an average of some 50 well-known authors. Note we didn't say "good" authors, as we are not in a position to pass judgement on what is good or bad. So we stick to well-known. Aside from that, we can only look at how they say it, rather than *what* they say. There is a difference. We looked at Defoe, Cooper, Harte, Twain, O Henry, Hemmingway and lots of others to get average figures for this comparison. It turned out that there were some common words used by all of them whose frequency was surprising. You can see them listed in line 2020. The words "the", "of" and "and" lead the list in order of frequency in most writing. This does not mean that your writing must conform to this curve, it's an average after all. It just makes for an interesting comparison.

The program will display a graph showing these common words from our famous authors. Then we want to see how we stack up against it. So the problem is how to get our words out of the sorted list in the same order the author graph shows. We are going to do it from lines 1420 through 1490. Here, we read our data statement in line 2020, then with an inner loop we go through our sorted list and find where our corresponding word is. Back in line 1370 we stuffed the number 50 into W$(U,3). Now as we go through our sorted list, we can put the loop counter number into W$(U,3) in place of the number 50 whenever we find correspondence. The very next section of code will sort again, this time on W$(U,3) and we will have our words which correspond to the author's in the proper order and all the rest will have remained as the number 50 and will be sorted at the end of the list. Line 1590 just says to go back through the sort until there are no more switches (which means that the entire list is sorted.)

Lines 1600 through 1630 take all of our values and adjust them to the proper percentages and the like so we can print them out in the next section of code. The Fog Index is figured here also, and is variable FI.

Some people do not put too much stock into the Fog Index. Frankly we don't either. We found a much more telling indicator of interesting writing in the unique word percentage. Most of the well-known authors works had unique word percentages of at least 60% and above. It is an excellent indicator of writing vocabulary. Less experienced authors generally come in at 50% and below on unique words, which means they use fewer words and use them repeatedly. Clearly, the greater diversity of words tends to make for more interesting reading.

Just for illustration: In one of O Henry's short works he takes a generous swipe at the Associated Press. He does it in 234 words and just four sentences, one of which is almost 150 words long. He used 65% unique words and his Fog Index was a grade level 26! This program critiqued that writing and offered the suggestion to shorten sentences to get the Fog Index down. We are glad he didn't, it would have ruined a priceless piece of writing. Don't give up on the program however, it can help a lot when writing directives, manuals or any other educational material.

Meanwhile, back in the code, we are preparing to display a graph of the well-known writer's common word usage. Lines 1850 and 1860 are re-usable headers. Notice that F1$ creates a rather non-linear scale, the first two points being a factor of 10, while the rest of the marks represent a

change of only one-half.

We read the data statement in line 2020 again, and after doing a little scale fitting in lines 1940 and 1950, we print the graph on the screen. After that, we clear the screen and see how our words compared.

Next, we use the same graph headings again and look at all the words we used three or more times. If there are more than a screenfull, the program tells you there are more. If there are none at all it tells you that apparently you used no other words three or more times. This is a great place to see those little unconscious quirks we all have when we repeat the same word endlessly without even knowing we do it.

The option to repeat the result sequences is given in 2370, and you can step through them again without having to wait for all that earlier sorting and computation. When you are satisfied with your results, we borrow a line from Joan Rivers and ask if we can talk. If you answer no, then it's all over. Otherwise, the program asks some questions and based on the data it has, makes some suggestions about your writing. You can tighten the controls (so to speak) by adjusting the values in lines 2570 to 2600.

Admittedly, this last section could stand some refinement and additions. Feel free to play with it. We felt that further and more meaningful conclusions could be drawn from all that data. However, what is there is all we could come up with. The matter of context constantly comes into any meaningful evaluation, and as we said earlier, we can't do context - yet. ●

```
100 REM ** WRITER.BAS * CREATED FOR CODEWORKS MAGAZINE **
110 DIM A$(50),S$(500),W$(200,3),P(500),R$(20):CLEAR 1000
120 GOSUB 140
130 GOTO 150
140 PRINT CHR$(12):RETURN
150 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
160 PRINT"                  W R I T I N G   A N A L Y Z E R"
170 PRINT"            examines the mechanics of your writing"
180 PRINT STRING$(60,"-")
190 PRINT
200 PRINT
210 PRINT
220 PRINT" 1) Enter new text to analyze."
230 PRINT" 2) Load and analyze existing text from disk."
240 PRINT
250 INPUT"Your choice";Z
260 IF Z<1 OR Z>2 THEN GOTO 250
270 ON Z GOTO 280,610
280 GOSUB 140
290 PRINT"Do NOT use hyphenated words at the end of the line."
300 PRINT"Do NOT use periods for abbreviations in any line."
310 PRINT"Do NOT indent paragraphs."
320 PRINT"Do NOT allow the line to exceed screen width."
330 PRINT"Hyphenated words in a line are counted as one word."
340 PRINT"Enter more than 100 but less than 300 words (about one scree
n full)."
350 PRINT
360 PRINT"Use ZZZ or zzz on a new line to terminate."
370 FOR I=1 TO 50
380 LINE INPUT A$(I)
390 IF LEFT$(A$(I),2)="ZZ" OR LEFT$(A$(I),2)="zz" THEN GOTO 410
400 NEXT I
410 N=I-1
420 REM ** CLOSE UP EXTRA SPACES BEFORE SAVING ON DISK **
430 FOR I=1 TO N
440 X=1
450 FOR L=1 TO LEN(A$(I))
```

```
460 A=INSTR(X,A$(I),"  ")
470 IF A=0 THEN GOTO 510
480 A$(I)=LEFT$(A$(I),A-1)+RIGHT$(A$(I),LEN(A$(I))-(A+1)+1)
490 X=A+1
500 NEXT L
510 NEXT I
520 INPUT"WHAT WILL YOU NAME THIS FILE";F$
530 OPEN "O",1,F$
540 PRINT #1,N
550 FOR I=1 TO N
560 PRINT #1,A$(I)
570 NEXT I
580 CLOSE 1
590 INPUT"ENTER 1 TO ANALYZE THIS TEXT, 2 TO RETURN TO MENU";Z1
600 ON Z1 GOTO 620,120
610 IF Z=2 THEN GOSUB 140:PRINT"NAME OF FILE TO LOAD";:INPUT F$
620 B$=" X"
630 J=1
640 OPEN "I",1,F$
650 INPUT#1,N:PRINT N
660 FOR I=1 TO N
670 LINE INPUT #1,A$(I)
680 PRINT A$(I)
690 A$(I)=A$(I)+B$
700 X=1
710 A=INSTR(X,A$(I)," ")
720 IF A=0 THEN GOTO 800
730 S$(J)=MID$(A$(I),X,A-X):P(J)=J
740 IF RIGHT$(S$(J),1)="."OR RIGHT$(S$(J),2)=CHR$(46)+CHR$(41) OR RIGH
T$(S$(J),2)=CHR$(46)+CHR$(34) OR RIGHT$(S$(J),1)="?" OR RIGHT$(S$(J),2
)=CHR$(63)+CHR$(34) OR RIGHT$(S$(J),1)="!" OR RIGHT$(S$(J),2)=CHR$(33)
+CHR$(34) THEN B=B+1
750 IF RIGHT$(S$(J),1)=CHR$(34) THEN QT=QT+1
760 IF RIGHT$(S$(J),1)=";" THEN D=D+1
770 IF RIGHT$(S$(J),1)="," THEN E=E+1
780 X=A+1
790 J=J+1:GOTO 710
800 NEXT I
810 CLOSE 1
820 IF J=<99 THEN GOSUB 140:PRINT"TEXT UNDER 100 WORDS TOO SMALL TO EV
ALUATE PROPERLY.":END
830 PRINT:PRINT"There are ";J-1;" words and ";B;" sentences,"
840 PRINT"it will take approximately two minutes to examine this text.
"
850 REM ** CHANGE UC TO LC AND REMOVE ENDING PUNCTUATION ***
860 FOR Q=1 TO J
870 IF LEN(S$(Q))<2 THEN GOTO 980
880 FOR Q1=1 TO 2
890 IF RIGHT$(S$(Q),1)="." OR RIGHT$(S$(Q),1)=CHR$(34) OR RIGHT$(S$(Q)
,1)="," OR RIGHT$(S$(Q),1)=";" OR RIGHT$(S$(Q),1)="?" OR RIGHT$(S$(Q),
1)="!" OR RIGHT$(S$(Q),1)=":" OR RIGHT$(S$(Q),1)=")" THEN S$(Q)=LEFT$(
S$(Q),LEN(S$(Q))-1)
900 NEXT Q1
910 IF LEFT$(S$(Q),1)="(" THEN S$(Q)=RIGHT$(S$(Q),LEN(S$(Q))-1):PN=PN+
1
```

```
920 IF LEFT$(S$(Q),1)=CHR$(34) THEN S$(Q)=RIGHT$(S$(Q),LEN(S$(Q))-1)
930 FOR I=1 TO LEN(S$(Q))
940 C$=MID$(S$(Q),I,1)
950 IF C$=>"A" AND C$=<"Z" THEN MID$(S$(Q),I,1)=CHR$(ASC(C$)+32)
960 IF C$="-" THEN DH=DH+1
970 NEXT I
980 NEXT Q
990 PRINT"Sorting words into alphabetical order."
1000 REM ** SORT THE WORDS ***
1010 N=J-1
1020 M=N
1030 M=INT(M/2):PRINT"+";
1040 IF M=0 THEN GOTO 1180
1050 J=0
1060 K=N-M
1070 I=J
1080 L=I+M
1090 IF S$(P(I))=<S$(P(L)) THEN GOTO 1140
1100 T=P(I):P(I)=P(L):P(L)=T
1110 I=I-M
1120 IF I<1 THEN GOTO 1140
1130 GOTO 1080
1140 J=J+1
1150 IF J>K THEN GOTO 1030
1160 GOTO 1070
1170 REM ** FIND % OF SMALL MEDIUM AND LARGE WORDS **
1180 FOR I=1 TO J
1190 IF LEN(S$(P(I)))=>5 AND LEN(S$(P(I)))=<9 THEN MD=MD+1
1200 IF LEN(S$(P(I)))=>10 THEN SF=SF+1
1210 IF LEN(S$(P(I)))=8 THEN MW=MW+1
1220 IF LEN(S$(P(I)))=9 THEN LW=LW+1
1230 NEXT I
1240 SF=SF+(.8*LW)+(.45*MW)
1250 FOR I=1 TO J-1
1260 IF S$(P(I))<>S$(P(I+1)) THEN VC=VC+1
1270 NEXT I
1280 VC=VC+1
1290 REM ** FIND FREQ OF WORDS ***
1300 X$="#.###"
1310 U=1
1320 I=1
1330 FC=0
1340 L=I+1
1350 IF S$(P(I))<>S$(P(L)) THEN GOTO 1400
1360 Q=I
1370 IF Q=>J+1 THEN W$(U,1)=S$(P(I)):W$(U,2)=STR$(FC):W$(U,3)=STR$(50)
:U=U+1:GOTO 1420
1380 IF S$(P(I))=S$(P(Q)) THEN FC=FC+1:Q=Q+1:GOTO 1370
1390 IF FC=>3 THEN W$(U,1)=S$(P(I)):W$(U,2)=STR$(FC):W$(U,3)=STR$(50):
U=U+1:I=Q-1
1400 I=I+1:GOTO 1330
1410 REM ** ORGANIZE MOST USED WORDS ***
1420 U1=U
1430 FOR I=1 TO 15
1440 READ R$(I),R
```

```
1450 FOR Q=1 TO U1
1460 IF W$(Q,1)=R$(I) THEN W$(Q,3)=STR$(I):GOTO 1490
1470 NEXT Q
1480 W$(U+1,1)=R$(I):W$(U+1,2)=STR$(0):W$(U+1,3)=STR$(I):U=U+1
1490 NEXT I
1500 F=0
1510 FOR I=1 TO U-1
1520 L=I+1
1530 IF VAL(W$(I,3))=<VAL(W$(L,3)) THEN GOTO 1580
1540 T$=W$(I,1):W$(I,1)=W$(L,1):W$(L,1)=T$
1550 T$=W$(I,2):W$(I,2)=W$(L,2):W$(L,2)=T$
1560 T$=W$(I,3):W$(I,3)=W$(L,3):W$(L,3)=T$
1570 F=1
1580 NEXT I
1590 IF F=1 THEN GOTO 1500
1600 AS=J/B:SW=((J-(SF+MD))*100)/J:MW=(MD*100)/J:LW=(SF*100)/J:SC=(D*1
00)/J
1610 PN=(PN*100)/J:HN=(DH*100)/J:QT=(QT*100)/J:UW=INT((VC/J)*100)
1620 CM=(E*100)/J:ML=INT(((MD+SF)/J)*100)
1630 FI=(((J/B)+((SF*100)/J))*.4):FI=INT(FI+.5)
1640 GOSUB 140
1650 PRINT "R E S U L T S  for ";F$
1660 PRINT
1670 PRINT"TOTAL WORDS ---------------- ";J
1680 PRINT"TOTAL SENTENCES ------------- ";B
1690 PRINT"AVG WORDS/SENTENCE ---------- ";AS
1700 PRINT"% SHORT WORDS --------------- ";SW
1710 PRINT"% MEDIUM WORDS -------------- ";MW
1720 PRINT"% LONG  WORDS --------------- ";LW
1730 PRINT"% SEMI-COLON ---------------- ";SC
1740 PRINT"% COMMAS -------------------- ";CM
1750 PRINT"% PAREN PAIRS --------------- ";PN
1760 PRINT"% HYPHENS ------------------- ";HN
1770 PRINT"% QUOTE PAIRS --------------- ";QT
1780 PRINT"% UNIQUE WORDS ---------------";UW
1790 PRINT"% MEDIUM & LONG WORDS USED ----";ML
1800 PRINT"FOG INDEX ------------------->";FI
1810 PRINT
1820 LINE INPUT"Press RETURN for further analysis.";ZZ$
1830 GOSUB 140
1840 RESTORE
1850 F1$="          0.1%   1%        2%        3%        4%        5%
     6%"
1860 F2$="             |    |    |    |    |    |    |    |    |    |
   |    |"
1870 PRINT"          Smoothed average of 50 well-known writers use of
most common words."
1880 PRINT TAB(17);"Common word usage as a % of total words in their t
exts."
1890 PRINT F1$
1900 PRINT F2$
1910 PRINT TAB(15);STRING$(56,"-")
1920 FOR I=1 TO 15
1930 READ R$,R
1940 R=(R/10)+1
```

```
1950 IF R<0 THEN R=10
1960 PRINT R$;TAB(15);STRING$(R,"x");TAB(72);R$
1970 NEXT I
1980 RESTORE
1990 PRINT
2000 LINE INPUT"Press RETURN to see how you used the same words.";ZZ$
2010 GOSUB 140
2020 DATA the,550,of,380,and,289,to,218,a,174,in,127,that,95,was,78,it
,55,had,48,for,36,be,26,is,15,as,6,or,1,0,-1
2030 PRINT TAB(20);"Your common word usage as a % of total words."
2040 PRINT F1$
2050 PRINT F2$
2060 PRINT TAB(15);STRING$(56,"-")
2070 FOR I=1 TO 15
2080 L=I+1
2090 READ R$,R
2100 R1=VAL(W$(L,2)):R2=R1:R1=(R1*100)/J:R1=INT(R1*100):IF R1>611 THEN
 R1=610:R3=INT((R2/J)*100):PT$=W$(L,2)+"("+STR$(R3)+"%)" ELSE PT$=STR$
(R2)
2110 R1=(R1/10)-5
2120 IF R1<0 THEN R1=1
2130 PRINT R$;TAB(15);STRING$(R1,"x");TAB(72);PT$
2140 NEXT I
2150 PRINT
2160 LINE INPUT"Press RETURN for next graph";ZZ$
2170 GOSUB 140
2180 PRINT TAB(20);"Other words you used more than 3 times."
2190 PRINT F1$
2200 PRINT F2$
2210 PRINT TAB(15);STRING$(56,"-")
2220 LS=17
2230 FOR L=17 TO U
2240 LS=LS+1
2250 R1=VAL(W$(L,2)):R2=R1:R1=(R1*100)/J:R1=INT(R1*100):IF R1>611 THEN
 R1=610:R3=INT((R2/J)*100):PT$=W$(L,2)+"("+STR$(R3)+"%)" ELSE PT$=STR$
(R2)
2260 R1=(R1/10)-5
2270 IF R1<0 THEN R1=1
2280 PRINT W$(L,1);TAB(15);STRING$(R1,"x");TAB(72);PT$
2290 IF LS<33 THEN GOTO 2340 ELSE LS=17:PRINT:PRINT"Press RETURN for m
ore..";:LINE INPUT ZZ$
2300 GOSUB 140
2310 PRINT F1$
2320 PRINT F2$
2330 PRINT TAB(15);STRING$(56,"-")
2340 NEXT L
2350 IF U=<17 THEN PRINT"Apparently you used no others."
2360 PRINT
2370 INPUT"Repeat the result sequences (y/n)";ZZ$
2380 IF LEFT$(ZZ$,1)<>"N" AND LEFT$(ZZ$,1)<>"n" THEN GOTO 1640
2390 GOSUB 140
2400 LINE INPUT" Psssst...  can we talk? (y/n)";ZZ$
2410 IF LEFT$(ZZ$,1)<>"N" AND LEFT$(ZZ$,1)<>"n" THEN GOTO 2450
2420 GOSUB 140
2430 PRINT"If we can't communicate, then it's all over."
```

```
2440 END
2450 PRINT"There is no way for a computer to evaluate the subject matt
er of"
2460 PRINT"your writing. Those writing techniques which may be quantif
ied"
2470 PRINT"however, will indicate your success in reaching your indend
ed"
2480 PRINT"audience. Please answer the following question or questions
."
2490 PRINT
2500 PRINT"What educational grade level (1 thru 24) are you aiming at"
;:INPUT GL
2510 IF FI=<GL THEN GOSUB 140:GOTO 2620
2520 PRINT"You missed the intended grade level by ";FI-GL
2530 PRINT
2540 PRINT"Based upon your sample text, here are some suggestions whic
h may"
2550 PRINT"bring your readability index down to the level you desire."
2560 PRINT
2570 IF AS>20 THEN GOSUB 2670
2580 IF ML >50 THEN GOSUB 2740
2590 IF SC>1 THEN GOSUB 2810
2600 IF UW<50 THEN GOSUB 2860
2610 GOTO 2640
2620 PRINT"Your writing should easily be understood by people with an"
2630 PRINT"education through grade";FI
2640 PRINT
2650 PRINT"End of program."
2660 END
2670 REM * long sentence response *
2680 PRINT"Your sentences have an average length of ";INT(AS);"words."
2690 PRINT"You may look for complex sentences and break them up into"
2700 PRINT"smaller ones. Change dependent clauses to independent claus
es."
2710 PRINT"If long sentences are necessary, counter-balance them with"
2720 PRINT"several shorter ones."
2730 PRINT:RETURN
2740 REM * too many medium long word response *
2750 PRINT"Shorten the average length of your words. Of all the words"
2760 PRINT"you used,";ML;" percent were medium and long. You may want"
2770 PRINT"to rewrite your sentence after changing words to maintain"
2780 PRINT"the rhythm. It helps to exchange long, complex words with"
2790 PRINT"two or three short ones."
2800 PRINT:RETURN
2810 REM * semi-colon response *
2820 PRINT"You used";INT(SC*100);"semi-colons. These may be necessary,
"
2830 PRINT"but if used to separate two or more complete thoughts, it w
ill"
2840 PRINT"help readability to restructure the sentence."
2850 PRINT:RETURN
2860 REM * unique word count response *
2870 PRINT"You have used more than half the words in your text two or"
2880 PRINT"more times. Both reader interest and readability increase"
2890 PRINT"when a greater variety of words are used."
2900 RETURN
```

Maker

Automatically Generate DATA Statements

Staff Project. Maker is one of those serendipitous programs that come up from time to time. We needed something to generate data lines for another program. The other program was a flop and is long forgotten. This, the tool we created, hung on and was used more and more. This one too, uses ASCII in saving the file, and like two other programs in this issue, should be a useful addition to your utility library.

When large amounts of data are to be entered we generally tend to ignore the use of DATA statements. They are somewhat inflexible, and when different data sets are desired they are time consuming and difficult to exchange. The commas separating data elements become a problem when entering numbers, since the numerical keypads on most computers do not contain the comma.

The short program presented here is designed to change that. With it, you can generate data statements of all kinds: numerical, string or mixed. It eliminates the need for entering line numbers, the word DATA and the commas between elements. You simply enter the starting information and then type in your data elements, pressing RETURN after each one. The program does all the rest of it for you.

After you have created your statements, the computer will automatically save them in ASCII format. Now, when you load your applications program that uses the data, you simply merge the data file.

This opens a large number of possibilities. It should be obvious that you can now create various sets of data and merge them as needed. If you used the same line numbers for all the data sets, you don't even have to delete the old set before merging the new one since it will write directly over the old.

In other cases, you may want to alternate line numbers so that the sets of data merge with each other. The data sets can easily be manipulated. They remain as distinct sets on your diskette and can be merged into your program as needed. If there are conflicts with line numbers between sets, you can load them from the diskette just like an ordinary BASIC program and renumber them. In this case, when you save them back be sure to use the ASCII identifier (SAVE"filename",A), so that they can later be merged into another BASIC program. You can now manipulate, renumber and merge as many data sets as you like. The merge command is simple: MERGE "filename.DAT".

The program allows you to specify how many elements will be in each data line. At first, this didn't seem so necessary, but after some use in entering monthly totals for several years, it became apparent that if there were twelve items per line, an entire year could be eliminated by deleting a single line number.

The program is relatively short, considering what it does, and as usual there is more housekeeping code than actual "guts". The only comment needed for the first third of the program is about line 120. This is a CLEAR SCREEN line. It happens again in line 520. The CHR$(12) in the program is clear screen for CP/M machines. Some CP/M based machines may need to change this to a CHR$(26). IBM-PC, most TRS-80's and others simply change the line to a CLS command.

While on the subject of compatibility, the file extension (.DAT) in lines 640 and 710 is rather standard for MS-DOS and MBASIC machines. Some, however, may need to change this to /DAT. The remainder of the program is standard BASIC throughout. Let's look at the code and see what it does.

In line 280 we define variables I, L, N and Y as integers. This just speeds up the processing a bit. You don't even notice it unless you have entered several hundred data elements. Next, we clear 5000 bytes for string space. You may want to up this if you have the space and need to enter large amounts of data.

In line 290 we set up two single dimension arrays. The first A$() will hold the data elements we type in. It is set for a limit of 1000 elements. The second Z$() will hold the completed data lines once they are put together by the computer. Obviously, if you were to tell the program you only wanted two elements per line and you have 1000 elements, Z$() will need to be changed to at least 500.

This is basically a program which writes a portion of BASIC code. So it needs to know what the starting line number is, the increment between lines and how many data elements there will be on a line. These questions are asked in lines 300 through 320. There must also be some method of ending input of data elements, so line 330 asks what that unique sentinel character will be. Since most data arrays need some sort of sentinel in any case, the same sentinel you use here will be appended to the actual data statements as the data sentinel.

In general, it is wise to pick a sentinel that cannot be confused with live data. When all values are positive, for example, a -1 may be a good sentinel. If data is mixed, both positive and negative, -1 won't work too well, so pick something very large, like 2E9 or -2E9. If your data elements are alpha, use the word END and if END is a valid data element, then use period end (.END), which would probably not be a valid data element.

The next thing we do in line 370 is to define A$ as the word DATA with a space before and after it. Some computers do not care about spaces around key words, but most do, so it is best to use them.

Lines 380 through 410 comprise a loop used to input the data elements. Line 400 checks each entry to see if it is the sentinel character. As long as we have not input the sentinel character, single dimension array A$(I) accumulates the data elements. Keep in mind that A$(I) and A$ are totally different variables. The "I" in A$(I) keeps track of how many elements there are in array A$(I). When we input the sentinel character, line 400 forces us out of the loop, right around the NEXT I, to line 420.

In line 420, N is set equal to I less one. The reason for this is that the loop counter I is already advanced one position for the next entry and there is none. So we subtract one from the I number to get the actual number of elements. Also at this point, we initialize Y to be equal to 1.

Lines 440 through 510 are the actual "guts" of this program where everything happens. In these lines we have two loops, an outer "X" loop and an inner "Q" loop. The outer loop will count the number of data statements we will end up with. The inner loop will count the number of elements within each data statement.

Taking a stroll through these lines lets us see what actually happens. First, let's set up some initial values. Say that L, our starting line number was set at 1000, the line increment IC, was set at 2 and our number of elements per line, LL, was set at 4. Let's also assume that our sentinel character was defined as END. Let's now say that we have entered data elements called 1, 2, 3, 4, 5, 6 and END, just to keep it simple.

Variable N now contains the number 7. In line 430 we start the X loop and set its termination point at the integer of N divided by the number of elements per line plus 1. Since N is seven and LL is four, the integer will be 1. It's obvious that we will need two lines of data statements to hold these elements, so that is why we have to add one in line 430.

We now know that there will be two BASIC lines of DATA, starting at line 1000. In line 440 we start to build our DATA line. Here, we introduce a new variable Z$. It is set equal to the string value of L (our starting line number) plus A$ (which is the word DATA with spaces around it, remember?) At this point, our data statement looks like this:

```
1000 DATA
```

Now we run smack-dab into our inner Q loop. It has to take the data elements from the A$() array and put the right number of them into the data statement. It also has to be careful to put the comma in the right places and be sure not to put one at the end of the line. Aside from that, it has to watch for the sentinel character and put it into the right place too, even if it is the only element on a line, and not put a comma after it either. Earlier, we initialized Y to be 1. It comes into play here.

The Q loop will start counting at 1 and go to 1 plus 4 less 1, which happens to be 4, the number of elements we want in each line of data. Line 460 says that if Q is equal to 4 then B$ is equal to nothing, else if Q does not equal 4 then B$ is equal to CHR$(44). CHR$(44) just happens to be the ASCII value for the comma. Since Q is at 1, B$ is equal to the comma. We can skip line 470 for now because we haven't reached the sentinel character yet. Note that we are reading the A$() array with the Q loop this time. Remember that it was created with an I counter? This is legal, you can create an array with one variable and read it back out with a different one anytime you want to.

If you will recall, our DATA statement already looks like this: 1000 DATA. Now, inside the Q loop

at line 480 we take that much of the data statement (which is equal to Z$) and make it equal to itself plus A$(Q) (the first data element, 1) and B$ (which is the comma.) The data statement now looks like this:

1000 DATA 1,

After three more passes through the inner Q loop, the data statement looks like this:

1000 DATA 1,2,3,4

Line 460 found Q equal to Y plus LL minus 1 after the fourth data element and so B$ was a null. Also, after the fourth data element was read in, the Q loop was exhausted and so we leave it and go back to the outer X loop in line 500.

Here, we add the increment 2 to the starting line number, start a new array called Z$(X) to hold the data statements and increment Y by the number of elements per line. Y now will equal 5, which happens to be the next data element we want to deal with on the second DATA line.

We now encounter the NEXT X in 510, and go back to start the whole process over. This time Z$ will equal: 1002 DATA, and the inner Q loop will supply the 5 and 6, and in line 470 it will encounter the sentinel END. When that happens in line 470, we make the last DATA line and get out of there. Note that when A$(Q) equals END, Z$(X) equals Z$ plus A$(Q) (without B$ this time.) So, now our entire data lines consist of:

1000 DATA 1,2,3,4
1002 DATA 5,6,END

That's just what we wanted. The data lines are held in memory in the Z$(X) array. Z$(1) is line 1000 and Z$(2) is line 1002. At this point, we are done formulating the data lines and so line 470 sends us to line 520, which simply clears the screen.

Line 530 prints the lines of data statements on the cleared screen for us to admire.

Lines 550 through 600 give the option of saving the file on disk or aborting. *Be careful!* If you decide to abort by typing R or r, all your data elements will be lost. The option is included in case you have made gross mistakes and want to start over.

If you opt for neither rerun or save, line 610 keeps nagging you to rerun or save. Any key, except upper or lower case R or S will send the nag back.

When you opt to save the file, line 620 asks for a filename, which becomes F$. The program automatically appends the file descriptor .DAT to the filename, so that you will know it is a data file when you see it in the disk directory.

Lines 640 through 680 open the sequential file for output and loops through the Z$() array and prints it to the diskette. It then closes the file. The file is automatically saved in ASCII format on the diskette. It can be loaded like any BASIC program and can be merged with any BASIC program. Keep in mind that once loaded like a BASIC program it will no longer be in ASCII format, and if you want it to be, you must save it back with the file specifier ,A after the filename.

We have found this program to be extremely useful in correlation programs where various sets of data were to be compared to each other. It was also used heavily in a horse-racing program where post positions were tabulated and compared. Someone in the office even used it to track and compare winning state lottery numbers.

The bootstrapping ability of this program is interesting. It is standard code that writes standard code. How far can this idea be stretched? Do you suppose we may yet get to plain English input? Have fun! ●

```
100 REM ** MAKER.BAS * CREATED FOR CODEWORKS MAGAZINE **
110 CLEAR 5000
120 PRINT CHR$(12):REM-CLEAR SCREEN-CHANGE TO SUIT YOUR MACHINE.**
130 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
140 PRINT"                    D A T A   M A K E R"
150 PRINT"          Automatically generates data statements."
160 PRINT STRING$(60,"-")
170 PRINT"  This program will generate DATA statements without the nee
d"
180 PRINT"of entering line numbers, the word DATA, or the commas"
190 PRINT"between elements.This is handy since numerical keypads usual
ly"
200 PRINT"do not contain the comma."
```

```
210 PRINT
220 PRINT"  The program generates an ASCII file of these data statemen
ts"
230 PRINT"which can be merged with any other BASIC program. Limits are
"
240 PRINT"set in lines 290 and 380."
250 PRINT"  Start your line numbers high enough to insure they will no
t"
260 PRINT"crash into existing numbers when you merge programs later."
270 PRINT
280 DEFINT I,L,N,Y
290 DIM A$(1000),Z$(100)
300 INPUT"WHAT STARTING LINE NUMBER DO YOU WANT";L
310 INPUT"WITH A LINE INCREMENT OF";IC
320 INPUT"HOW MANY ELEMENTS PER LINE";LL
330 LINE INPUT"WHAT IS YOUR SENTINEL CHARACTER? ";S$
340 PRINT"Use < ";S$;" > to end input - it will"
350 PRINT"also be recorded as the last data element."
360 PRINT
370 A$=" DATA "
380 FOR I=1 TO 1000
390  PRINT"Enter data item #";I;"> ";:LINE INPUT A$(I)
400  IF A$(I)=S$ THEN GOTO 420
410 NEXT I
420 N=I-1:Y=1
430 FOR X=1 TO INT(N/LL)+1
440  Z$=STR$(L)+A$
450   FOR Q=Y TO Y+LL-1
460    IF Q=Y+LL-1 THEN B$="" ELSE B$=CHR$(44)
470    IF A$(Q)=S$ THEN Z$(X)=Z$+A$(Q):GOTO 520
480    Z$=Z$+A$(Q)+B$
490   NEXT Q
500  L=L+IC:Z$(X)=Z$:Y=Y+LL
510 NEXT X
520 PRINT CHR$(12)
530 FOR I=1 TO X:PRINT Z$(I):NEXT I
540 PRINT
550 PRINT"You may (S)ave the file on disk or,"
560 PRINT"abort and (R)erun the program."
570 PRINT"       (Enter S or R)"
580 INPUT"Your choice ";X$
590 IF X$="R" OR X$="r" THEN GOTO 100
600 IF X$="S" OR X$="s" THEN GOTO 620
610 GOTO 570
620 PRINT"What name will you use for this file?"
630 LINE INPUT"I'll append DAT for the file type. ";F$
640 OPEN"O",1,F$+".DAT"
650 FOR I=1 TO X
660  PRINT#1, Z$(I)
670 NEXT I
680 CLOSE 1
690 PRINT
700 PRINT"DONE, NOW LOAD YOUR BASIC PROGRAM AND"
710 PRINT"ISSUE THE COMMAND TO MERGE ";F$+".DAT"
```

Sources

Where to Find Programming Tools

The listing of products in this column is to tell our readers what is available in the marketplace. We take news release items at their face value. No endorsement by this publication is implied by the appearance of products in this section.

Logical Systems Inc. has made available Little Brother, a database manager with sophistication, simplicity and value, for the TRS-80 Models 4/4P and IBM-PC. The program is menu driven and comes with complete on-line help information always at your fingertips. It was designed for ease of use in both normal operation and in setting up the database. Hardware specifications for the Model 4/4P are a minimum two floppy disks and 128K of RAM (hard disk owners need only have 64K and one floppy). For the IBM-PC, two floppy disk drives (or one hard disk and one floppy) and 128K of RAM are required. The price for either version is $99. Logical Systems Inc., 8970 North 55th St., Milwaukee, WI 53223 (414) 355-5454

Software Studios, Inc. has introduced PC-DESK III, a new desk top management program for the IBM-PC and compatibles that includes a full-function word processor. The $49 program has a Calendar/Reminder, Calculator, Automatic Phone Dialer, Repetitive Letter Writer (Mail Merge), and a name and address database. In addition, PC-DESK III features memory partitioning which divides memory into discrete segments allowing two programs to be resident at the same time. Information can be transferred betweeen partitions, or "DESKS", through simple commands. It is available for $49. plus $2 shipping from Software Studios, Inc., 8516 Sugarbush, Annandale, VA 22003 (703) 978-2339

TaxCalc Software Inc. now offers a spreadsheet template with a tool kit full of financial decision making software. This template uses "what if?" analysis to help users calculate loan balances and payments, adjustable mortgage rate payments, amortization schedules, investment income and more. It includes three sections - investments, loans and amortization. The investment section calculates future value, minimum investment for withdrawals, regular withdrawal amounts, nominal interest rate, annual effective interest rate, regular deposit, continuous interest compounding and more. Loans calculates principal, loan payment amount, remaining loan balance, term of loan, adjusted rate mortgage payments and more. The amortization section is an unlimited amortization schedule. Financial Decisions Planner works with Lotus 1-2-3, VisiCalc, SuperCalc and Multiplan on IBM-PC and compatible systems. Minimum requirements are one disk drive and 128K of RAM. TaxCalc Software Inc., 4210 West Vickery Blvd., Fort Worth, TX 76107 (817) 738-3122

Software Studios, Inc. has introduced a new programmer's tool, UNNUMBER, which will automatically strip a BASIC or BASICA program of all unreferenced line numbers prior to compiling. UNNUMBER permits compiler optimization between statements and line numbers and reduces the size of EXE files. An unnumbered file will execute up to 30% faster and consume less disk storage space. The program will run on floppy disk or hard disk and operates quickly and reliably. UNNUMBER is available for $25 plus $2 shipping from Software Studios, Inc., 8516 Sugarbush, Annandale, VA 22003 (703) 978-2339

Enter Computer, Inc., manufacturer of single and six-pen "Sweet-P" plotters has introduced "TYP-SET", an easy to operate lettering software package for use on the IBM-PC and compatible microcomputers. The software enables the PC user to quickly place individual lines of text (letters, numbers and punctuation marks) at specific positions on an output page of paper, acetate for overhead projections or self-adhesive labels, through use of a plotter. To operate the TYP-SET fully menu driven software, a user needs an IBM-PC or compatible micro with 192K RAM and two disk drives. For plotters other than Enter Computer's Sweet-P 600, an RS-232 interface operating at 9600 baud is required. The TYP-SET software package, with six fonts, costs $299. Additional sets of four fonts cost $199 per set. Enter Computer, Inc., 6867 Nancy Ridge Drive, San Diego, CA 92121 (619) 450-0601

ASCII Codes

What's this ASCII all about?

ASCII (Ask-key) American National Standard of Information Interchange. The "ASCIIbet" predates computer technology to the extent that only a very few of the old technoids remember its inception. That fact alone would not be notworthy, except the old technoids don't have grey hair yet.

ASCII derives its most fundamental characteristic from the eight bit byte. Two to the eighth power is the source of the 256 character combination range. By definition, one of the bits is used as a parity bit, cutting the total number of possible combinations in half, or down to 128 characters.

The largest chunk of the code set goes to the alphabet with 26 upper and 26 lower case letters, claiming 52 total combinations. The numeric digits zero through nine get an additional 10 characters bringing the count up to 62. Of the remaining 66 code bytes possible, the first 32 were claimed back in the old days as function codes for serial data transmission, bell ringing on KSR's (teletypes), tab's, carrage returns and most of the control codes you now use on your computer.

That leaves only about 34 possible characters, most of which have been used by scientific types for algebraic and relational operators. Which is just about great. Of the 128 possible combinations, there's almost enough space for everything.

On the ground floor lobby of the building where I work there is a thirty-five year old "burned-out" systems analyst named Ralph. He sells newspapers and cigars. I figured Ralph knew what ASCII was all about.

"Say, Ralph, what is ASCII anyway?"

"You mean the 'ASCIIbet'? It has real staying power. Can't compute without it."

"What do you mean - staying power?" I asked.

"If you remember way back to the olden days of computers there were things around like card readers, core memory and Hollerith code which you never hear about anymore. Want a paper?"

"You got today's? You mean ASCII goes way back to the old days of computing?"

"Yeh, Hollerith code used by the tab-card people was the predecessor, but I haven't seen a tab-card for years now. I still get an urge to fold, spindle or mutilate. I got a yesterday's paper."

"I'd rather have today's. I saw a vacuum tube once in a museum. Well doesn't the ASCII character set just basically have an upper and lower case alphabet and the numbers zero to nine?" I asked.

"Yeh, but that's only 62 of the 128 possible combinations. There are 32 characters for controlling data transmission and peripheral equipment functions, plus all the algebraic operators, and don't forget punctuation. Look at any keyboard and you'll see most of the ASCII character set. Today's hasn't happened yet, so there's no such thing as today's news." he reminded me.

"Well, the II in ASCII stands for information interchange." I ventured, "So aren't you really saying ASCII is an agreed upon standard character set for all computers everywhere?"

"But the A stands for American, so by everywhere you must mean every-American-where. There was a lot of news made the day before yesterday." says Ralph, "I'll get you a paper."

"Okay, if the numeric digits are really hex FO to hex F9 then you wouldn't want to do arithmetic operations in ASCII, but I guess text editing, formats and compares work okay, don't they?"

"You're right, but some even know ways to do ASCII arithmetic. That will be 50 cents.", he said, handing me a day old paper and smiling.

"Maybe this is ASCII arithmetic." I said, handing over the 50 cents. "I was just wondering, Ralph, if you know this much about computers why are you selling papers?"

"You must not know what the profit margin is for selling day old newspapers, Hey, you want to buy some cigars?" he asked.

"No thanks, by the way, do you have a favorite ASCII character?"

"Yeh, it's Hex 04."

"What's that?"

"End of transmission."

Jay

Hype

Where we ask you to Subscribe!

To help us serve you properly, would you kindly take a moment to answer the following questions -

Make and Model of your computer _____

Do you have disk drives? ☐ **Yes,** ☐ **No** Which DOS do you use? _____

Do you have a line printer? ☐ **Yes,** ☐ **No**

Do you use a Modem? ☐ **Yes,** ☐ **No** What baud rate **300** ☐ **1200** ☐

As a Basic programmer, do you rate yourself as ☐ **Novice,** ☐ **Intermediate or** ☐ **Advanced?**

Which other programming languages do you prefer? _____

Comments:

Fold

Subscription ORDER FORM

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

☐ **Check or MO enclosed.**
☐ **Charge to my VISA/MasterCard #**_____**Exp date**_____

Please Print clearly:

Name _____

Address _____

City _____ **State** _____ **Zip** _____

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

Charge card orders may be called in (206) 475-2219 between 9 AM and 4 PM weekdays, Pacific time. Sorry, no "bill me" orders.

Coming Attractions

What to expect in future issues

Most programs operate around a core formula or algorithm. How do you write a program when there is no such thing? We took that idea and made it work in a program called "Wood", which lets the computer try to cut up sheets of plywood efficiently. It even takes into account the saw kerf and the grain of the wood. Even though it's approach is cut and try, the odds are in favor of the program. It does multi-sheet cuts in a few minutes, and prints out a pictorial cutting schedule for your shop wall. Look for it in a future issue.

Card File is a program built for the novice computer user. You have often heard the question: "But what can it do? Will it keep my Christmas card list?" Yes, it will, and a lot of other smaller tasks as well. It was designed with the non-computer person in mind, and even loads and saves the files automatically. Aside from that you can sort or search on any field and you can have as many fields as you want. It should be the project in the next issue if all works well.

New Year we will just have to have a calendar program, no? Golly, like there aren't enough already. Maybe we'll fix this one up to do something personalized and special for you.

Of course, there will be shorties and tips and tricks. We will continue with the beginner programming articles too, in case you are new to programming. See you then..

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

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

CodeWorks

Issue 2 Nov/Dec 1985

CONTENTS

CODEWORKS

Issue 2 Nov/Dec 1985

Editor/Publisher
Irv Schmidt
Associate Editors
Terry R Dettmann
Greg Sheppard
Jay Marshall
Circulation/Promotion
Robert P Perez
Editorial Advisor
Cameron C Brown
Technical Advisor
Al Mashburn

Produced by 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. Neither is any liability assumed for damages resulting from the use of any information contained herein. Please address 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 please) and allow 4 to 6 weeks for editorial review. Do not send diskettes, rather send a hard copy listing of programs. Media will be returned if return postage is provided. Cartoons and photographs are welcome. 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), one year only. Not available outside United States Zip codes. **VISA and Master Card orders are accepted by mail or telephone.**

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in the United States of America. Bulk rate postage 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 (at no cost).

Point of View

Starting a new magazine is exciting. You plan and plot and hope to find acceptance. Watching the mail come back and reading the comments is the only way to see how well things are going. It has been a fun time, during the past two months, doing that here. We apparently judged correctly, that there is interest in programming in BASIC. Your response is appreciated and encouraging.

It was surprising though, to find that many people who did not order, took the time to write back anyway. Some said that they were "way beyond" what we were doing. Others said that their favorite language was *anything* but BASIC, and that BASIC was a "yuk" language. We certainly expected some of that, but to take the time to write and tell us seems a bit unusual. It's not that often that you get such good feedback from people who have decided, for one reason or the other, *not* to buy.

There are three major aspects to this publication. First, and most important, is the content of the magazine. It must be something you want or it won't sell. Simple as that. Second is the form the magazine takes - typesetting, layout, paper, on-time delivery and all that. Third is the magazine as a business. This is the one that you, as readers, do not worry about until the business is no longer viable and falls apart.

I am happy to report that your response has been such that we are off to a very good start. We are still tabulating the different computers that our readers have. Right now, it seems to be a good mix of all of them. One person still has the old IMSAI 8080, vintage 1976 or 77. Once we have a good look at the distribution of machines, we can better target the material we present.

We are continuing our efforts to reach more people. Because of the economy of scale, there is some magic number we would like to reach. We are currently about one-fourth of the way there.

A few years ago, you could take a shot at the computer community and hit most of them with a well-choked shotgun effect. Nowadays, you have to spread that shot very thin and wide, and the return is very much smaller. In plain English, it's costing a lot more to get a lot less.

We appreciate all the names and addresses many of you sent along with your order. Our free sample copy policy is still in effect. If you know someone who would be interested, just let us know. We are growing and need all the push we can get. In the end, it will make a better magazine for all of us.

Since we won't be in touch again till January, we wish each of you happy holidays.

Irv

Forum

An Open Forum for Questions & Comments

I am having a few problems with the program "Writer" in your premier issue which I have punched in from the keyboard and saved on disk. I am using a TRS-80 Model IV and TRSDOS 6.2 and have debugged all my own errors in transcribing the program, but run into "subscript out of range" on line 670. I hope you can give me a hint or two so I can make this program run properly. It is an interesting program and I would certainly like to use it if it would work. - C W Preble, Mill Valley, CA

We put a CLEAR statement at the end of line 110. This was clearly a mistake. The reason for that mistake is that the program was originally written on a later version of BASIC which dynamically clears string space. We caught the error before publication when we tried to run the program on a different machine. Problem was, we stuck the CLEAR statement at the end of line 110 instead of before it. If your computer needs to clear space, add line 105 CLEAR 10000 and remove the CLEAR at the end of line 110 (leave the DIM statement as it is though).

I can't get your program "Writer" to work on my Model 100 TRS-80. I keep getting bad subscript errors. Do you have any suggestions? - Kenneth G Oxley, Dayton, OH

See answer to the above letter. Also, unless you change the PRINT commands to LPRINT (and have an 80-column printer) you will have problems seeing what is going on with the limited screen of the Model 100.

I like the idea of your new magazine! However, I noticed an error in the program listing of Writer in your first issue. I believe that the CLEAR statement in line 110 should have preceeded the DIM statement instead of vice versa. When running the program as listed, the CLEAR statement clears the recently dimensioned variables. This results in a subscript out of range error. I look forward to your publication... - Alex Roosakos, Millbrae, CA

Thank you, and here is yet another letter reference the Writer program...

...let me say again how much I am enjoying the use of the Writer program. Now for a request or a suggestion: There does not seem to be any method to correct typing errors on the writing sample submitted to the program. If the error is not caught before the line is entered, the error either remains or the program has to be restarted. Can you come up with a way to permit corrections to be made? - Jack Hill, Omaha, NE

That brings up an interesting question which we pondered over when we wrote the program. We chose not to include editing because it would necessarily complicate the program and probably end up being a mini-text editor in itself. If you own a word processor, you can call up the sample text and edit it. We did it regularly with WordStar. If you do this and add lines though, be sure to update the number in the first line of the file. This is the number of lines of text that is saved with the file when you created it with Writer. Minor editing can be done this way, but we found that files created with WordStar or Scripsit would not play right with Writer, mostly because of the way these programs treat the carriage return at the end of the line (and other assorted imbedded codes).

Congratulations! Assuming that your first issue is but the forerunner of many, you have hit my interest and needs so squarely dead center that I almost cry "Synchronicity!" There has been a cadre of readers of all the "other" computer magazines who have bugged editors for just the type of material you are presenting, but who have been given short, though courteous, "Tough luck, bro!" It is my hope that you will find those disappointed seekers ...The once swollen computer magazine racks are decimated - a fitting memorial to publications fat with advertisements and larded with excessive hard- and software reviews, and editorial selections designed to capture everyone as a reader who ever heard of a computer. As a result of such generalizing it hardly mattered which ones you scanned, they all read the same. Withal, the user who learned BASIC and stuck with it could find few magazines worth the heavy tab, and even these went belly up with but a notable exception or two... I would not object to a few very special "messages" from hardware and software merchants, but they should be of the type that rewards the reading, not just hops up and down on one leg like the king's jester, screaming nonsense for the sole purpose of attracting attention. Let your policy ring loudly with SERVICE to the reader, and you won't go wrong... - Waldo T Boyd, Geyserville, CA

Many magazines get many letters from readers because their listed programs contain typo mistakes. Many listed programs are written in such small type that it is most difficult to copy them into the computer without lots of mistakes and missing lines. Why can't photo-copies of a computer listing be put into magazines without setting the type and eliminate one source of error? - John Grass, Redwood City, CA

We do, John. Our listings are run off on a daisy-wheel printer and pasted up directly in full size. In spite of that, there can still be some problems to overcome. For example, we used the variable AS in the Writer program and found that on

Sanyo 550 BASIC, AS is a reserved word (change it to AV if this is a problem). The CLEAR statement in the same program was another problem. In this issue, there is one place in the Calendar program that requires the use of the linefeed (CRTL-J) that will probably cause a few problems. We use 80-column screens here, and sometimes (as in the first issue) forget that some people have only 64 columns. Writing for a variety of machines is a challenge, but we are catching up on the differences and overcoming them when possible and avoiding them when necessary.

Looks like a winner. The "Maker" program alone is worth the yearly subscription cost to me. (It also works fine on my TRS-80 Model 100). Any endeavor with which Terry Dettmann associates himself interests me. - Victor F Wright, Indianapolis, IN

We enjoy hearing that, Victor - thank you.

I just received your premier issue and was quite impressed, particularily with the "Forum" and imaginative listings enclosed in it. I possess (not own) an IBM-PC, with two 360K drives and a 132-column printer, but no modem. To get to the point. I can't justify the cost of your magazine for the following reasons: 1. Twenty-five dollars for a 6-issue magazine is extraordinarily high, where I am unable to download programs. 2. You present yourself as a user-oriented magazine and then subtly discourage input from those same users. Should you decide to expand to 12 issues per year and open up a little more, contact me again. I would love to be a subscriber to a "true" user-oriented magazine. PS: This is constructive criticism. - Joseph Whalen, Westbrook, ME

You may be right on all counts except one: We are all puzzled about your statement concerning "subtle discouragement" of input from our readers. We fail to see how that could possibly have been read into our first issue.

...I like and agree with your "Point of View". After putting in forty years at (The Boeing Co.) and retiring from that good place, I'm now established as a management consultant ... and writing business programs in MBASIC. Many people pooh-pooh and bad-mouth BASIC, but I find I can do just about anything I need in this very tolerant and forgiving language. As you stated, if one uses discipline in his programming with plenty of meaningful remarks, MBASIC is remarkable. True, it may run slower, because of the extra interpreter function, but it certainly gets the job done. I'm also very excited about computer programming - I think the challenge and satisfaction is beyond compare. Good luck with CodeWorks - you've made a nice start! - Bill Strang, Seattle, WA

I received the premier issue of CodeWorks the other day and was favorably impressed with the Point of View and the Forum. I think that over $2.00 an issue for 30 to 60 pages is a little steep but I realize that without advertisers the money to support a magazine must come from subscribers. I would even be in favor of supporting such a magazine if I thought that it was edited by technically knowledgeable people. However, the article on page 30, ASCII Codes, apparently written by Jay Marshall, an associate editor, does not contribute to reassuring me that such is the case. I think that a person technically knowledgable in the field of programming would know the difference between ASCII code and EBCDIC code. Disappointedly, Don Williamson, Agoura Hills, CA

Now you know why Ralph is selling cigars and day-old newspapers. You are right though, the reference to hex F0 to hex F9 should have been ASCII 48 to ASCII 57.

My first comment about CodeWorks is, "It sure does!" I like your style... A compliment that I hope your layout staff never forgets when the going gets "tight": It's great to be able to read a program listing when it's lying on the desk or in a copy holder two feet away. Please, don't reduce them. We don't have enough hands to hold a magnifying glass, a magazine and still type... It's gratifying to know there is someone else out there who still thinks BASIC has value. I don't give a fig for FORTH, can't see C, pass up Pascal, and fell asleep in Assembly, even in school. - Ed Chapman, Alexandria, VA

Have tried Pascal and found it more trouble after programming in BASIC. I appreciate your programs, Search, Maker and Extract. They assist in my endeavors as an amateur hobbyist programmer. - Raymond Eck, MD, Tigard, OR

(This is just a sample of the numerous letters we have received. Your comments and suggestions are well taken and appreciated. Thank you. - The editors.)

Halley's Comet

Track the position of the comet

©1985 Terry R Dettmann

An interesting problem with Halley's Comet approaching is just where is Halley compared to where the Earth is? The program included with this article answers that question to a fair level of accuracy.

The determination of the position of a body orbiting the sun is a problem whose solution goes back to the time of Kepler and Newton. Kepler worked out the method and Newton explained it in terms of his theory of gravitation. We still go back to Kepler and Newton to solve the problem of where such bodies are. While our methods have become more sophisticated, the procedures are still the same.

The location of a planet or comet is based on the solution of Kepler's Equation. Unfortunately, we can't simply solve it the way we do other types of equations. The only way to solve the equation known is by trial and error (a method known as "iteration".) The program is built around the need to solve Kepler's Equation.

The book *Practical Astronomy with your Calculator* was used to obtain the necessary orbital parameters and the basic equations. Given the basic information, the program to solve the equations was structured like this:

1. Initialize the program and set up parameters.
2. Input the starting date.
3. Repeat 10 times -
   a. Compute the current day number.
   b. Compute the locations of the comet, Earth and Venus.
4. Print the positions of the comet, Earth and Venus for the 10 calculated dates.

That seems pretty easy, but there are some problems to consider. Since we have a computer, we expect its answers are always correct. If we just give it all the decimal places we have available we'll get good answers. Well, that just isn't so!

Calculations of this sort are affected very much by errors in the accuracy of computations within the computer. In a small computer in BASIC, we are generally dealing with single precision numbers good for an accuracy of about 5 to 6 digits. We could introduce double precision and increase the accuracy, but this sort of calculation relies heavily on built-in functions such as trigonometric functions, powers, and so forth. These functions are generally single precision functions only.

So what, you say, single precision is good enough, isn't it? It's not. There are ways to increase the accuracy of the computations. We can compute our own functions, we can use double precision, we can be very careful about error propagation through our computations. Although it can be done, this program is not an attempt to provide that high an accuracy, but rather to show some general principles. The program includes the orbital parameters for the comet, Earth and Venus to an accuracy greater than can be used by the program. The orbital parameters are provided in their full accuracy for those who wish to revise the program. (In printing the hardcopy graphic some accuracy is lost in any case.) The program was designed to be applicable only to the 1985-86 apparition of Halley's Comet. It is best between about October 1985 and May 1986. Outside that range, factors such as leap years and other influencing factors have not been taken into account.

The program reproduces the orbit of Halley's Comet fairly well. The dates for various points along the path are reasonably accurate since they are based on orbital parameters for the approach to the sun in February 1986. The Earth and Venus are based on orbital parameters from the beginning of 1980 and show some effect of that. Both Earth and Venus appear to be too advanced along their orbits compared to more accurate charts computed with larger computers.

The program is divided into logical sections by subroutine calls. It was built from start to finish by deciding the steps necessary to solve the problem and then providing the code for them. Let's first look at initialization.

We first set aside memory for the orbital

parameters (8 parameters, 3 objects). Note that we are actually wasting some space, since 0 is a legal index in many forms of BASIC, but it is more understandable to number things from one. We also provide for the number of days in a year to the first of each month (array MD),and x and y positions for the Comet (HC), Earth (EA) and Venus (VN). The date of each position (array DT) is also provided for.

The rest of the initialization sets up the parameters such as orbital parameters (lines 140-180), days to the first of the month (190- 210), scaling factors for plotting (line 220), the number of days (ND) to compute, the Julian Date (last five digits) for 1900, degrees to radians conversion and character strings for use in displaying the orbits.

Once we've got these things set up, we can proceed to the main part of the program. We need to input the starting date for our computations and convert it to a Julian Date (subroutine 660). For each day, we get the Julian Date for the day (subroutine 390) and then compute the orbital positions for that day (subroutine 430). After we have all of the positions, we then plot the results on the printer (subroutine 580) and print out the dates corresponding to each position (subroutine 860).

In subroutine 430, we have to compute the positions of three objects, so we take them one at a time, refer to the correct orbital elements (parameter OP=1 for the Comet, 2 for the Earth and 3 for Venus). Once we have set up the parameter, we refer to a more general subroutine (990) which computes the position of a single body for a given set of orbital elements. At this level (subroutine 430) we use the returned values from subroutine 990 to record the x and y positions of the objects already scaled for the paper we will print it on.

Once we reach subroutine 990, we have a simple situation. Extract the orbital parameters, use them in the equations, and return the x and y coordinates of the object. In order to keep fairly close to the form the equations are written in, the parameters are put into variables that are close to the actual parameters used in the equations. For clarity, these parameters are:

EP - The epoch at which the object's parameters are recorded (the year in decimal years).
TP - The period of the object (time to go around the orbit once).
EX - The mean longitude of the object at the recorded epoch (relative to the Sun).
WM - The longitude of the object at perihelion (point of closest approach to the Sun).
EC - The eccentricity of the orbit (measures departure from circularity).
AX - The length of the semi-major axis in astronomical units (the distance from the Sun to the Earth).
IN - The inclination of the object to the orbit of the Earth (the plane of the ecliptic).
OM - The longitude of the object when it passes from below to above the place of the ecliptic (ascending node).

We make adjustments in these parameters to make them all consistent, and then proceed in three standard steps: solve Kepler's Equation for the number E, known as the Eccentric Anomaly. This then factors into the solution for V, the True Anomaly which is used to get the actual longitude of the object relative to the Sun. The true anomaly is the difference between the longitude at perihelion and the present position of the object. Subroutine 1050 solves Kepler's Equation using a standard procedure, subroutine 1110 solves for the true anomaly and the distance of the object from the Sun. Finally subroutine 1160 turns these numbers into the actual x and y locations of the object relative to the Sun.

With all the computations done, we finally go to subroutine 580 to plot out the numbers and subroutine 860 to print out the key for interpretation of the plot. Since we have 10 dates computed, we simply work through the 60 possible lines and see if we have anything to plot on each line (subroutine 760). If there is anything to plot, we use subroutine 830 to insert it into the line. Finally, we send the line to the printer.

The final step, printing the table of dates and locations (subroutine 860) involves converting the date for each point to a month, day and year (subroutine 920) and then printing the tabled data as a reference for interpreting the plot.

Someone who wants to take more time can improve the accuracy of the results by recoding with double precision numbers and writing special function subroutines. The program will be slower, but more accurate. ■

```
100 REM ** HALLEY'S COMET ** DO NOT DELETE REMARK LINES **
110 CLEAR 1000:REM ** DELETE THIS LINE IF YOU DON'T NEED IT **
120 PRINT CHR$(12):REM ** CHANGE TO CLS IF NECESSARY **
130 DIM OB(3,8),MD(12),EA(2,10),HC(2,10),VN(2,10),DT(10)
140 FOR I=1 TO 3: FOR J=1 TO 8:READ OB(I,J):NEXT J:NEXT I
```

```
150 REM ** ORBITAL PARAMETERS FOR THE COMET, EARTH AND VENUS **
160 DATA 1986.112,76.0081,170.0110,170.0110,0.9673,17.9435,162.2384,5
8.1540
170 DATA 1980.0,1.00004,98.833540,102.596403,0.016718,1.000000,0,0
180 DATA 1980.0,0.61521,355.73352,131.2895792,0.0067826,0.7233316,3.3
94435,76.4997524
190 REM ** DAYS TO FIRST OF THE MONTH **
200 FOR I=1 TO 12:READ MD(I):NEXT I
210 DATA 0,31,59,90,120,151,181,212,243,273,304,334
220 SX=(80/8)*2:SY=(66/11)*2
230 ND=10:JD=15020:DR=3.14159/180
240 ES$="earthorbit":HS$="HALLEYSCOM":VS$="venuspathx"
250 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
260 PRINT"          H A L L E Y' S  C O M E T  T R A C K E R
270 PRINT"                        by Terry R Dettmann
280 PRINT STRING$(60,"-")
290 REM ** MAIN COMPUTATION LOOP **
300 INPUT"STARTING DATE (mm,dd,yy)";MM,DD,YY
310 GOSUB 660:TB=T
320 DY=10
330 FOR I=1 TO ND
340 GOSUB 390:GOSUB 430
350 NEXT I
360 GOSUB 580
370 GOSUB 860
380 END
390 REM ** GET THE CURRENT DAY FOR THE COMPUTATION **
400 T=TB+(I-1)*DY
410 DT(I)=T
420 RETURN
430 REM ** COMPUTE EARTH AND COMET LOCATIONS **
440 PRINT "TIME:  ",T
450 REM ** THE COMET FIRST **
460 OP=1:GOSUB 990
470 HC(1,I)=INT(40+X*SX+.5):HC(2,I)=INT(30-Y*SY+.5)
480 PRINT "COMET: ",HC(1,I),HC(2,I)
490 REM ** EARTH NOW **
500 OP=2:GOSUB 990
510 EA(1,I)=INT(40+X*SX+.5):EA(2,I)=INT(30-Y*SY+.5)
520 PRINT "EARTH: ",EA(1,I),EA(2,I)
530 REM ** VENUS **
540 OP=3:GOSUB 990
550 VN(1,I)=INT(40+X*SX+.5):VN(2,I)=INT(30-Y*SY+.5)
560 PRINT "VENUS: ",VN(1,I),VN(2,I)
570 RETURN
580 REM ** PLOT THE POSTIONS AS GIVEN **
590 FOR Y=1 TO 45
600 IF Y<>30 THEN Y$=" " ELSE Y$="-"
610 LN$=STRING$(78,Y$):MID$(LN$,40,1)="!"
620 GOSUB 760
630 LPRINT LN$
640 NEXT Y
650 RETURN
```

```
660 REM ** DETERMINE THE FIRST JULIAN DAY NUMBER **
670 GOSUB 730
680 T=JD+INT(365.25*YY)+DN
690 RETURN
700 REM ** DETERMINE JULIAN DAY FOR EPOCH **
710 T0=JD+INT(365.25*(EP-1900))
720 RETURN
730 REM ** DETERMINE THE DAY NUMBER (DN) **
740 DN=MD(MM)+DD-1
750 RETURN
760 REM ** FIND ANY LOCATIONS TO PUT IN THIS PLACE **
770 FOR I=1 TO ND
780 IF HC(2,I)=Y THEN X$=MID$(HS$,I,1):X=HC(1,I):GOSUB 830
790 IF EA(2,I)=Y THEN X$=MID$(ES$,I,1):X=EA(1,I):GOSUB 830
800 IF VN(2,I)=Y THEN X$=MID$(VS$,I,1):X=VN(1,I):GOSUB 830
810 NEXT I
820 RETURN
830 REM ** PLACE THE CURRENT LOCATION INTO THE LINE **
840 IF X>LEN(LN$) THEN RETURN
850 MID$(LN$,X,1)=X$:RETURN
860 REM ** WRITE OUT THE FINAL DATE TABLE **
870 LPRINT"DATE","COMET","EARTH","VENUS"
880 FOR I=1 TO ND:JX=DT(I):GOSUB 920
890 LPRINT USING"##/##/##";MX,DX,YX;:LPRINT,MID$(HS$,I,1),MID$(ES$,I,
1),MID$(VS$,I,1)
900 NEXT I
910 RETURN
920 REM ** GET THE CURRENT DATE FROM JULIAN DATE **
930 CD=JX-JD-INT(365.25*YY):YX=YY
940 IF CD>365 THEN YX=YX+1:CD=CD-365
950 FOR J=1 TO 11:IF CD>=MD(J) AND CD<MD(J+1) THEN MX=J:GOTO 970
960 NEXT J:MX=12
970 DX=CD-MD(MX)+1
980 RETURN
990 REM ** COMPUTE THE X,Y COORDINATES FOR A BODY IN ORBIT **
1000 EP=OB(OP,1):TP=OB(OP,2):EX=OB(OP,3)*DR:WM=OB(OP,4)*DR
1010 EC=OB(OP,5):AX=OB(OP,6):IN=OB(OP,7)*DR:OM=OB(OP,8)*DR
1020 GOSUB 700
1030 GOSUB 1050:GOSUB 1110:GOSUB 1160
1040 RETURN
1050 REM ** SOLVE KEPLER'S EQUATION **
1060 M=DR*(360/365.242)*(T-T0)/TP+(EX-WM)
1070 AC=.001:E=M
1080 E1=M+EC*SIN(E)
1090 IF ABS(E1-E)<AC THEN RETURN
1100 E=E1:GOTO 1080
1110 REM ** RADIUS AND ANOMALY **
1120 A=SQR((1+EC)/(1-EC))*TAN(E1/2)
1130 V=2*ATN(A)
1140 R=AX*(1-EC^2)/(1+EC*COS(V))
1150 RETURN
1160 REM ** XY COORDINATES **
1170 X=R*(COS(OM)*COS(V+WM)-SIN(OM)*SIN(V+WM)*COS(IN))
1180 Y=R*(SIN(OM)*COS(V+WM)+COS(OM)*SIN(V+WM)*COS(IN))
1190 RETURN
```

DATE	COMET	EARTH	VENUS
11/ 1/85	H	e	v
11/11/85	A	a	e
11/21/85	L	r	n
12/ 1/85	L	t	u
12/11/85	E	h	s
12/21/85	Y	o	p
12/31/85	S	r	a
1/10/86	C	b	t
1/20/86	O	i	h
1/30/86	M	t	x

Sample output of Halley.BAS using 11,01,85 as input

The Sun is located at the crossed lines. Viewed from above the Sun, the Earth and Venus are in counter clockwise orbit. Halley is coming in and will go around the Sun in a clockwise fashion. The letters in the table tell you where each of the bodies is on that date. The "s" in the path of Venus is where Venus will be on the 11th of December. On that date, Earth will be at letter position "h" and Halley will be at letter "E".

In line 320 of the program, the increment for days between calculations is set at 10. You can change this to any increment you wish. Using a smaller number here will crowd the letters, while a larger number may clutter the diagram and make it harder to read.

Beginning BASIC

The PRINT statement

Back in the early days of computing, they did not have video screens as they do today. Consequently, when anything was to be output from the computer, the command "PRINT" was issued, and the printer (usually a Teletype device) printed the required data on paper.

Because the output device was an actual printer, one could tab to various positions along the line, but since the carriage only rolled the paper in one direction in those days, you could not move back up the paper and print over what was already there.

With the coming of the video screen there had to be a few changes. There was now ambiguity in the print command. Print where? To the screen or the printer? Most BASIC implementations use PRINT to mean to the screen. (Maybe the command DISPLAY would have been a better choice, but that's hindsight.) LPRINT was chosen as the command to print to the printer.

The video screen also made it possible to print at any location on the screen, even if that position was above already printed information. New commands in BASIC had to be implemented to provide for this positioning. One popular method was to use PRINT@XXXX, where XXXX was the number of spaces starting from the upper left corner and counting across the screen and continuing to the next row. Eventually, the LOCATE command was implemented, which gives row and column information at which the printing will take place. Both these commands release the cursor from its regular scanning position and move it directly to the specified spot on the screen. Printing then commences from that position.

With an 80-column by 24 line screen the cursor may be positioned in any one of 1920 different locations. It is possible to position the cursor back up the screen, where information has already been printed. The new information will simply overwrite the old.

BASIC has many forms of the PRINT command. Sometimes, you will see PRINT on a line all by itself. This provides for a linefeed, and moves the cursor to the beginning of the next line on the screen. This one acts very much like the return key on the keyboard, only the program now pushes the key instead of you doing it.

PRINT may be used to print messages on the screen. At whatever point the cursor currently resides, the command to PRINT "Hello there, Genius" will print everything between the two sets of quote marks, including the comma. Everything between the quote marks is called a literal string, which means that it is exactly what it says it is.

Next, we come to the command to PRINT A. This will display the value of the variable stored in location A (or any other variable you wish to display.) When you assign a value to a variable, as in B=25, the machine will find its own place called "B" and in that location in memory it will store the value 25. Now, when you tell it to PRINT B, it will go to that location and get the 25 and display it on the screen. Where on the screen? If you don't tell it otherwise, it will print wherever you last left the cursor.

If you want to print a line with more than one variable on it you can use PRINT A;B. This will put one character space between the values for A and B. This can be extended to PRINT A;B;C;D; etc., and if the line then becomes too long for the screen, it will wrap around.

The comma is another printing device. The expression PRINT A,B,C will print the values of A, B and C in neat columns spaced (on most machines) eight characters apart. Yes, you can mix semicolons and commas within a print line - like this: PRINT A;B;C,D.

Tabbing can be done on a computer much like that on a typewriter. The TAB key on your machine will (on most machines) tab the cursor eight spaces to the right. In a print statement however, you can tell the cursor exactly which column to tab to. For example: PRINT TAB(23);A;TAB(31);B will print the value of A at column 23 and the value of B at column 31. Tabs must always be in increasing order, i.e., you cannot tab to column 40 and then on the same line tab back to column 12. Tabs may be "hard" numbers or they may be variables. You may tab to column 38 by defining A as 38 and then saying PRINT TAB(A);X which will print the value of X at column 38. You may also compute a tab by saying PRINT TAB(A+B);X, which adds the values of A to B and tabs to that position and prints the value of X.

A variation of the print statement is the form: PRINT STRING$(60,65). This will print a string of sixty capital letters A on one line. The first number

in the parens will tell how many to print, while the second number is the ASCII value of the character to print. (ASCII 65 is capital A.) You can also designate the second number like this: PRINT STRING$(60,"T"), in which case it will print the letter T. You may ask what for? Well, for one thing it is handy in making heading lines on either the screen or on hardcopy. In most of the programs we publish you will find a line like this: PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-"). This centers the words "The CodeWorks" within a line of hyphens.

There is a variation on this command on some machines called SPACE$( ). It is used like this: PRINT SPACE$(15);X. This will print 15 spaces and then the value stored in memory location X. There is yet another command like this that does not use the string identifier. It is used like this: PRINT A;SPC(10);B and it will print the value of A, space 15 spaces and then print the value of B.

You use the print command to display the value of any variable. If your machine has TIME$ or DATE$, you can PRINT TIME$ or PRINT DATE$ to see the time or the date. In some machines PRINT MEM will show how many free bytes are available in memory. In others, PRINT FRE(0) will do the same. Try it on yours.

By the way, the print command is used so much that the people who wrote BASIC included a nice shorthand way to enter it without writing out the word PRINT. It's the question mark. If you tell the computer to ?"HELLO" or to ?A it will print HELLO or the value of A. When you use the question mark in a program line, the BASIC interpreter will change it to the actual word PRINT as soon as you list the line.

There are yet two other forms of the print statement. One is PRINT #1, which is used to print the contents of buffer #1 to the diskette. The other is PRINT USING, which is one of the most powerful commands in BASIC, and needs an entire article all by itself to be explained. We will cover both of these statements in future articles on Beginning BASIC.

Here are a few lines of code for you to try and see how your machine reacts:

```
10 A=5
20 B=10
30 PRINT A;B
40 PRINT A,B
50 PRINT TAB(5);B
60 PRINT TAB(A);B
70 PRINT STRING$(B,"*")
80 PRINT TAB(B-A);"HI";TAB(B);" BYE"
```

Try mixing the print statement variations to see what works. All of the above statements will work either in immediate (or command) mode or in a program line. If your machine supports the command LOCATE, you must locate the cursor first, and then issue the print command: LOCATE 12,35:PRINT"line 12 character position 35". If you have an 80-column screen and use PRINT@, you might say: PRINT@120,"line 2 character position 40".

PRINT and LPRINT are the most common ways the computer lets you know what it is doing. Try some of the examples. ■

Programming Notes

Since we have received many requests for "jargon explained", here are a few. CP/M stands for "Control Processor/Micro". This is one of the earlier versions of DOS (Disk Operating System). A DOS is like the prime contractor on a construction project. Everything that goes on must be coordinated with the prime contractor. In your computer, the DOS takes care of all the little chores that need to be taken care of. This way, you can concentrate on what you are doing and not worry about the details. Just for one small example: Because of DOS, you need never concern yourself with where on the disk some program will be, nor updating the directory entry for that program. DOS does it. TRSDOS is the Tandy Corp. name for their DOS. The TRS probably stands for "Tandy Radio Shack". It is not terribly unlike other DOS. MS-DOS is the Microsoft Corp. version of DOS. It is undoubtedly the most commonly used DOS. PC-DOS is MS-DOS for all practical purposes. PC is the IBM version of MS-DOS.

DOS and BASIC come in versions numbered like "Version 2.0" or "Version 3.1.2". The first number is the main version number. The second number is usually used to indicate some upgrade or fix that has been applied. We have never fully understood the third digit. The first number is all important. When it changes, it usually indicates an entire re-write of the version.

One of the many readers who asked for jargon explained was John Ross of Rome, GA. How do you pronounce DOS? It rhymes with Ross.

Calendar

For the years 1753 through 3999

Staff Project

Back in the good old days you used to get a calendar from every merchant in town. Seems they tried to outdo themselves by getting the biggest, fanciest and most colorful one to distribute free around Christmas. Of course, those were also the days when you used to get free state road maps at any gas station. How times have changed.

These days, you go to a specialty shop and pay up to ten bucks a pop for a decent calendar. Now, you can let your computer print one for you, not only for 1986, but for any year from 1753 through 3999. Sort of overkill, but it's there if you want it. You may well ask why the limits?

It turns out that September, 1752, may have been the worst month in the history of the world. At least, if it happened today, it certainly would be. Look at figure 1 to see what that abbreviated month looks like.

The quick jump from the 3rd to the 14th was to get the calendar back to "real" time. It was off by eleven days due to an error in the way it was set up originally. (It is said that in England at that time, people rioted and demanded the King give them back their eleven days!) Some of the middle-east countries didn't change their calendars until the 1800's.

So in our program, we simply assume that we don't need to see anything before 1753. On the other end, we stop at 3999 since an adjustment will need to be made in the year 4000 (and again in the year 8000). These adjustments could just as well have been programmed into this calendar program, but it adds unnecessary code and accomplishes little. For a complete description of the calendar correction problem, see references in your trusty encyclopedia under the heading of "Calendar". It provides a fascinating story of how and why the calendar of today is what it is.

Our program description starts with line 110. This is a clear statement that you may or may not need depending on your BASIC. Microsoft BASIC versions up to about 5.1.x needed string space to be cleared. After that version the clear will result in a syntax error because the string space is allocated dynamically. If you don't need the clear, then simply leave it as a remarked line or don't use that line.

Lines 120 and 130 are two different "clear screen" lines. Again, choose the one appropriate to your machine and remark (or leave out) the other. Lines 140 through 190 dimension A$ and then print the standard CodeWorks heading on the screen. Some initialization goes on from line 200 through 270. Line 210 is rather important. It is a remark line containing the numbers 0 through 9 repetitively. This line immediately preceeds a line of the day numbers which must be typed in exactly as shown. Remark line 210 gives you a reference as to the placement of the numbers in the line following it. This idea is followed again later in the program where spacing within a literal string is important. It happens again, for instance, in line 230. Here it helps you to get the correct spacing for FM$, the line following it, and for H$, the line

```
      September 1752
 S  M Tu  W Th  F  S
          1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

Figure 1

How would you liked to have paid bills in a month like this? Today, something like this would probably wreck the economy in short order.

A sample output of this program is shown on the opposite page. The fancy heading we promised is not so fancy, but it all fits on a standard 8 x 11 page. The programming to get the big numerals is interesting. See how it is done from line 720 on, then adapt it to make your own.

following that one.

A word or two about lines 240 and 250 is in order. These two lines will be used later with a PRINT USING statement. They are format strings. The backslashes you see in both lines may need to be changed to the percent sign (%) depending on the version of BASIC you use. Note that the slash is not a normal slash but a *backslash*. Some machines have this as a defined key, in others you may need to use the CTRL key (control key) with another key. Check both the PRINT USING command and your keyboard description in your BASIC manual if in doubt.

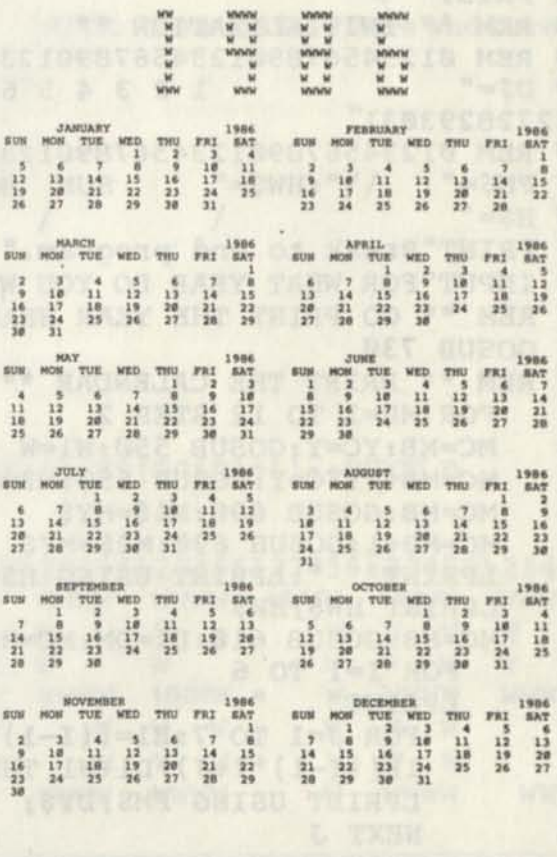
Line 270 asks which year you want a calendar for. This should be a year between 1753 and 3999. Anything else will either not be accurate, or will cause the computer to error. The input statement is not "bullet proofed", i.e., you can put in anything you want, even if it is out of the specified range. Simply be aware that any year out of range will not be accurate.

At line 290 we go to the subroutine at line 730 to print the year heading. Let's go there now. In line 730, we take the integer value of the year and change it to a string. Then, using the MID$ function we strip off the digits one by one and stuff them into array DG(I). Two little interesting things happen here. First, when we change the integer for the year into a string, we get a leading space before the year number. Second, in line 750 when we stuff the array DG(I), we stuff it with the *ASCII* value of the number. If our year, for example, is 1986, YD$ will look like "space,one,nine,eight,six". Now, in line 750 we find the first digit of the year and get its ASCII value, which happens to be ASCII 49. (We also read I+1 to get past that leading space.) To get the ASCII value of 49 back to the digit 1 we simply subtract 48 from it, and DG(1) will hold the integer 1. DG(2) will hold the 9, and so forth. Now that we know what each digit is, we will need to pick the proper numbers from the data statements in lines 880 through 940 and print them, centered, on top of our calendar page. This all happens from lines 770 through 850.

Since there are seven lines of data, we set up a loop in line 770 to read them, one at a time. We will then also pick the appropriate part of the right numbers and print them on the printer immediately before going to get the rest of what makes up the entire number. In line 780 we read the first data statement. We then initialize I6 to be 2. This eliminates the period at the beginning of each data line. The period was necessary to tell where the line actually started, but we don't want to print it. We next jump into an inner loop at line 800 and put the top portion of each number in the data statements into array Z$(I). We then jump into yet another inner (inner,inner?) loop at line 840 and print the top portion of the year numbers on the printer. The program then goes back to line 770 and gets the next line to print. In line 840, the TAB(D*10+15) centers the big type year on the printer page, and spaces properly between the digits of the year. Line 860 simply puts two vertical spaces between the big type of the year and the beginning of the calendar portion.

We now return to line 300 and begin to print the calendar itself. Variable MB is the mainline loop counter. It will cause twelve months to be printed, two across, six times. Within the mainline MB loop we have another I loop which steps from 1 to 6, there being two months in each step. Within the I loop there are two other loops, both called J loops (no inteference here, they work sequentially and do not conflict with each other).

Back in lines 320 to 350 we did some subroutine calls to find out what months will be printed and how many days there were in those months. In line 360 we then use the format line H$ to print the month name and year, then space and print the next month and year. This is followed immediately by line 370, which prints the format line from line 240 (the second part of the line, containing HW$,



which is the week headings). It prints this twice on the same horizontal line, with the space at the beginning of the format line being the space between the two months.

In line 380 we go to the subroutine at line 610 to find the number of days in the month. While there, we also find out if we are dealing with a leap year, and make the proper exceptions for years divisible by 100 and 400. Upon returning from the subroutine, we go to the two inner J loops and print the numbers for the days of the week using the formatted D$ from line 220 and FM$ from line 240. This continues until the MB loop is exhausted, at which time the calendar is printed and done.

Note line 680, which is another "calibration" line to help you type in the next lines correctly. If you don't get line 690 right, you will find pieces of month names scattered around on the page. Also, in line 690, you need to use the linefeed (CRTL-J) after the quote after May and October. Use the RETURN/ENTER key only after the quotes after December.

You can make the big type numerals in the data statements anything you like. We just chose a fat looking character like the W or the M to make them stand out. You could though, make the zero out of zeros, the one out of 1's, and so forth. You can also change the shape of the characters if you like. But you need to keep the same spacing (unless you want to modify program lines 770 through 850 too). We stuck this at the end of the program so you could expand on the heading art in any way you wish. From line 720 on, it's all yours. If you want to create pin-up art and make it a whole page long by itself, you can. If you do though, please send us a copy so we can print it and show everyone. You might even make extra copies and sell them through your local gas station. ■

```
100 REM ** CAL.BAS * WRITTEN FOR CODEWORKS MAGAZINE **
110 'CLEAR 10000:REM INCLUDE THIS ONLY IF YOU NEED IT.
120 PRINT CHR$(12):REM UN-REMARK APPROPRIATE LINE FOR CLEAR SCREEN
130 'CLS:REM UN-REMARK APPROPRIATE LINE FOR CLEAR SCREEN
140 DIM A$(15)
150 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
160 PRINT"                C A L E N D A R  P R O G R A M"
170 PRINT"          prints a calendar for any year from 1753 to 3999"
180 PRINT STRING$(60,"-")
190 PRINT
200 REM ** INITIALIZATION **
210 REM 0123456789012345678901234567890123456789012345678901234567890
220 D$="              1 2 3 4 5 6 7 8 9101112131415161718192021222324252627282930 31"
230 REM 0123456789012345678901234567890123456789012345678901234567890
240 FM$="   \\":HW$="     SUN  MON  TUE  WED  THU  FRI  SAT"
250 H$="              \       \              ####"
260 PRINT"Break to end program."
270 INPUT"FOR WHAT YEAR DO YOU WANT THE CALENDAR";Y
280 REM ** GO PRINT THE YEAR HEADING **
290 GOSUB 730
300 REM ** PRINT THE CALENDAR **
310   FOR MB=1 TO 12 STEP 2
320   MC=MB:YC=Y:GOSUB 550:W1=W
330   MC=MB+1:YC=Y:GOSUB 550:W2=W
340   MC=MB:GOSUB 690:M1$=MY$
350   MC=MB+1:GOSUB 690:M2$=MY$
360   LPRINT"  ":LPRINT USING H$;M1$;Y;M2$;Y
370   LPRINT HW$;HW$
380   MC=MB:GOSUB 610:D1=DM:MC=MB+1:GOSUB 610:D2=DM
390     FOR I=1 TO 6
400     LPRINT"   ";
410       FOR J=1 TO 7:E1=((I-1)*7+J+6-W1)*2-1
420       IF((I-1)*7+J)>D1+W1 THEN DY$="  "ELSE DY$=MID$(D$,E1,2)
430       LPRINT USING FM$;DY$;
440       NEXT J
```

```
450      LPRINT"    ";
460        FOR J=1 TO 7:E2=((I-1)*7+J+6-W2)*2-1
470        IF((I-1)*7+J)>D2+W2 THEN DY$="  "ELSE DY$=MID$(D$,E2,2)
480        LPRINT USING FM$;DY$;
490        NEXT J
500      LPRINT"  "
510      NEXT I
520    NEXT MB
530 RUN 120
540 REM ** DAY OF THE WEEK ROUTINE **
550 IF MC>2 THEN 560 ELSE MC=MC+12:YC=YC-1
560 W=1+2*MC+INT(.6*(MC+1))+YC+INT(YC/4)-INT(YC/100)+INT(YC/400)+2
570 W=W-INT(W/7)*7
580 W=W+6:W=W-INT(W/7)*7
590 RETURN
600 REM ** NUMBER OF DAYS IN THE MONTH **
610 IF MC<>2 THEN 630 ELSE LP=0
620 IF(Y-INT(Y/4)*4)=0 THEN IF((Y-INT(Y/100)*100)=0)AND((Y-INT(Y/400)
)<>0) THEN LP=0 ELSE LP=1
630 M$="312831303130313130313031"
640 DM=VAL(MID$(M$,2*MC-1,2))
650 IF MC=2 THEN DM=DM+LP
660 RETURN
670 REM ** MONTH OF THE YEAR **
680 REM 012345678901234567890123456789012345678901234567890
690 MN$="JANUARY  FEBRUARY MARCH    APRIL    MAY        "
+"JUNE     JULY     AUGUST   SEPTEMBER OCTOBER  "
+"NOVEMBER DECEMBER "
700 MY$=MID$(MN$,(MC-1)*9+1,9):RETURN
710 END
720 REM ** PRINT YEAR HEADING ROUTINE **
730 YD$=STR$(Y)
740    FOR I=1 TO 4
750    DG(I)=ASC(MID$(YD$,I+1,1))-48
760    NEXT I
770    FOR L=1 TO 7
780    READ ST$
790    I6=2
800      FOR I=1 TO 10
810      Z$(I)=MID$(ST$,I6,5)
820      I6=I6+6
830      NEXT I
840        FOR D=1 TO 4:LPRINT TAB(D*10+15);Z$(DG(D)+1);:NEXT D
850    NEXT L
860 LPRINT" ":LPRINT" "
870 REM* 123456123456123456123456123456123456123456123456123456123456
880 DATA . WW    WW   WWW   WWWW     W  WWWW  WWWW  WWWW  WWWW  WWWW
890 DATA .W  W    W       W     W    W  W       W          W  W  W  W  W
900 DATA .W  W    W      W      WW   W W  W       W          W  W  W  W  W
910 DATA .W  W    W     W       WWW  WWWW  WWWW  WWWW       W  WWWW  WWWW
920 DATA .W  W    W    W         WW     W     W  W  W       W  W  W     W
930 DATA .W  W    W    W          W     W     W  W  W       W  W  W     W
940 DATA . WW    WWW  WWWW  WWWW     W  WWWW  WWWW       W  WWWW   WWW
950 RETURN
```

NOTE: Use a CRTL-J here and here.

Piles to Files

William L Norris, Edmonds, WA

Several months ago I became very unhappy about the piles of papers and disks which represented my file system. So I set out to design a new one. The project required extensive study of files and programming.

One of the toughest nuts to crack was the Direct Access File. I have since accumulated information on many subjects, but have found it was more fun playing with the system than to use it. Since the whole project was for my own amusement, I was not under pressure to follow conventional programming procedures. By avoiding excessive GOTO's and GOSUB's, and by repeating a few line here and there, I was able to make a program which worked, and at the same time is easily modified.

I find sometimes that, following conventions, it is easy to take a simple program and make it completely useless in less than a hundred lines. By following my own free-form BASIC, I avoided a great deal of stress.

Direct Access Files are great - they can be as long as the disk will hold, but for Word Processors, or Modems, not good. Sequential files are also great, but they can only be as long as RAM holds out. Why not, then, convert them form one form to the other when needed? I am hoping to stir up enough interest or controversy with this brief article to hear from some other amateur programmers on this subject. If not, I will let it die a quiet death. Since I am over seventy, and no longer work for hire, I can take the advice of Lao-tse, and stop walking around "beating a drum."

(Two of Mr. Norris' programs, combining sequential files and converting sequential to direct access, are included in this issue. Direct access to sequential, combining two direct files and converting sequential to direct access and appending to a direct access file will be presented in the next issue - Ed.) ■

```
100 PRINT CHR$(12)
110 CLEAR 25000
120 DIM A$(300)
130 PRINT"COMBINE TWO SEQ FILES INTO A THIRD"
140 INPUT"NAME OF 1ST SEQ FILE";F1$
150 INPUT"NAME OF 2ND SEQ FILE";F2$
160 INPUT"NAME OF COMBINATION FILE";F3$
170 OPEN "I",1,F1$
180 OPEN "I",2,F2$
190 OPEN "O",3,F3$
200 IF EOF(1) THEN GOTO 240
210 X1=X1+1
220 LINE INPUT #1,A$(X1)
230 GOTO 200
240 CLOSE 1
250 IF EOF(2) THEN GOTO 290
260 X1=X1+1
270 LINE INPUT #2,A$(X1)
280 GOTO 250
290 CLOSE 2
300 L=X1
310 FOR X=1 TO L
320 PRINT #3,A$(X)
330 PRINT X;A$(X)
340 NEXT X
350 CLOSE 3
360 PRINT"FILE "; F3$" HAS "L" LINES"
370 END
```

```
100 PRINT CHR$(12)
110 CLEAR 25000
120 DIM A$(350)
130 PRINT"TO TRANSFER SEQUENTIAL FILE TO DIRECT ACCESS FILE"
140 INPUT"NAME OF SEQUENTIAL FILE ";SF$
150 OPEN "I",1,SF$
160 IF EOF(1) THEN GOTO 200
170 X=X+1
180 INPUT #1, A$(X)
190 GOTO 160
200 CLOSE
210 N=X:PRINT"L=";N
220 INPUT"NAME OF DIRECT ACCESS FILE ";DF$
230 INPUT"LENGTH OF LINES (LRL)";L
240 OPEN "R",2,DF$
250 FOR X=1 TO N
260 FIELD 2, L AS WD$
270 LSET WD$=A$(X)
280 PRINT X;
290 PRINT WD$
300 PUT 2,X
310 NEXT X
320 CLOSE
330 END
```

Programming Notes

Projects which require the development of numerous versions of many modules have a tendency to tax even the most creative programmers for meaningful names. Have you ever spent hours updating an *old* version of a program? The method suggested here works good enough to have achieved the distinctive status of "no big deal". It involves suffixing part of the date to the file name. You should occasionally purge old versions. In BASIC, for a program named SORT, it looks like this:

```
100 ..(1st  line of pgm, goto somewhere)
110 PG$="SORTdt.BAS"
120 MID$(PG$,5,2)=MID$(DATE$,7,2)
130 CLS:PRINT"Backup ";PG$
140 SAVE PG$
150 GOTO 100
160 ..(the rest of the pgm)
```

This will plug the day of the month over the "dt" in "SORTdt.BAS". Simply GOTO 110 when you want to save the program. You may not remember what changes were made to any particular program, but you can usually remember what changes you made last Friday.

When typing in PRINT lines with spaces between the quotes it is easy to see how many spaces are required by looking either up to the previous line or down to the next, where there are character spaces you can count.

Both UNIX and MS-DOS have a sort utility. Both let you start sorting on a particular column. If your text contains tab columns though, neither will sort properly (on other than the first column) because the tab is seen by the system as one character, not the actual number of spaces to the next column. Seems the only way around this is to use the space bar to the next column, which is a bit awkward.

A quick and easy way to maintain lists that can be edited is to use BASIC itself. Simply type in a line number followed by the apostrophe (shorthand for REMARK) and then enter lines of text. You can then save the text like a BASIC program, call it back and edit it using the normal BASIC editor or print it out to the printer. Naturally, you cannot run it, but it's a quick way to keep notes that can easily be updated and edited.

CARD.BAS

A mini-database using sequential files

Staff Project

The problem which prompted the creation of this program came from a small school administration office. It seems the secretary had a metal box full of 3 X 5 cards with information on students. She also had access to a personal computer, but didn't want to install a commercial data base program. Using a ready made program was too much like learning another language, she said. All she wanted was to get her cards in the computer, primarily so that they could be searched or sorted faster and she could be relieved of typing names and addresses every time notices needed to be sent. The program needed to be versatile, but easy to operate, and with as little jargon as possible. No file names or any of that, she said. Volunteer helpers should be able to use it with little or no tutoring.

Well, that sure sounded like an interesting challenge, and so CARD.BAS came into being. The program was written using sequential files. All fields are defined within the program, and all fields are alphanumeric. Any or all fields may be as long as the allowable record length (usually 256) of your computer. The operator need not know anything about the file names. To keep from clobbering a file, the program automatically loads the file into memory as soon as you run it. If you exit properly through the menu selection, the file is automatically saved. Sorting and searching may be done on any of the fields. Chain editing and searching may be done without going back to the main menu after each search or edit.

Anyone with minimum skills in BASIC should be able to add or remove fields and format the report generator at the end of the program. The field designators I$ through P$ were purposely not used so that you can use them to extend the number of fields if you want to. The number of records the program can handle varies, of course, with both the amount of free memory you have in your computer and the number and length of your fields. With 31,000 free bytes in memory, we found we could enter about 200 records with each record containing an average of 100 bytes. You can't use it all because space is needed when searching and sorting. The maximum number of records is set by you in line 110. Leave a little breathing space for the file.

With a few minor changes, this program worked on just about all the machines we tried. Here are some changes that may be required. Change line 150 to: 150 CLS:RETURN if your computer uses it instead of the CHR$(12) there now. In lines 160 and 330, change the CARD.DAT to CARD/DAT if your machine requires the slashes. In line 850, change the "FRE(0)" to "MEM" if your machine uses MEM to show memory available. Additionally, if your BASIC does not clear string space dynamically, you will need to add line 115: CLEAR ##### (here you clear as much memory for strings as your machine will let you clear.) Note that if you must add line 115 you should also remove line 850 since it will be meaningless.

The first thing that happens when you run the program is that the maximum number of records is set. Next, the field variables are set to that number. Notice that P in line 120 is not a string like the others. Variable P stands for "pointer", and will contain the integer of the loop counter as we read through the records. Later, when we sort the records, we will *look* at the string data to see which is larger, but we will *switch* only the pointer P. That way, we don't move strings around (which gets messy) and the sort is considerably faster. To say it more concisely: We leave the strings where they are in memory but we rearrange the pointers to those strings and then read the data out by using the pointers as guides to the strings. To continue, we have issued the command to run and the dimensions are set. The next thing which occurs is a GOSUB to line 330. Lines 330 through 500 are the subroutine which reads data from the disk.

The disk read subroutine is rather straightforward. It is a loop which is set to read more than we could possibly need which jumps out

when the data sentinel is reached. If, for some reason, we miss the data sentinel (ZZZ), then we exit by testing for end of file (EOF). In line 340 we have set L1 equal to 0. After the data is all read in, we will set L1 equal to the number of items read in plus one. It doesn't say plus one there, but the I counter will be at the number of records plus one at that point. We will need L1 when we add records to the file. Then we can start at L1 and go on. The loop counter I can and will be used for other things as we go along, but L1 will be unique and will always indicate where to start adding to the file. Note that A$(I) through H$(I) will all have the same I number within one record. For that record, we also make P(I) that same number for that record (in line 460). Or, all fields within a record and the pointer too, have the same I number.

No mystery yet, and no magic either. But where did that sentinel (ZZZ) come from? Let's digress a bit. When you run the program for the very first time, it will error with "file not found" because no data file exists on disk. So, the very first time you run the program you do not simply run it, you tell it to RUN 570. This will give you the main menu. From that menu, you select option 1 to add to the file. As soon as you get to the add portion, simply press RETURN (or ENTER on some machines) and you will be back at the main menu. Now select option 6, to save the file and end. At this point, the program will open the file on disk, and write out one record, the sentinel ZZZ. From here on in then, you can simply load the program and run it. Now let's get back to the normal program flow.

After returning from the read file subroutine, the program sends us to line 570. At that line, the field names are initialized. This is the only place in the program where they are defined and here is where you can change the names to suit your own needs. You can, for example, put city and state in separate fields if you like. Or, you can rename them all to Part #, Quantity, Stock Level, Reorder, etc. After field initialization, the standard CodeWorks head is displayed with the six-item main menu. Note that an ON GOTO statement is used to select the menu choice. If the number you choose is out of the range allowable, line 800 simply takes you back and redisplays the main menu. Throughout the program you will see lines that say GOSUB 150. This is the one-line clear screen subroutine.

Now let's take a look at the "add a record" option. It starts at 810 and ends at 990. First, we clear the screen, then drop down a line and start an input loop that starts at L1 (the end of our current file) and goes to V+1 (the maximum number of records we can hold plus 1). On the screen, in line 840, we print information on how to get out of the input loop (simply press RETURN on the first field), how many free characters you have remaining in memory (line 850), how many records are currently in the file (line 860), and, if the file contains more than two records, the name of the last record entered. The last name entered is handy when you are entering names and the phone rings. It lets you know where you are. The reason it does not show if the file only contains one record is that A$(I-1) will give an error if I is equal to 1 or less (see line 870). In line 890 we check to see if A$ (the first field) is a null string, and if it is we make that record the new sentinel (ZZZ), update L1 to equal the current I, and go back to the main menu. If it is not a null string we check to see if the number of records has become equal to our maximum allowable (V). If it is, we print a sorry message, set the sentinel on that record, update L1 and go back to the menu. If it isn't, we continue adding data to the fields in our record. After each record is entered, the screen clears, updated information appears at the top of the screen and we can add more.

Let's leave option 2 for now and look at it with option 4 later since option 2 uses some of option 4's code. Let's take option 3 next. The code for option 3 sits at lines 1000 through 1040. First, we clear the screen. Then we simply read through the entire file of records and print them on the screen according to the format statement in line 1020. You can't do anything with the data you see on the screen; it's more of a reassurance feature than anything else. It lets you know that you do, indeed, have data in the file and that it hasn't slid down your power cord and is piling up under your house somewhere. Besides that, it's cheap - only took five lines of code.

Now let's look at Search/Edit/Delete. Before you can edit or delete, you really need to find the record, so at line 1070 we clear the screen, print a

heading and then set a search flag called F3, to 1. We are going to use some of the code the sort feature uses, and the flag is necessary to let that code know if we are searching or sorting. We set the flag in line 1100 and then go to a subroutine at 1600. In 1600, we print the field headings on the screen and ask which item to search or sort on. In lines 1650 through 1750 we make our choice of field equal to Q$. Let's say we decided to search on F$(I), now Q$(I) equals F$(I). This way, when we search or sort we will always be operating on Q$(I), which simplifies things somewhat. After this little operation, we check the flag in line 1760. Since we set it before coming here, it is set to 1, and so we reset it to zero and return to line 1110. At line 1110, we enter the search string (or a part of it) and call it Z$. Let's digress again for a bit. The more specific you make your search string, the faster you will find the record you are looking for because it will go directly to it. "son", for example will get you Olson, Johnson and Peterson, but "Olson, Eric Q" will get you directly to that record (unless there is more than one Eric Q Olson in the file). If you are consistent in the way you enter data in the first place, you can do some very interesting searches. Back to the code...

We are now at line 1140, and have set A to 0. Why? Because we are going to read our file from beginning to end, searching using our search string. Let's say you are looking for Olson and there are two or three of them in the file. If we don't use A in our loop, you could never get to the second or third Olson. This way, when we find the first Olson and he is not the one we want, we can tell the program to keep looking and A will mark that point so we keep looking from that point on instead of starting all over. In short - using A in the loop lets us look through the file, find some record, make adjustments to it and keep on looking further through the file. Line 1160 needs some explanation. The INSTR function as used here is one of those true/false deals. What we are doing here is comparing our search string (Z$) with each Q$(I) to find a match. The way it works is like this: The INSTR function returns a logical zero as long as *no match* is found. In our case, if no match is found, the program simply falls through to the NEXT I in line 1170 and we look at the next Q$(I). If no match was found in the entire file, we fall through again to line 1180 and print that fact on the screen and then go back to the main menu.

If a match is found, however, then line 1160 sends us to line 1220, where we print the record out on the screen along with some options as to what to do with that record now that we have found it.

The sub-menu contains options to continue looking (it's not the right Olson), to do a totally different search (we picked the wrong field to search), to edit or delete the record, and to return to the main menu. In line 1310 we had adjusted variable A to be I+1. This will cause the search to continue after the record we have just found. Selecting a different search will send us back all the way to line 1090, where we set a different field equal to Q$(I) and use the flag again. The record and the sub-menu are both on the screen together. If we choose to edit the record, the program takes us to line 1410, where we are asked what item to edit. The correct information is put into temporary U$ in line 1430 and exchanged with the incorrect information in lines 1450 through 1520. In line 1530 we make A equal to I minus 1 and then go to 1230 where the updated record is displayed showing the effect of our edit. The minus 1 was necessary because we had already advanced the I counter in line 1310 in case we wanted to continue looking. In this case, we didn't want to continue looking, so moving the counter back one will show us the record we have just edited. Multiple edits can be made this way without going back to the main menu each time.

The delete option is very simple. Lines 1550 through 1570 do it. In 1550 we simply set the first field of the record to be a null string. That's it. The rest of the record remains intact and remains in memory. When we save the file and end the session, the code way back in line 180 will check to

"SEE--I FOLDED, SPINDLED, AND MUTILATED THIS COMPUTER CARD-AND NOTHING HAPPENED!"

see that null string and will not print it to the disk. At that point, it will be gone forever, but until you save the file and end you have the possibility of resurrecting a dead record. Simply search for it on the second or higher field, edit in a name in the first field and it will be a valid, live record again. We would like to say that was planned - but it wasn't - it just worked out that way.

Having taken care of the sub-menu, we can now return to the remaining options in the main menu, sort and print. We'll take the sort option first. It starts at line 1580 and ends at 1990. We clear the screen first and then print the field headings and ask which one to sort on. This is the same section of code we used in the search option, but this time flag F3 will not be set. Whichever field we choose will be made equal to Q$, so that Q$ will always be the string we compare for the sort. Flag F3 is not set this time, so we go right around line 1760 into the sort routine. The sort is a modified Shell-Metzner sort. Lines 1870 and 1880 are where the actual comparison and switching takes place. In line 1870 we compare each item in the list to be sorted with the next item in the list. If it is smaller or equal to the next item, we jump around the switching code and continue with the comparisons. If the item we are comparing is larger than the one following it we switch the pointers to those items in line 1880. Note that the strings themselves are not moved, only the pointers to them are. Moving strings is slower, and fills memory with the temporary strings which would need to be created. When memory fills with these temporary strings, BASIC suspends all other operations and goes through memory clearing out these extra strings. This takes time and is generally referred to as "garbage collection". Integers do not take up this string space, and since the pointers we are switching are integers, the sort is considerably faster.

The print options were intentionally left at the end of the program so that you can extend or modify them to suit your individual needs. In lines 2090 through 2120 you can format a report to fit your desires. The label routine, as it stands, will print standard sticky labels. Either of these two routines can be changed to print on tractor-fed card stock, or you may want to make labels for parts bins or whatever. Such formatting is left to you. Keep in mind that the fields are all string fields, and if you want to do totals in some field you can, but you will need to convert the string to an integer using the VAL function.

Use your imagination on the print section. You could use this program as a small inventory program. We used it to summarize a check book for a year with over 450 entries. We changed the field headings to date, check number, what for and amount. We then sorted the file on the "what for" field and printed out a report that collected all similar "what fors" and printed a sub-total for that item, as well as a grand total at the end of the report. That way, the 450 checks were neatly summarized on one standard page. The accountant loved it.

The Save files and end routine is at 510 through 560. Within those lines we go to a subroutine at 160 through 320. The subroutine simply opens the file for output and prints the records to the disk. Two checks are made on the data as it goes out. The first at line 180 checks to see if A$(I) is a null string. If it is then that record is simply ignored and the code goes on to the next item in the list. The second check is made at line 280, where we check for the sentinel ZZZ, which says that is the end of the file. If it is, we close the file and return to line 520, which clears the screen and tells us the file is saved and how many records it contains. At this point, the program ends, giving us the BASIC ready prompt.

A couple of operational hints may be in order. If you are approaching the limit of your memory and have several deleted records, it is wise to save the file and end. This clears out the deleted records and gives you more space. Also, it may be prudent to save the file and end occasionally when doing repeated sorts and searches. The conversion of the field strings to Q$ during sorts and searches apparently clutters memory with extra temporary strings.

Incidentally, line numbers ending with 5 are remark lines and need not be typed in. This was a fun program to write, and it could have been done in many different ways. The way we did it does not necessarily represent the most efficient or best way to do it, just one way. The bottom line is that it does the job required of it. ■

```
100 REM ** THIS IS CARD.BAS * WRITTEN FOR CODEWORKS MAGAZINE **
110 V=200:REM SET LIMIT OF # OF RECORDS HERE
120 DIM A$(V),B$(V),C$(V),D$(V),E$(V),F$(V),G$(V),H$(V),Q$(V),P(V)
130 GOSUB 330
140 GOTO 570
150 PRINT CHR$(12):RETURN: REM ** CLEAR SCREEN SUBROUTINE **
155 REM *** WRITE TO DISK ROUTINE ***
```

```
160 OPEN "O",1,"CARD.DAT"
170 I=0:J=0
180 IF A$(P(I))="" THEN GOTO 290
190 PRINT #1,A$(P(I))
200 PRINT #1,B$(P(I))
210 PRINT #1,C$(P(I))
220 PRINT #1,D$(P(I))
230 PRINT #1,E$(P(I))
240 PRINT #1,F$(P(I))
250 PRINT #1,G$(P(I))
260 PRINT #1,H$(P(I))
270 J=J+1
280 IF A$(I)="ZZZ" THEN GOTO 310
290 I=I+1
300 GOTO 180
310 CLOSE 1
320 RETURN
325 REM *** READ FROM DISK ROUTINE ***
330 OPEN "I",1,"CARD.DAT"
340 L1=0
350 FOR I=0 TO 1000
360 LINE INPUT#1,A$(I)
370 IF A$(I)="ZZZ" THEN P(I)=I:GOTO 480
380 IF EOF(1) THEN GOTO 480
390 LINE INPUT#1,B$(I)
400 LINE INPUT#1,C$(I)
410 LINE INPUT#1,D$(I)
420 LINE INPUT#1,E$(I)
430 LINE INPUT#1,F$(I)
440 LINE INPUT#1,G$(I)
450 LINE INPUT#1,H$(I)
460 P(I)=I
470 NEXT I
480 L1=I
490 CLOSE 1
500 RETURN
505 REM *** END SESSION ROUTINE ***
510 GOSUB 160
520 GOSUB 150
530 PRINT "FILE IS SAVED, SESSION ENDED."
540 PRINT
550 PRINT"THE FILE NOW CONTAINS <";J-1;"> RECORDS"
560 END
565 REM *** INITIALIZATION ROUTINE ***
570 A$="1-Name    :"
580 B$="2-Address :"
590 C$="3-City St.:"
600 D$="4-Zip code:"
610 E$="5-Item    :"
620 F$="6-Item    :"
630 G$="7-Item    :"
640 H$="8-Item    :"
645 REM *** DISPLAY MENU ROUTINE ***
650 GOSUB 150
660 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
```

"I'd like some unsophisticated software"

```
670 PRINT"              C A R D   F I L E   P R O G R A M"
680 PRINT"                an in-memory replacement for 3x5 cards"
690 PRINT STRING$(60,"-")
700 PRINT
710 PRINT "1 - ADD a new record"
720 PRINT "2 - SEARCH/EDIT/DELETE a record"
730 PRINT "3 - QUICK SCAN all records"
740 PRINT "4 - SORT records"
750 PRINT "5 - PRINT list or labels"
760 PRINT "6 - SAVE files and END"
770 PRINT
780 INPUT "NUMBER OF YOUR CHOICE";X
790 ON X GOTO 810,1070,1000,1580,2000,510
800 GOTO 650
805 REM *** ADD A NEW RECORD ROUTINE ***
810 GOSUB 150
820 PRINT
830 FOR I=L1 TO V+1
840 PRINT "To QUIT adding records press RETURN only for name entry."
850 PRINT" You have "FRE(0)" characters available in memory."
860 PRINT" The file now contains <";I;"> records."
870 IF I=>2 THEN PRINT" The last name entered was: ";A$(I-1):PRINT
880 PRINT A$;:LINE INPUT A$(I):P(I)=I
890 IF A$(P(I))="" THEN A$(P(I))="ZZZ":L1=I:GOTO 650
900 IF I=>V THEN PRINT"SORRY - FILE IS FULL - PRESS RETURN FOR MENU":
A$(V)="ZZZ":L1=I:INPUT X:GOTO 650
910 PRINT B$;:LINE INPUT B$(I)
920 PRINT C$;:LINE INPUT C$(I)
930 PRINT D$;:LINE INPUT D$(I)
940 PRINT E$;:LINE INPUT E$(I)
950 PRINT F$;:LINE INPUT F$(I)
960 PRINT G$;:LINE INPUT G$(I)
970 PRINT H$;:LINE INPUT H$(I)
980 GOSUB 150
990 NEXT I
995 REM *** SCAN THE FILE ROUTINE ***
1000 GOSUB 150
1010 FOR I=0 TO L1
1020 PRINT "---> ";A$(P(I));" / ";B$(P(I));" / ";C$(P(I));" / ";D$(P(
I));" / ";E$(P(I));" / ";F$(P(I));" / ";G$(P(I));" / ";H$(P(I))
1030 PRINT
1040 NEXT I
1050 INPUT"PRESS RETURN FOR MENU";X
1060 GOTO 650
1065 REM *** SEARCH/EDIT/DELETE ROUTINE ***
1070 GOSUB 150
1080 PRINT" ** SEARCH/EDIT/DELETE A RECORD **"
1090 PRINT
1100 F3=1:GOSUB 1600
1110 PRINT "Enter the item (or part of item) you wish to find."
1120 LINE INPUT Z$
1130 GOSUB 150
1140 A=0
1150 FOR I=A TO L1
1160 IF INSTR(Q$(I),Z$)<> 0 THEN GOTO 1220
1170 NEXT I
```

```
1180 PRINT"<<< NO MATCHING RECORD WAS FOUND >>>"
1190 PRINT" PRESS RETURN FOR MENU"
1200 INPUT X
1210 GOTO 650
1220 PRINT
1230 PRINT A$;A$(I)
1240 PRINT B$;B$(I)
1250 PRINT C$;C$(I)
1260 PRINT D$;D$(I)
1270 PRINT E$;E$(I)
1280 PRINT F$;F$(I)
1290 PRINT G$;G$(I)
1300 PRINT H$;H$(I)
1310 A=I+1
1320 PRINT
1330 PRINT" ENTER 1 - to continue looking"
1340 PRINT" ENTER 2 - for a different search"
1350 PRINT" ENTER 3 - to EDIT this record"
1360 PRINT" ENTER 4 - to DELETE this record"
1370 PRINT" ENTER 5 - to RETURN to main menu"
1380 INPUT X
1390 ON X GOTO 1150,1090,1410,1550,650
1400 GOTO 1320
1410 INPUT" WHICH ITEM # DO YOU WISH TO EDIT";X
1420 IF X=>9 OR X=<0 THEN GOTO 1410
1430 LINE INPUT" ENTER CORRECT INFORMATION ";U$
1440 ON X GOTO 1450,1460,1470,1480,1490,1500,1510,1520
1450 A$(I)=U$:GOTO 1530
1460 B$(I)=U$:GOTO 1530
1470 C$(I)=U$:GOTO 1530
1480 D$(I)=U$:GOTO 1530
1490 E$(I)=U$:GOTO 1530
1500 F$(I)=U$:GOTO 1530
1510 G$(I)=U$:GOTO 1530
1520 H$(I)=U$:GOTO 1530
1530 A=I-1
1540 GOTO 1230
1545 REM *** DELETE A RECORD ROUTINE ***
1550 A$(I)=""
1560 PRINT"RECORD IS DELETED"
1570 GOTO 1320
1575 REM *** SORT THE FILE ROUTINE ***
1580 GOSUB 150
1590 PRINT"** SORT THE FILE **"
1600 PRINT A$;B$;C$;D$
1610 PRINT E$;F$;G$;H$
1620 PRINT
1630 INPUT"WHICH ITEM # DO YOU WISH TO SEARCH OR SORT ON";X
1640 IF X=>9 OR X=<0 THEN GOTO 1620
1650 FOR I=0 TO L1
1660 ON X GOTO 1670,1680,1690,1700,1710,1720,1730,1740
1670 Q$(P(I))=A$(P(I)):GOTO 1750
1680 Q$(P(I))=B$(P(I)):GOTO 1750
1690 Q$(P(I))=C$(P(I)):GOTO 1750
1700 Q$(P(I))=D$(P(I)):GOTO 1750
```

```
1710 Q$(P(I))=E$(P(I)):GOTO 1750
1720 Q$(P(I))=F$(P(I)):GOTO 1750
1730 Q$(P(I))=G$(P(I)):GOTO 1750
1740 Q$(P(I))=H$(P(I)):GOTO 1750
1750 NEXT I
1760 IF F3=1 THEN F3=0:RETURN
1770 PRINT "Wait while I sort....."
1780 F=0
1790 N=L1-1
1800 M=N
1810 M=INT(M/2)
1820 IF M=0 THEN GOTO 1960
1830 J=0
1840 K=N-M
1850 I=J
1860 L=I+M
1870 IF Q$(P(I))=<Q$(P(L)) THEN GOTO 1930
1880 T=P(I):P(I)=P(L):P(L)=T
1890 F=1
1900 I=I-M
1910 IF I<1 THEN GOTO 1930
1920 GOTO 1860
1930 J=J+1
1940 IF J>K THEN GOTO 1810
1950 GOTO 1850
1960 IF F=1 THEN GOTO 1780
1970 PRINT "SORTED - PRESS RETURN"
1980 INPUT X
1990 GOTO 650
1995 REM *** PRINT LIST OR LABELS ROUTINE ***
2000 GOSUB 150
2010 PRINT
2020 PRINT "ENTER 1 - TO PRINT THE ENTIRE FILE"
2030 PRINT "ENTER 2 - TO PRINT LABELS"
2040 PRINT
2050 INPUT "NUMBER OF YOUR CHOICE";X
2060 ON X GOTO 2080,2170
2070 GOTO 2010
2080 FOR I=0 TO L1-1
2090 LPRINT A$;A$(P(I)),B$;B$(P(I))
2100 LPRINT C$;C$(P(I)),D$;D$(P(I))
2110 LPRINT E$;E$(P(I)),F$;F$(P(I))
2120 LPRINT G$;G$(P(I)),H$;H$(P(I))
2130 LPRINT " "
2140 NEXT I
2150 LPRINT "TOTAL NUMBER OF RECORDS PRINTED ";I
2160 GOTO 650
2165 REM *** PRINT LABEL ROUTINE ***
2170 FOR I=0 TO L1-1
2180 LPRINT A$(P(I))
2190 LPRINT B$(P(I))
2200 LPRINT C$(P(I));TAB(20);D$(P(I))
2210 FOR J=1 TO 3:LPRINT" ":NEXT J
2220 NEXT I
2230 LPRINT"TOTAL NUMBER LABELS PRINTED"; I
2240 GOTO 650
```

Puzzler #1

How can you make a variable out of a loop counter? Let's say that you have several values, A through E, that you would like to run through a FOR...NEXT loop. You want to be able to choose which item to vary, but only use one loop to do any of them.

It's easy to see that if the loop counter is I, then making I equal to A or B will not work because you cannot step a loop when the loop counter is a number. (I would take on the *value* of A or B). If you try it you get a syntax error.

Rather than write five different loops with ON GOTO or some such, we'd like to see simple code that will allow single FOR...NEXT loop to be used for any of the variables A through E. This problem does have a solution. Send us yours. We can't promise trips to Hawaii for it, but the best solution will be printed with full credit to the author. If two or more of you have the same solution, the earliest postmark will determine the winning entry. The general form of the problem is like this:

```
100 A=1:B=2:C=3:D=4:E=5
110 REM
120 FOR I= 1 TO 10
130 F=A+B+C+D+E
140 PRINT F
150 NEXT I
160 REM
```

We want to select, under program control, which value, A through E, to step through the loop. How would you do it?

Sorting

An introduction of sorts

Staff Project. We are biting our tongues to keep from saying "Try this if you are feeling out of sorts". So we won't say it.

One of the most boring jobs - and one that computers can help with - is sorting. There are probably as many sorting algorithms as there are computer science majors, but the one we will describe here is the "Bubble Sort". In future issues, we will delve into some of the more exotic sorts.

The little demo program (listing 1) with this article illustrates the form of the bubble sort. The first lines (100 through 210) provide a way to get some data into an array, A$(I), which we can work with. Lines 180 through 200 display our list of unsorted data before we go into the actual sort.

The purists will be quick to point out that we don't need string arrays to sort integers. True, if we only want to sort integers, A$( ) array can be changed to A( ) array. Note also that if you are going to sort strings, you may need to insert a CLEAR 1000 at line 115, or you will get an "out of string space" error. Array A$( ) holds the data, while array T$( ) is a temporary holding place, used when we do the switch later. Both need to be dimensioned, as in line 120, to some value more than the maximum number of items we intend to sort.

The actual sort takes place from lines 220 through 290. Basically, what we intend to do is to take the unsorted list, compare the first item in the list with the next, and if the first is larger than the next, switch them. Then, take the second item in the list, compare it with the third and if the second is larger than the third, switch them. If an item is already smaller than the next item, we do not switch, but go on and look at the next pair of items.

Let's stop for a minute and find out what this "smaller" and "larger" business is all about. Inside your computer there are only ones and zeros, represented by the presence or absence of a voltage level. Each bit can be represented by either a one or a zero, and eight of these bits form a byte. Any byte, then, can represent a number from zero through 255. The numbers, letters (both upper and lower case) and punctuation are each represented by a unique number. When we sort then, no matter if we are sorting numbers, letters or punctuation, the computer sees only numbers and can tell if one is larger, smaller or equal to another.

In the first line of the actual sort, 220, we set a flag (F) to zero. We will see later how this flag will tell us that the sort is complete. In line 230 we start a loop that takes us from the first item in the list to the next to last item. Inside that loop, in the next line, we make J equal the next item in the list (I+1). Now, A$(I) will always be the current item of the list and A$(J) will always be the next item. Now we can see why the loop only goes to the next to the last item (the N-1 in line 230). When we get to the next to last item in the list, A$(J) will be looking at the very last item. If we don't use the minus one in line 230, A$(J) would be looking one past the end of the list, find a zero there and sort it to the beginning of the sorted list. We would also then lose the last item of our sorted list.

In line 250 the comparison takes place. A$(I), the current item of the list, is compared to A$(J), the next item in the list. If the value stored in A$(I) is equal or less than the value stored in A$(J), we simply leave it alone and jump to the NEXT I in line 280 to compare the next two items. If A$(I) is greater than A$(J), we need to switch them. If your BASIC has the SWAP command, you can simply replace line 260 with: SWAP A$(I),A$(J). Then you wouldn't need to worry about T$( ) at all. For those who do not have SWAP, we need T$( ) to temporarily hold the value of A$(I) while we make A$(J) equal to A$(I). Then we take the value in T$(I) and move it to A$(J). You can see that if we did not use T$( ), we would have lost one of the values because it would have been overwritten.

Now, since we have made a swap, the flag (F) needs to be set. We do this in line 270, and then go to the NEXT I, where the next pair of items will be compared. When we have gone through the entire list once, the program flow drops through to line 290. At line 290 we test the flag to see if it has been set. If the flag has not been set (changed from zero to one), it means that we have gone through the

```
 1 5
 2 4
 3 3
 4 2
 5 1
SWAPPING 4 WITH 5
SWAPPING 3 WITH 5
SWAPPING 2 WITH 5
SWAPPING 1 WITH 5
 1 4
 2 3
 3 2
 4 1
 5 5
SWAPPING 3 WITH 4
SWAPPING 2 WITH 4
SWAPPING 1 WITH 4
 1 3
 2 2
 3 1
 4 4
 5 5
SWAPPING 2 WITH 3
SWAPPING 1 WITH 3
 1 2
 2 1
 3 3
 4 4
 5 5
SWAPPING 1 WITH 2
 1 1
 2 2
 3 3
 4 4
 5 5
SORTED
 1 1
 2 2
 3 3
 4 4
 5 5
# PASSES THRU THE OUTER LOOP=  5
# OF SWAPS MADE=  1Ø
```

Figure 1

entire list of items without the necessity of making a switch - which means that the list is in sorted order. If the flag has been set it indicates that a switch had to be made and that the list may not yet be in sorted order. The program then sends the flow back to line 220, where the flag is reset to zero and the FOR..NEXT loop takes over again looking for swaps to make. This continues until no swaps are made and the flag is not set to one, at which time we leave the sort routine and print the sorted list in lines 320 through 340.

You have probably noted that there are two loops in operation here. An outer loop starts at line 220 and ends at line 290. Inside that outer loop the inner loop (the FOR...NEXT) operates from lines 230 through 280. The outer loop simply operates until the inner loop finds no more swaps to make. Note that the outer loop must always force the inner loop to make one more pass after the list is sorted to determine that it is, indeed, sorted.

If you want to see some of what is happening inside the program, add these lines to listing 1:

```
225 PS=PS+1
265 PRINT"SWITCHING ";A$(I);" WITH
";A$(J)
267 SP=SP+1
350 PRINT"# PASSES THROUGH OUTER
LOOP= ";PS
360 PRINT"# SWAPS MADE= ";SP
```

In figure 1 you can see some of what happens. The list to be sorted is in reverse order from 5 to 1. At the top of figure 1 this reverse order is shown. During the first pass through the outer loop, the digit 5 is moved successively down the list until it is the last item in the list. The 4, 3, 2 and 1 are still out of order. On the next pass, the digit 4 is moved down the list, leaving the 3, 2 and 1 still out of order. Next the digit 3 is moved down the list with the 2 and 1 still out of order. Finally, the digit 2 is exchanged with digit 1 and the list is in order. After one more pass, finding nothing to swap, the outer loop is exited and the pass and swap information is printed. (Perhaps the name "bubble" comes from the fact that the digit 1 in this example "bubbled" up to the top of the list - who knows.)

A worst-case for this sort routine is to have the list in reverse order. In that case, the number of exchanges (swaps) that need to be made is the number of items in the list squared, less the number of items in the list, all divided by two, or to say it another way:(n squared - n)/2. By the way, making this routine sort in reverse order is very simple: change the < in line 250 to a >.

You can see that because of the square term in the expression, the number of swaps (and the time to do them) go up dramatically with the number of items in the list. You would not want to use this method to sort 500 names and addresses. It would probably take several hours. But because of its simplicity (and low overhead) it can't be beat when a couple of dozen or so items need to be sorted. Another thing that slows down any sort with strings is "garbage collection". Every new assignment of a string, as well as intermediate

Listing 1

```
100 REM ** BUBBLE SORT DEMO PROGRAM 1 **
110 PRINT CHR$(12)
120 DIM A$(50),T$(50)
130 PRINT"BUBBLE SORT DEMO"
140 INPUT"HOW MANY ITEMS WILL YOU ENTER";N
150 FOR I=1 TO N
160 PRINT"ENTER ITEM";I;" - ";:INPUT A$(I)
170 NEXT I
180 FOR I=1 TO N
190 PRINT I;A$(I)
200 NEXT I
210 INPUT"PRESS RETURN FOR THE SORT";Z
220 F=0
230 FOR I=1 TO N-1
240 J=I+1
250 IF A$(I)=<A$(J) THEN 280
260 T$(I)=A$(I):A$(I)=A$(J):A$(J)=T$(I)
270 F=1
280 NEXT I
290 IF F=1 THEN GOTO 220
300 PRINT"SORTED"
310 PRINT
320 FOR I=1 TO N
330 PRINT I;A$(I)
340 NEXT I
350 END
```

Listing 2

```
100 REM ** BUBBLE SORT DEMO PROGRAM 2 **
```

(repeat lines 110-210 from above)

```
220 M=1:N1=N
250 IF M=>N1 THEN GOTO 510
255 P=P+1
270 FOR I=M TO N1-1
290 L=I+1
310 IF A$(I)=< A$(L) THEN GOTO 350
330 SWAP A$(I),A$(L):SP=SP+1
340 PRINT "SWAPPING ";A$(I);" WITH ";A$(L)
350 NEXT I
360 PRINT "--------------------"
370 FOR J=N1-1 TO M+  STEP -1
390 K=J-1
410 IF A$(J)=>A$(K)  THEN GOTO 450
430 SWAP A$(J),A$(K):SP=SP+1
440 PRINT "SWAPPING ";A$(J);" WITH ";A$(K)
450 NEXT J
470 M=M+1:N1=N1-1
490 GOTO 250
510 FOR X=1 TO N:PRINT X,A$(X):NEXT X
520 PRINT"# PASSES = ";P
530 PRINT"# SWAPS MADE = ";SP
```

Programming Note

When you want a program to pause, we normally use an INPUT statement like this:

```
10 INPUT"PRESS ENTER";X
```

This will display the message and then continue when ENTER (RETURN on some machines) is pressed. This leaves you with a simple statement with a question mark following it. To get rid of the question mark, do this:

```
10 PRINT"Press any key";
20 IF INKEY$="" then 20
```

This will cause program control to loop at line 20 until any key is pressed, then the program will continue on its way, and the question mark is gone.

Elsewhere in this issue we talked about the PRINT statement. Try this: Find out which character gives you one linefeed on your printer. It will probably be CHR$(10) or CHR$(13). Then LPRINT STRING$(4,13) should make your printer space 4 lines. Note that the CHR$( ) functions do not always perform the same function on the screen and printer. On some some machines PRINT CHR$(12) clears the screen but LPRINT CHR$(12) will give a form feed.

If computers could really reason, would they first insure their own survival or would they go on strike and quit? Maybe we should think about it before we endow them with more power than we would like them to have. Or is the Genie already out of the bottle?

string calculations, are put in memory, leaving behind all the old versions of the string. This fills up memory, and BASIC needs to go through and "clean house". In sorting strings, this can take up several minutes each time it occurs. Naturally, the more strings you are sorting, the less free space there is, and the garbage collection will occur more often and for longer periods of time. This is not only a problem with this sort, but with all string sorts unless some method is devised to prevent it. This problem does not occur when sorting integers. One way to overcome this is to assign pointers to the strings. Then, when comparing strings in the sort instead of swapping the strings themselves, swap the pointer (which is an integer). If your BASIC has the SWAP command it also eliminates the problem of garbage collection.

In general, sorting is based on the ASCII value of a number, letter or punctuation. This causes upper case letters to be sorted before lower case, since upper case letters have a lower ASCII value than upper case. Most punctuation is sorted before numbers or letters. A blank (ASCII 32) will always sort to the top in an ascending list. Try putting these numbers into the program with this article: 9, 8, 10, 5, 1, and 2. These numbers will sort into: 1, 10, 2, 5, 8, and 9. The "10" has a lower ASCII value than "2 nothing", and so it sorts ahead of the 2. Because our little program is sorting strings, try putting a zero in front of the single digit numbers we just tried and do it again. Now it will work right.

The bubble sort, because of its ease of coding and debugging, is very useful when limited numbers of items are to be sorted. In our next issue, one of the programs will use it in some really crazy ways. In this issue, see CARD.BAS for a different sort that uses pointers.

Program listing 2 with this article shows a variation on the bubble sort. It goes through the list moving the largest item to the end of the list. Then it sweeps backward, finding the smallest and leaving it at the beginning of the list. Now that we know that the smallest and largest are in their proper places, we need not look at them again. So the counter which defines the length of the list is incremented by one at the beginning and decremented by one at the end. The next time through the list we do the same as we did the first time, only now we are looking at two less items. Each sweep through the list reduces the number of items to look at by two. When the incrementing number and decrementing number meet, the list is sorted. There is no need then, to go through the sorted list again to see if it is sorted as in the normal bubble sort.

With this sort, the worst case is still a reverse ordered list. It's a slight improvement over the bubble, but not that much. It is fun though, to play with these sort routines and try different approaches. Try them. ●

Programming Notes

How do you assign variables? It's easy to just sit down and start using the ones that first come to mind. But after a while, on a long program, you forget what stands for what and it gets rather confusing. It's a good idea to assign blocks of variables to be used and write them down on paper before you start a program. That way, you can tell by the variable itself what function is should be performing. Don't feel bad though, most of us don't do it, even when we know we should.

If you are an "occasional" programmer, it's easy to forget that a FOR...NEXT loop that is supposed to decrement must be followed by STEP -X. BASIC automatically assumes a step of 1 in an incrementing loop, unless you specify otherwise, but in a decrementing loop the step, even if it is 1, must be specified as STEP -1.

The BASIC function, FIX, removes all digits to the right of a decimal point. It does the same thing as the INT function, except that FIX does not round down negative numbers, while INT does. If your BASIC does not have FIX, you can simulate it like this:

X=SGN(N)*INT(ABS(N)).

TRACE on/off is a handy debugging tool most of us forget is there. If you use the command TRON from command mode and run a program, it will show you the line numbers as they execute. TROFF will turn TRON off. Both statements may be used as commands within a BASIC program to inspect program flow through a particular section of code. Simply insert a line with TRON before the section you are interested in, and another line containing TROFF after that section. Now, when the program gets there, it will print the line numbers as they execute, in the order that they are executed, and you can see what's happening.

Sources

Where to Find Programming Tools

Software Studios, Inc has introduced a new data and text encryption program called Scrambler. It will convert any ASCII text or data file into undecipherable representations based upon a code key input by the user. The same key will restore the text to its original form. Code keys may be any character, word or phrase up to 255 characters in length. It is case sensitive and respects the difference in upper, lower or mixed case code keys. The original file is overwritten, so there is no trace of the original on disk. Files may be encrypted any number of times for multi-layered security and may also be telecommunicated without interference or alteration. Scrambler requires 64K, a disk drive, and will run on IBM-PC, XT, AT and MS-DOS compatibles. It is available for $49 plus $2 shipping from Software Studios Inc., 8516 Sugarbush, Annandale, VA 22003 (703) 978- 2339.

Source Telecomputing Corp. has announced 2400 baud service for The Source. Prime-time 2400 baud usage, weekdays 7 am to 6 pm will be 46 cents per minute, only 3 cents per mintue more than 1200 baud. Non-prime time usage, including evenings, weekends and holidays, will be 20 cents per minute, only 2 cents per minute higher than 1200 baud. This new baud rate will first be made available by Uninet in ten major cities, including Los Angeles, San Francisco, Washington DC, Atlanta, Chicago, Boston, Kansas City, New York City, Dallas and Houston. Several hundred additional cities will be rapidly added to the roster by Uninet. Concurrently, STC has replaced all hourly pricing with per-minute pricing to reflect more nearly the way members use The Source and make it easier for members to reconcile their monthly invoice.

The Dental Computer Group is composed of dentists, physicians, and other health care professionals interested in office computing. We support all brands of software, micro and mini computers, and feature tapes, lecture meetings, a hot line, software reviews, and a free contributed software library. The monthly Dental Computer Newsletter contains a wealth of information about the latest computer hardware and software, good buys, book reviews, helpful hints and practice management advice. To get a free copy of the newsletter and group information, send a large, stamped, self-addressed envelope to Dental Computer Group, 1000 North Ave., Waukegan, IL 60085 or call Skip Nye at (312) 223-5077

Polygon Industries announced the release of TRANSLATOR, the artificial intelligence software program which translates languages; not computer languages, but French, German, Spanish, English and Italian, and soon several others. The software has been over one year in development, passed beta test and is now in stock for delivery. Anyone may order directly from Polygon Industries at the price of $49 for each language; to and from English. An input text file, which can be a letter, book passage, news story, etc., is specified to the program and an output file, that will contain the translated text. Alternatively, a line of text can be input for immediate translation. The software is offered for all computers which use at least one disk drive and 48K. Each module will translate to and from English with about 90% accuracy. The user can add to the vocabulary. A hard disk size vocabulary is optionally available. Polygon Industries, PO Box 24615 New Orleans, LA 70184

Polygon Industries announced the release of FLIGHT CHECK, the airplane operation software program that computes best altitude and power settings for a given flight, weight and balance, and reserve fuel at destination. These computations should be done before each flight, but few take the two hours to do them, so this program can save up to 10% of the cost. Best of all is the assurance that the takeoff will succeed and there is enough fuel for the trip. Polygon Industries, PO Box 24615, New Orleans, LA 70184 (504) 282-5372

The listing of products in this column is to tell our readers what is available in the marketplace. We take news release items at their face value. No endorsement by this publication is implied by the appearance of products in this section.

Hype

Where we ask you to Subscribe!

To help us serve you properly, would you kindly take a moment to answer the following questions -

Make and Model of your computer _____

Do you have disk drives? ☐ **Yes,** ☐ **No** Which DOS do you use? _____

Do you have a line printer? ☐ **Yes,** ☐ **No**

Do you use a Modem? ☐ **Yes,** ☐ **No** What baud rate **300** ☐ **1200** ☐

As a Basic programmer, do you rate yourself as ☐ **Novice,** ☐ **Intermediate or** ☐ **Advanced?**

Which other programming languages do you prefer? _____

Comments:

Fold

Subscription ORDER FORM 1185

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

☐ **Check or MO enclosed.**
☐ **Charge to my VISA/MasterCard #**_____**Exp date**_____

Please Print clearly:

Name _____

Address _____

City _____ **State** _____ **Zip** _____

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

Charge card orders may be called in (206) 475-2219 between 9 AM and 4 PM weekdays, Pacific time. Sorry, no "bill me" orders.

Download

Current status of the CodeWorks download

The CodeWorks download system is almost ready to be tested online. We expect to be online by December 1st. The dedicated telephone number for the download is (206) 475-2356. You may call that number prior to December if you wish. If we are not up when you call, the number will not answer and you will not be billed for the call. If we are up and running, you will get a login prompt, and we will ask you to enter your name.

The download will initially be set up for 300 baud, 8 bits, no parity and one stop bit. At first we will handle ASCII transfer only. (X-modem and 1200 baud come later.)

Please bear with us during this testing period. When the system is fully operational, we will have special subscriber logon procedures, a place for non-subscribers to get a sample program and leave their names and addresses, and a method for subscribers to assign their own passwords.

All major programs listed in CodeWorks will be available for download once the system is up. We expect it to be online 24 hours per day, seven days per week (except for power failures and maintenance breaks.)

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Address correction requested

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

CodeWorks

Issue 3 Jan/Feb 1986

CONTENTS

CODEWORKS

Issue 3 Jan/Feb 1986

Editor/Publisher
Irv Schmidt
Associate Editors
Terry R. Dettmann
Greg Sheppard
Jay Marshall
Circulation/Promotion
Robert P. Perez
Editorial Advisor
Cameron C. Brown
Technical Adviser
Al Mashburn

Authors: We constantly seek material from contributors. Send your material (double spaced, upper/lower case please) and allow 4 to 6 weeks for editorial review. You may send IBM-PC compatible diskettes (please save your programs in ASCII). Also send a hard copy listing of the program and article. Media will be returned if return postage is provided. Cartoons and photographs are welcome. 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 will receive all issues for the subscription year. Not available outside United States Zip codes. **VISA and Master Card orders are accepted by mail or telephone (206) 475-2219.**

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in **The United States of America. Bulk rate postage 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.)

Editorial

By the time you read this our download ought to be up and running. We have tried to anticipate all the problems and bugs, but past experience says there will be some anyway, so bear with us during the shakedown period.

You will find this issue slightly larger than the previous ones. This is mostly due to the length of our Wood program. I didn't feel right about taking that much space from other things, so the added pages sort of make up for it.

Does anyone miss the "Sources" page? I went through a big box of press releases and would you believe, there wasn't much there. Some of them are so hyped up about their thing they forget to tell you what it actually is or what it's for. I like to plug the small guys when they have something good. After all, we are small guys ourselves and know what it's like.

Does anyone know anything about the origins of a game called "Startraders" (or sometimes just "Trader") which has been around since the 1970's? It looks like it might be public domain, but we're not sure. We have been working with it to put it into a new setting with added features, but would like to give the proper credit to the original authors.

Speaking of giving credit, all our programs (unless specifically noted) are public domain. We do insist, though, that our credit lines be left intact when they are passed around. It's not ego on our part, but a way to wave our hands and say "here we are." Call it advertising if you will, but unseen and unheard of is unsold, no matter how you cut it.

Several of you have called asking for back issues. We have none. Here is how it works. All subscribers during the first year get all issues for that year. First of all because we do a lot of backward referencing to previous articles. Secondly, because everyone will come up for renewal at the same time, saving us a bundle because we can announce renewal in one issue that way. Also, all first year subscribers get seven for the price of six, since the first one is free to everyone. What a deal.

And speaking of issues, our last one (Issue 2) was not entirely clean. It seems about 300 or so copies had solid black ink on at least two pages. They got out undetected, and ended up mostly in the 89XXX and 90XXX zip codes. Many of you wrote or called and asked for a better copy, which we immediately sent. If you have one of these and want a replacement, please don't be bashful. Drop us a postcard and we will get you a clean copy right away. Our printer has made apologies to us and we pass them right along to those of you who got "blacked out."

Since we are on a yearly basis, any of you who are missing any copy can let us know and we will supply it.

Happy New Year. Happy computing, and see you next time.

Irv

Forum

An Open Forum for Questions & Comments

Glad we subscribed to your new magazine. (We are) learning a lot and enjoying the articles. We have a Tandy 1000. We have found little support for the 1000 BASIC programs so we have been keying in the IBM-PC programs. Although the 1000 is supposed to be compatible with the IBM-PC we are having trouble running the programs. Maybe there are some helpful hints that you could explain in your magazine that would make these programs more compatible.

D. Degenhardt
Aurora, IL

Our experience shows that the Model 1000 is very compatible with the IBM-PC. The ASCII character sets are identical and the BASIC commands and functions are almost identical. Some "store-bought" software may not work with the same function keys, but the programs we (and others) publish should run without problems. You obviously already have the BASIC manual for the 1000. May we suggest that you purchase the book "Learning IBM BASIC", by David A. Lien, published by Compusoft Publishing, ISBN 0-932760-13-9, for $19.95?

Good for you! A computer magazine that dares to give us software, ignores the color and omits the ads. For those who like to hack away and use logic to create something of our own. Stick to your guns. How about games? I think they are the best source of learning graphics. Count me in.

Calvin E. Host
Harrisburg, PA

Thank you. How about games? Well, I used to think that most computer games were trivial. Then not long ago, Cam Brown asked if I knew a way to deal a deck of cards into four Bridge hands. I finally got the deck made up, shuffled and dealt in about 2 seconds. I gave it to Cam, but was so intrigued with the idea of a card game that I started working on a Poker program. Now I am up to my neck in it and haven't been so involved in tricky code since I wrote the Wood program (in this issue). It, and another game we have in mind, will appear in some future issue. I am now of the opinion that writing an involved game from scratch, and stubbornly attacking every problem that comes up, instead of circumventing it, is probably some of the best programming experience one can get. - Irv

Re: Writer in issue 2, line 650. I assume it means any text file I have out there - if so is N a counter and of what? How does my text file get an N in its beginning? I always get N=0 and the note that the file is not long enough! P.S. Would you check and see if I've subscribed yet - I like it!

Sally Dion
Goleta, CA

Because of the way most word processors handle line feed and carriage return characters the Writer program can't use them. It assumes you have entered text through the program itself, in which case it assigns the N value as the first line of the file and N tells how many lines there are. Yes, you have subscribed and thank you.

Enjoy your magazine very much. Just finished typing CARD/BAS from Issue 2. Tried to RUN and got "Sorry, File is full" error. In the text it is mentioned to CLEAR XXXX in line 115, which should be before line 110. *(Our text said line 115 when it should have said 105.)* I made this change and it worked fine. Probably many others will write about this but thought I would get my two cents worth in. Keep up the good work with CodeWorks.

George Phillips
Sun City, AZ

You are right. We keep telling ourselves to "clear up" that CLEAR XXXX problem. In Microsoft BASIC, release 4.51 and earlier, the CLEAR statement was necessary to clear string space in memory. The next release of BASIC was 5.0. In it, the format for CLEAR is: CLEAR (arg1),(arg2). Its purpose is to set all numeric variables to zero, all string variables to null, to close all open files and, optionally, to set the end of memory and the amount of stack space. Argument1 is a memory location which, if specified, sets the highest location available for use by BASIC. Argument2 sets aside stack space for BASIC. The default is 256 bytes or one-eighth of the available memory, whichever is smaller. In previous versions of BASIC, argument 1 sets the amount of string space and arguement 2 sets the end of memory. Release 5.0 and later allocates string space dynamically. An "out of string space" error occurs only if there is no free memory left for BASIC to use.

When I first received the sample issue of CodeWorks I thought it was of no use for TI-99 users. I almost threw it in the trash but saw the upcoming attractions on the back cover. Being a woodworker who has spent many hours figuring out how best to cut out the pieces for a project, your program "Wood" got me real excited. I decided to try translating some of the programs. I now have two of your programs running fine on my TI- 99/4A. One of the programs I chose was Maker. I figured that if I could get this one to work the others would be easy. I like the no advertising policy and the program explanations. If you use any PEEKS or POKES please explain what they do. I like the idea of putting the programs into public

domain. I like programs that get the job done without a lot of razzle-dazzle that only gets in the way, slows the program down and gets to be a real pain after a while. I think you are going to have a great magazine. Good luck.

Robert L. Keeney, Jr
Wilmington, NC

Because TI BASIC is so utterly different, we didn't really expect TI owners to accept our publication. We admire your fortitude, and welcome to CodeWorks!

I am having problems. My basic problem is that I am dumb when it comes to programming. From this all the rest of my problems stem. I have a Kaypro-4, using CP/M, with the Perfectwriter word processing software program. I also have a MBASIC handbook and an SBASIC handbook, both as part of the sales package. For the past two years I have had need only to use the word processor software. Now I want to use your magazine material which I think is excellent. I apparently cannot use your material without some modifications while using the SBASIC compiler. Is there a way to enter your programs using only the MBASIC interpreter instead of the compiler? Please bear with us novices; we yet may be able to see the light at the end of the tunnel.

Harlan Trent
Ashland, OR

No point in messing around with a compiler if you don't know interpreter BASIC. Yes, our programs are designed for your MBASIC interpreter. Some of them were written on a CP/M machine using MBASIC. Your "clear screen" command on the Kaypro is probably PRINT CHR$(26) instead of the CHR$(12) we sometimes use. This may be oversimplification, but call up your MBASIC, type our listings in and RUN them. If you have made no typographical mistakes they should run.

I was very much impressed with your first issue of CodeWorks. However, your second issue was not up to snuff. It seems to me that CodeWorks should be aimed towards the intermediate to advanced BASIC programmer and that Beginning BASIC has no place in the magazine except as a space filler. I think that too many magazines give too much space to beginning BASIC when there are many, many books on the subject. Please, NO more Beginning BASIC!

H. Lawrence Abbott
Wyomissing, PA

I am enjoying getting CodeWorks. My interest is in programs (in BASIC) of various procedures: one liners, programming short cuts, scientific procedures, mathematics, programming notes, utility programs, etc.. I do not care what president of what company has changed jobs or who got fired, etc., etc.. New products are interesting. I disagree with the one who said that BASIC was "yuk". Please continue with the "Beginning BASIC." As a self-taught programmer of 5 years, I find Beginning BASIC valuable.

William McCord
Lodge, SC

It would be nice if we could write a magazine that exactly matched the current level of expertise of every reader. But we could no more do that than you could describe your exact level of expertise. Aside from that, each of you would have your own edition of the magazine, which would be horrendously expensive! We are aiming at the "adult user" who uses Microsoft BASIC. That user has an IBM-PC, a clone, one of the Tandy computers or uses MBASIC and CP/M. We think that two pages of Beginning BASIC is not too much. I find in writing it that I usually rediscover some fine point I have ignored or missed along the way. You may find the same in reading it.

I enjoyed Issue 2 as much as Issue 1. Keep 'em coming. They are great. I loved the calendar program and intend to send it out to my public accounting practice clientele as a yearly "Happy New Year" sort of thing from now on. I had one problem, which may just be peculiar to my Tandy Model II, but finally located and corrected it. It's line 840, where you go through the D loop four times to print the four numbers for the year. The semi colon, which correctly holds the line the first three times through, also holds it the fourth time - which created disaster on my machine. My solution was to run the D loop through just three times, then - on line 845 - do the LPRINT bit again, without the semi-colon. It worked just fine then. I wish you and your staff the best of luck. Thanks again for a great publication.

Harry Birchard, CPA
West Chester, PA

The Calendar program is an excellent piece of work - congratulations. My calendar problems are solved until 3999 - I believe my computer may be obsolete by that time. I would like to point out two modifications that would make the program more efficient. 1) Line 690 - remove the space between the R in September and the O in October. This will make the start letter of each month begin over the N in MON. Note in your reprint on page 13 - October, November and December are displaced one space to the right of N in MON. 2) There is a line feed - in the form of LPRINT " "- missing after the FOR...NEXT loop in line 840. The enlarged year title wants to print in the same line without the line feed. Keep up the good work.

LeRoy Carson
Goodland, IN

I want to thank you for a magazine that has really caught my attention. I am doing more with my computer since CodeWorks came. I am writing about the program Calendar (Issue 2). In line 620, in the third formula (the one just after the word "and", the *400 is not there. It is necessary, or the 29th day of February will not be printed in century years that can be divided by 400. I am not that good at programming, but I like reading and understanding programs when I can. Fixing this omission has been challenging and rewarding to me.

C. N. Harrid
Baltimore, MD

*You are right. The *400 is missing from that line, and the year 2000 should be a leap year. After the AND, it should read: ((Y—INT(Y/400)*400)<>0).*

As to the previous two letters, we have had several readers who had problems with the LPRINT in the year heading. We have three different printers here and it worked on all three. In looking at the code now, we wondered how it ever worked at all. It turns out that all three printers provide a line feed and a carriage return at the end of each line, unlike some, where the computer sends the line feed.

Thanks for the prompt response to my inquiry about the difficulty with Writer.BAS from your first issue. Your suggestion solved my bug problem and I have it up and running in fine style. In the meanwhile, I have received Issue 2 and find it as interesting and useful as your first issue. So, I feel that my investment in a subscription is going to be well rewarded. Your "biggie" for Issue 2, Card.Bas, was something else that I thought I could use, so I punched it in and played with it a little. Then I made some changes which improved it for my need, and I think might be useful to others. First, I changed the data format fields to:

```
570 A$="1- FNAME"
580 B$="2- LNAME"
590 C$="3- COMPANY"
600 D$="4- ADDRESS"
610 E$="5- CITY ST"
620 F$="6- ZIPCODE"
630 G$="7- PHONE"
640 H$="8- ITEM"
```

This format for the data entries now allows the list to be sorted and searched by Last Name (as well as for Company, City, Zip, etc.) For print-out purposes, it is well to remember to add one space at the end of each FNAME field entry so that there will be a space between the first and last names in the labels. This could undoubtedly be written into line 2180, but I haven't tried it yet. Of course, the label print format has to be changed in line 2180 to 2200 as follows (for my needs):

```
2180 LPRINT A$(P(I));B$(P(I))
2190 LPRINT C$(P(I))
2195 LPRINT D$(P(I))
2200 LPRINT E$(P(I));F$(P(I))
```

Additionally, if PHONE and ITEM are also wanted in the printout (for file cards for example) add line(s) to accommodate them and change J accordingly.

Since I have several address files that I want to use, I've renamed (and copied) the program as CARD1.BAS for the first one and then changed line 160 and 330 to name the data file CARD1.DAT. Then the next different program is named CARD2.BAS and 160 and 330 are changed to CARD2.DAT, etc.

I wanted to print out one of my files onto form-fed 3 x 5 cards, so I changed line 2210 to accommodate the 3 inch separations by changing the J values to J=1 to 14 (for 6 lines per inch printing) using my 4 line format.

It is an elegant and very useful program. The only thing that it lacks perhaps, is the ability to select out individual groups of records to be printed without printing the whole file. (All the records for one city, for example.) Probably they could be selected and collected into a temporary CARD.TEM file which could be printed then cleared?

Keep up the good work. You are certainly filling a void for those of us who have an interest in BASIC programming.

Allie C. Peed, Jr
Rochester, NY

When we started this magazine we hoped to be a springboard for ideas and for readers to take our programs and expand, enlarge and modify them. You, and many others who have written, are reassuring us that this is exactly what is happening. We like it.

Thanks for an unusual and useful magazine. On page 7 of your Sep/Oct 1985 issue under Programming Notes, you point out some differences between MSDOS and TRSDOS BASIC clear screen commands. There must be other differences. Have you discussed these differences comprehensively in a past issue? Or, do you know where I can get a comprehensive listing of such differences?

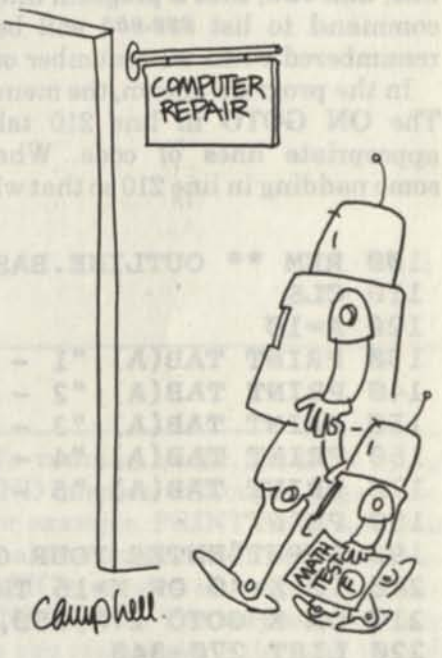
Carlyle Maw
Washington, DC

There were no past issues. Sep/Oct 1985 was our premier issue. Yes, there are some differences which we are covering as they come up. With any luck, and a little space, there should be a discussion of the difference between PRINT@ and LOCATE in this issue's Programming Notes, for example.

I subscribe to three computer magazines, but expect to drop them and get what I want from CodeWorks. I believe you'll "do it."

Wayne Shaneyfelt
Aurora, NE

If we don't, it will not be because we didn't try.



Outline.Bas

Putting Form into your Programs

Text by the Staff
Program idea by JoAnn Blume, Seattle, WA

Most of us at one time or another have gotten bogged down in a program trying to keep track of where the subroutines are. This is especially true when the program grows larger and when we have renumbered several times.

BASIC, being unstructured and forgiving, allows us to paint ourselves into almost any kind of corner. Most of us usually do.

Other languages force a structure, don't use line numbers, and allow the use of labels for GOTO and GOSUB. That way, no matter how you add or subtract from the program, the labels remain intact. In BASIC, we can get the same effect as a label would give, and some added convenience as well.

The program OUTLINE.BAS (see listing) is an example of how to do it. It looks more complicated than it is, and if we may say so, is rather clever. It works on two interesting ideas: One, that the command to LIST can be used within a program line, and two, that a program line containing the command to list ###-### will be automatically renumbered when we renumber our lines.

In the program shown, the menu has five items. The ON GOTO in line 210 takes you to the appropriate lines of code. What is added is some padding in line 210 so that when you pick the menu item plus ten, the program will list the appropriate section of code corresponding to the menu item. What this means is that during program development you may choose menu item 4 to run, but if there is a problem in that section of code you can break, run again, and pick menu item 14, which will list that section of code *no matter where it is in the program*. If you have renumbered, then the lines in 220 through 260 will also have been properly renumbered, and you can always get to the section of code you are interested in.

So now you have the effect of labels, with the added convenience of being able to list that section selectively no matter where it is. Of course, when the program is finished, you remove the extra line numbers in the ON GOTO line and the LIST lines. In their place you may want to add some error trapping for numbers less than the lowest menu item or greater than the highest.

This idea has additional merit because it almost begs you to decide on some sort of structure before you start to code. Not a bad idea in itself.

What do you do if your program does not have a menu? Make one anyway, use it during development, and then dump it when you are done. We have been using this for some time now and find it works nice and keeps things in order. ■

```
100 REM ** OUTLINE.BAS * FOR CODEWORKS MAGAZINE **
110 CLS
120 A=10
130 PRINT TAB(A);"1 - TO CREATE A FILE"
140 PRINT TAB(A);"2 - TO ADD TO THE FILE"
150 PRINT TAB(A);"3 - TO EDIT THE FILE"
160 PRINT TAB(A);"4 - TO LOAD THE FILE"
170 PRINT TAB(A);"5 - TO SAVE THE FILE"
180 PRINT
190 INPUT"ENTER YOUR CHOICE";X
200 IF X=<0 OR X>15 THEN GOTO 190
210 ON X GOTO 270,350,430,510,590,0,0,0,0,0,220,230,240,250,260
220 LIST 270-340
230 LIST 350-420
240 LIST 430-500
250 LIST 510-580
260 LIST 590-660
270 REM *** START OF CREATE A FILE ROUTINE **
```

```
280 PRINT" START OF CREATE A FILE IS HERE"
290 '
300 '
310 'CODE GOES IN HERE
320 '
330 '
340 END
350 REM *** START OF ADD TO FILE ROUTINE **
360 PRINT"START OF ADD TO FILE IS HERE"
370 '
380 '
390 'CODE GOES IN HERE
400 '
410 '
420 END
430 REM *** START OF EDIT A FILE ROUTINE **
440 PRINT"START OF EDIT A FILE IS HERE"
450 '
460 '
470 'CODE GOES IN HERE
480 '
490 '
500 END
510 REM *** START OF LOAD A FILE ROUTINE **
520 PRINT"START OF LOAD A FILE IS HERE"
530 '
540 '
550 'CODE GOES IN HERE
560 '
570 '
580 END
590 REM *** START OF SAVE A FILE ROUTINE **
600 PRINT"START OF SAVE A FILE IS HERE"
610 '
620 '
630 'CODE GOES IN HERE
640 '
650 '
660 END
```

(See Notes, below, covering REMarked lines.)

Programming Notes

Many of the computers we address can only see two letters as variable designators. You can use more with these machines, but only the first two are recognized. Because of this, we tend to stay away from more than two letter variables. If they should creep in, we try to make sure that there is no conflict with other variable names. Don't be alarmed if you see a program here (or any other magazine for that matter) that uses long variable names. You can still use them. Just be sure that no two first characters are alike.

Most BASIC's running under MS-DOS allow the use of the RND function without an argument. You can say, for example, PRINT RND, and you will get a decimal fraction between zero and one. Most other BASIC's require an argument with RND, like, PRINT RND(0). You may want to keep this in mind as you read pages 8 through 11 and page 39 of this issue.

The shorthand way to REMark a line in BASIC is to use the apostrophe (') instead of REM. But did you know that "REM" requires 2 *less* bytes of memory than does the apostrophe? This doesn't sound right, but it is true.

Random

How it works and what is it for?

Staff Project

There are many forms of Random (RND) on the various machines existing today. Sometimes, trying to get what you want from RND can be tricky. This article addresses some of the difference in the random function. It should cover the majority of BASIC's. Try yours and see how it compares.

Random is only useful in games to decide moves and to simulate the throw of dice or pick cards from a deck, right? Not really. There are many other uses for the random (RND) function.

One especially valuable use of random is in testing a program. Elsewhere in this issue we present a program which attempts to determine the best way to cut a sheet of wood. Although the program does not now contain one occurrence of the RND function, it was used extensively during development of the program. To test the program, a random piece generator was programmed which created 1000 random groups of pieces of wood. These groups were then fed to the program, and results (good or bad) were printed on the printer for later evaluation. The particular computer we used required a "seed" number for the random generator. Because a seed was needed, the same random sequence could be easily repeated by using the same seed number. This was especially helpful in comparing results after slight modifications were made to the program. It allowed us to repeat the same random sequence and compare results with previous runs.

Another application of the random function could be in creating a large set of random characters to act as a test for a sort routine. An entire diskette full of random characters in a random file can be created this way. This technique has a way of showing the programmer how to "bullet proof" a program by taking almost any situation into account.

One of the most interesting applications for the random function is in using the "Monte Carlo" method of calculating probability. Essentially what this method entails is that if enough random points are chosen within a defined area, the distribution of those random points will be uniform. Now, given an aerial photograph (or a surveyor's map) of a piece of land containing a lake with an irregular shape, you can get a very good approximation of the area of the lake.

Before we can play with these interesting ideas though, we need to find out more about the RND function.

RND(X)

Many computers use the simple expression A=RND(X) to get a random integer between 1 and X. If X=0 however, the number returned will be a number between 0 and 1 (.12345 for example.) A=RND(10) will return a random integer between 1 and 10. Note that you cannot get this expression to return a zero. If you need zero, use A=RND(10)-1. Now you get numbers between 0 and 9, and if you want to include the possibility of getting the number 10, do this: A=RND(11)-1. Tandy Models I,II,III and several others use this method to generate random numbers. But where do they get the "seed"? For the most part, it's done internally using the system clock and the memory refresh circuits. This makes it very easy for the programmer to create random numbers, but does not allow for a repeatable sequence of random numbers.

Most MS-DOS machines and several others, including MBASIC running under CP/M, use an internal clock to give a seed to get a decimal fraction for RND(0). Almost all these machines will return the same sequence because the seed does not change. This is not all bad however, since most of these machines will also allow the operator to enter his own seed number. If the statement RANDOMIZE is given, BASIC will ask for a seed number (usually between -32767 and 32767.) Now you have the choice of a very large number of different seeds and also the capability to repeat

any random sequence.

One of the disadvantages of RANDOMIZE is that it stops the program and asks for a seed number. You can program it like this: RANDOMIZE 76. Now the program will not stop, but the seed is 76 and every time the program is run the random sequence will be the same.

If your machine has TIME$ (and most these days do) you can strip off the last two or four digits, change them to integers using the VAL function and use them to seed your random generator. The minutes and seconds from TIME$ will give you a respectable array of different seeds. Here is an example of one way to do it:

```
10 A$=MID$(TIME$,4,2)+MID$(TIME$,7,2)
20 A=VAL(A$)
30 RANDOMIZE A
```

If TIME$ happens to be 11:23:02 when this code is called, A will be equal to the integer 2302. This will give 3600 different possible numbers for your seed, and will not stop execution of the program to wait for operator intervention. At this point the random number will still be a decimal fraction. How do we get random integers, or better yet, a range of random integers?

By using the INTeger and some razzle-dazzle we can get just about anything we want from the random function. R=INT(RND*10+1) will return random integers between 1 and 10. R=INT(RND*100+1) will return random integers between 1 and 100, etc. As in the discussion at the beginning of this article, zero is not included in these examples. If you need to include zero use: R=INT(RND*10+1)-1. Now you will get numbers between 0 and 9, and if you need to have the 10 included also, then change the 10 to 11.

How to get a range of numbers starting and ending at your choice? This should do it: R=INT(RND*A)+B. Substituting numbers we get:

```
R=INT(RND*100)+50
```

A is now 100 and B is 50. The range of numbers this will produce is between 50 and 149 inclusive. In other words, B is the smallest number you can get and the largest will be (A+B)-1. Again, you can "doctor" the formula to suit your needs.

Defining random as a function

In BASIC we have a (too little used) statement called DEF FN, which stands for "Define Function". We can put our formula from above into a defined function and then when we need it, we can feed it the range we want and it will return our random number within that range. It can be done like this:

```
10 DEF FNRAN(A,B)=INT(RND*A)+B
20'
30 PRINT FNRAN(100,50)
or, later in the program,
90 RN=FNRAN(51,10)
```

Keep in mind that a DEF FN cannot be executed in immediate, or command, mode. It will only function during program execution.

Monte Carlo and the Seattle Busses

It turns out that the Seattle bus system found that the number of riders on their busses decreased as the fare was increased (not an unusual finding.) The rate of decrease was expressed as the number e to the minus X cubed. It makes a graph like figure 1. The area under the curve indicates the total revenue collected.

There are several methods to calculate the area under the curve. The one we will consider here is the Monte Carlo method. Program Monte Carlo (see listing) will determine with fair accuracy the percentage of the total area of the graph which is under the curve. It does it by confining 1000 random numbers to the limits of the rectangular area and letting the numbers fall within that area where they may. It then assumes that the distribution of the random points was uniform and counts those that fell above the curve. Subtracting this number from the original 1000 gives the number of points below the curve.

The program first defines the curve in lines 150 through 180. The formula EXP to the -X cubed appears in line 170. (We used variable L instead of X, but it makes no difference.) In line 160 it was necessary to divide L by 1000 to keep the number within the range of the EXP function. EXP(N) will overflow if N is greater than about 87. All the values for the curve are stuffed into array A(I).

The loop from lines 190 through 230 then generates random numbers and compares them (line 210) to the value for the curve in A(I). If the value of the random number is larger than the value of the curve at that point, counter Q is incremented by one. In the end, Q will tell us how many random numbers landed above the curve. The rest of the program simply prints out Q and the percentage of points below the curve.

The problem with Monte Carlo is in defining the boundary of the area to be measured. In our bus case it was easy enough - we had a formula to

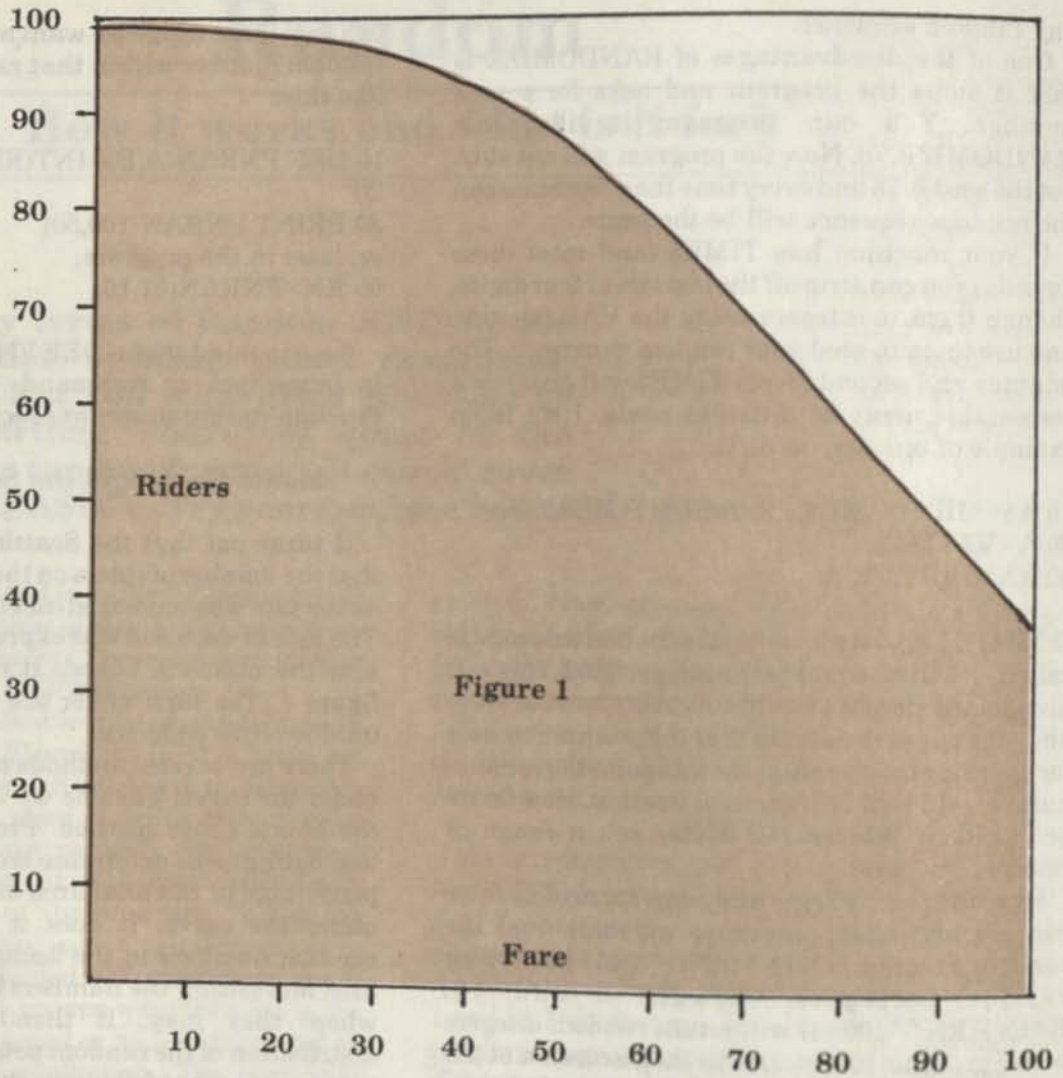


Figure 1

Program "Monte Carlo"

```
100 REM ** MONTE CARLO METHOD *
110 REM ** SEATTLE TRANSIT FARE
    AND RIDER PROBLEM **
120 RANDOMIZE
130 N=1000
140 DIM A(N)
150    FOR I=1 TO N
160     L=I/N
170     A(I)=INT(EXP(-L^3)*10000)
180    NEXT I
190    FOR I=1 TO N
200      RN=INT((RND*10001)+1)-1
210      IF RN>A(I) THEN Q=Q+1
220      PRINT RN,A(I)
230    NEXT I
240 PRINT
250 PRINT"THERE WERE ";Q;" OUT OF
   ";N;" ABOVE THE CURVE"
260 PRINT" AREA UNDER THE CURVE
IS ";((N-Q)/N)*100;" % OF TOTAL"
```

The area under the curve indicates total revenue collected by the bus company. As the fare increases, the number of riders goes down. It was found that the curve fit the formula e to the minus X cubed. If your random function works like most of those on personal computers we have tried, you should find that about 81% of the area of the rectangle is under the curve.

NOTE: Use RND(0) in place of RND on non-MS DOS machines. Also, you may need to use EXP(—(L^3)) on some machines that don't evaluate powers prior to subtraction.

define the curve. In a sprawling lake with switchback inlets on a piece of property it would be another matter entirely. Obviously, the better the definition of the boundaries and the more random points you use, the better the accuracy. Even though the program we used is short, it still takes a while to calculate and compare all those points.

One would rather expect that the curve the bus company came up with would fit any situation with price/buyers. This is not necessarily the case. There are many other influencing factors. On some items it has been shown that there is a secondary bump in the curve as price goes up. This has been attributed to "status" buyers. That curve looks like the one we have shown, but has a nice bump on the downside, after which it continues down. Someone in our office labeled it the "snob knob", although we are unaware it really has a name at all. The reason we mention it is that it would be difficult to come up with an expression to define that kind of curve.

In Seattle they had a policy for a while called "free Wednesdays," during which one could ride the bus for free. Although we do not know exactly what they were trying to do, it seems likely they were trying to determine the universe of riders available and set the high end of the curve.

Random numbers can also be used in testing multiple probabilities. It is a fascinating subject, and one we will delve into in some upcoming issue.

HOW RANDOM IS RANDOM?

Computers don't produce random numbers. Once a program is written, the numbers that the computer produces can be deduced. The best we can do is produce number sequences that have properties which are very similar to those of random numbers.

Don't confuse random with arbitrary. Random requires that every number should be equally likely, arbitrary means we don't care what the number is. In almost every case we desire more than just one number. It is a sequence of random numbers that is needed for an accurate simulation or calculation.

Computer generated random numbers are called pseudo-random: they may walk like ducks, quack like ducks and look like ducks, but they aren't ducks. How can a program be checked to see that it does create uniform random numbers (each value equally likely)?

The statistics required to evaluate a sequence are well developed but quite complicated. One of the easiest tests is to do a Chi-Square analysis. In a set of N random numbers, all less than or equal to 100, we can expect N/100 numbers of each value. At the same time, the frequencies of occurrence of all the values should not be exactly the same; that would not be random.

Let's make 1000 randomly-generated integers, all of which are less than or equal to 100 and keep track of their frequency in an array. Program Chi-Square will compute the value of your array.

If your value of CHI is close to 100, the numbers are random. If it is far from 100 they are not random. Tables exist to determine what is meant by close or far (within 20 is close.) Good generators should pass this test 90 percent of the time; bad ones will not. How well does your computer do?

Reference: *Algorithms* by Robert Sedgewick, 1983, Addison- Wesley Publishing Co., Inc.

Program notes: See related article on Random Numbers. Depending on your computer, you may run the program as is or delete or modify lines 130 through 160 and 190. As shown, the program was developed on a computer which required a seed. The seed is derived from the system clock in lines 130 and 140 and uses the minutes and seconds from TIME$. Just as a matter of interest, we ran this program on five different machines and found that in the ten tests all those machines fell outside of the plus/minus 20 percent limit at least once. ■

```
100 REM CHI SQ TEST FOR RANDOM
110 DIM F(1000)
120 RN=INT(MID$(TIME$,4,2)+MID$(
    TIME$,7,2)):RANDOMIZE RN
130 FOR I=1 TO 100
132   F(I)=0
133 NEXT I
140 FOR I=1 TO 1000
150   N=INT(RND*100)+1
160   F(N)=F(N)+1
170 NEXT I
180 REM FREQ OF N IS NOW IN F(N)
190 REM NOW TO COMPUTE CHI SQ
200 TL=0
210 FOR I=1 TO 1000
220   TL=TL+F(I)^2
230 NEXT I
240 CHI=(TL/10)-1000
250 PRINT"CHI SQ VALUE IS ";CHI
```

Beginning BASIC

The way we enter values for variables in BASIC is to use the INPUT statement. There are several forms of this statement. Let's look at some of them and see what they do and how they differ.

The INPUT statement we use most is simply: INPUT N. When BASIC sees this statement, it stops and prints a question mark on the video screen, then waits for you to input some integer value. The number you give it at this point is assigned to variable N, which means that it will be stored in a location in memory which the program will call N. Nothing happens though, until you press the Enter or Return key. BASIC then stores the value in N and goes on to the next line of the program.

If you input two values when only one is called for, as in answering an input request with: 25,30 BASIC will print "Extra Ignored" and take only the first value. If you want to enter two or more values, the form the INPUT statement must take is: INPUT A,B (or INPUT A,B,C,D for four values). BASIC then expects as many values as you have separated by commas and waits for you to enter all of them. When the program expects two values, for example, you enter them like this: 10,20 and ENTER.

INPUT N simply puts a question mark on the screen. You may not know what is expected or what the value is for (it could be one of several values.) The INPUT statement may be combined with a form of the PRINT statement so that you have an intelligible prompt to follow. Its form is: INPUT "Enter something for value A here";A Now, instead of simply stopping and printing a question mark, the program will print everything between the quote marks on the screen along with the question mark. In our case, it will print: Enter something for value A here? The question mark is decidedly not needed here, since we are not asking a question, and there is a way to get rid of it. With this form of the INPUT statement however, we must live with it.

What happens if you input the letter A when the program asks for INPUT A? Since variable A has not been defined as anything else, it is assumed to be an integer value. If you answer this input prompt with the letter A, BASIC will return a message telling you to "Redo from Start." As you can see, INPUT can be very selective and fussy. If you want to input the letter A, you would have to say INPUT A$. Then the string variable A would be stored in memory location A$. As a matter of fact, you can input your entire name, including spaces, to the INPUT A$ statement. A$ will then contain whatever you typed in, even if you had entered integers (they will now be assigned as strings though.)

LINE INPUT is another form of the input statement. This one does not print the bothersome question mark, and allows you to enter any character or punctuation mark (except the Enter key.) With LINE INPUT A$, you can enter anything you want up to 255 characters long, and it will be assigned to A$. When using LINE INPUT with a prompt, as in LINE INPUT"Enter your name";A$, it is wise to leave a space before the last quote mark to give at least one space separation between the prompt and your reply. Otherwise, it might look like: Enter your nameHenry. Because you can enter anything for a LINE INPUT statement, you can include commas without getting the Extra Ignored message from BASIC.

Another form of input is the function INPUT$(N), where N is an integer from 1 to 255. This function allows you to type in N characters without showing them on the screen. It might be used for passwords, or in entering telephone numbers where it could provide some error checking immediately upon input (based on the fixed length of the numbers.) The other feature of this function (note that this is a built-in function and not a statement or command) is that the Enter key need not be pressed after the required number of characters is input. This function does it for you.

Yet another form of input is with INKEY$. It is used to read a character from the keyboard every time the function is executed. Yes, this is another function, and not a command or statement. It normally does not stop the program flow and is usually used inside a loop which calls itself until some character (any key) is pressed. INKEY$ can be used for many interesting and useful routines. Because it is normally used in a loop, you can write code to *exclude* characters you do not want input, thereby restricting input to only those characters which are permissible.

Here is a short program you can try to see how INKEY$ works:

```
10 A$=INKEY$:IF A$="" THEN GOTO 10
20 B=ASC(A$)
30 PRINT "YOU PRESSED ";A$;
40 PRINT " ITS ASCII VALUE IS";B
50 GOTO 10
```

It will print the character you pressed and its ASCII value. ■

Puzzler

Our Puzzler in Issue 2 attracted some attention. Thank you all for sending in your solutions. Many of you suggested that the problem was not too well stated. In looking back, we find that it may not have been, but then it was a difficult question to ask without giving away the answer.

Only two of you found a way to solve the problem; both of you did it essentially the same way and yet different from the solution we had for it. Coincidentally, both of the correct solutions were postmarked on the same day!

Edward Green of Naperville, Illinois, did it as in listing 1 (we took out his error trapping to lay the problem bare.) His solution asks for the variable letter, changes it to X$ and then *inside the loop* equates that letter to I with a series of IF statements. It does exactly what we had intended.

```
100 REM ED GREEN NAPERVILLE, IL
110 A=1:B=2:C=3:D=4:E=5
120 INPUT"WHICH VARIABLE DO YOU
   WANT TO CHANGE";X$
130 FOR I=1 TO 10
140 IF X$="A" THEN A=I
150 IF X$="B" THEN B=I
160 IF X$="C" THEN C=I
170 IF X$="D" THEN D=I
180 IF X$="E" THEN E=I
190 F=A+B+C+D+E
200 PRINT A;B;C;D;E;F
210 NEXT I
```

Program listing 1

William J Pottberg of Burlingame, California, also did it this way except he put the IF statements outside the loop as a subroutine and put a GOSUB inside the loop. Both work and both accomplish the same thing.

```
100 REM THE CODEWORKS VERSION
110 A(1)=1:A(2)=2:A(3)=3:A(4)=4:
    A(5)=5
120 INPUT"VARY WHICH ITEM";X
130 FOR I=1 TO 10
140 A(X)=I
150 F=A(1)+A(2)+A(3)+A(4)+A(5)
160 PRINT A(1);A(2);A(3);A(4);
    A(5);F
170 NEXT I
```

Program listing 2

Our solution is shown as listing 2. We never even thought of trying strings, but used subscripted variables. Both methods can get lumpy if many items are to be selected to vary. The string method seems to be cleaner for few items, but ours has the ability to read from an array. We are happy to see another way to do it and will incorporate the idea into something for future issues.

Those who wrote in with the incorrect solution were generally trying to change the STEP or the FROM TO values, not the loop counter itself. Thank you all again for your participation. If puzzlers turn you on, we can come up with one in every issue, some which don't require a written reply to us. Now, on to Puzzler 2.

Puzzler 2

What will the following code do? Will it even work, and if yes or no, why?

```
10 A=1:B=1:C=2
20 X=A=B:Y=B=C
30 PRINT X,Y
```

Try to figure it out before you type it in and try it. For the answer, turn the page upside-down.

Answer: Most Microsoft BASIC's return a -1 for a true statement and a zero for false. Your answer should have been -1 and 0. The first equal sign in line 20 is an assignment of equality, the second equal sign then becomes a logical operator which says that A and B are equal (which is true), and in the second part of line 20 it says that B is equal to C (which is not true.) You don't see this used that much, but it is legal and proper. It would probably be more apparent if line 20 had read: X=A<>B:Y=B<>C. That way though, the answer would have been 0 and -1 instead of -1 and 0. ■

Calculating Accuracy

Obtaining the precision they left out

Raymond L. Murray, Ph.d. and Nancy K. Reid,
North Carolina State University

Most BASIC's, even though they boast of double precision, don't have it when working with exponentials, powers and logs. This article tells how to get it.

Beginning computer users are often nonplused by the inaccurate values of built-in functions such as the exponential, logarithmic and trigonometric functions, as well as the important process of exponentiation. The manuals do not reveal that the answers are in single precision (6 significant figures) even if the variables are declared double precision (16 significant figures). Having need of accurate values of several of the mathematical functions, and finding that double precision subroutines did not allow powers of numbers other than 1/2, we developed a double precision program (see listing) to calculate and independently check EXP(N), LOG(N), and N↑X. The latter is obtained by the algorithm

$$N\uparrow X = EXP(X*LOG(N))$$

The exponential is calculated by summing the terms in the standard formula

$$e^x = \sum_{n=0}^{\infty} z^n/n$$

but employing ratios of successive terms to avoid having to raise the argument to a power, a single precision operation. Advantage was taken in programming of the fact that $e^{-x}=1/e^x$. In order to avoid slow convergence of the exponential series for large arguments, the product theorem was used in the form $e^z=(e^{z-5})(e^5)$. Thus $e^{16.5}=(e^{1.5})(e^5)^3$.

The natural logarithm was calculated by the best of several series appearing in *Handbook of Mathematical Functions*, Edited by Milton Abramowitz and Irene A. Stegun. Dover Publications (Equation 4.1.17, page 68).

$$\ln z = 2 \sum_{n=0}^{\infty} [(z-1)/(z+1)]^{2n+1}/(2n+1)$$

This formula is useful for either small or large z, but it is convenient to apply the same expression for values less than and greater than 1, noting that $\ln(1/z) = -\ln z$. To avoid very slow convergence of the logarithm series for large arguments, the addition theorem was used in the form $\ln z = \ln(z/5) + \ln 5$. Thus ln 1000 would be calculated as ln 1.6 + 4ln 5.

The logarithm and exponential subroutines are then used to calculate powers of numbers, noting that if $y = n^x$ then $\ln y = x \ln n$. A distinction is made between numbers that are decimals or quotients such as 22/7.

In calculating series, a term size limit of 10^{-17} was used. Since the upper and lower limits on numbers that most Basic's can handle are approximately 1.70×10^{-39}, stops were placed on INPUT numbers, viz.,

N<6D + 37	for LOG(N)
N<86	for EXP(N)
ABS(X)<86/ABS(LOG(N))	for N↑X

The user who attempts to find the logarithm of a negative number is told "redo".

Our program is generally slower than that of most computers in the calculation of functions that the programs have in common. However, we can obtain more significant figures — 15 or 16 versus 14 or 15. The optional check is based on the inverse relationships of the three functions, i.e., if Y = EXP(N) then N = LOG(Y) and vice versa, and if Y = N↑X then N = Y↑(1/X).

The extension of these methods to other functions used in mathematical analysis and calculations is straightforward. For example, the exponential integral function $E_n(X)$ uses both the exponential and the logarithm and applies recursion formulas to go from lower order to higher order functions. ●

```
100 CLS:REM CLEAR THE SCREEN
110 PRINT" LEAP IS A PROGRAM TO CALCULATE, TO DOUBLE"
120 PRINT" PRECISION, NATURAL LOGARITHMS, EXPONENTIALS,"
130 PRINT" AND POWERS OF NUMBERS (LOG, EXP, AND POWER)."
140 PRINT" BY RAYMOND L. MURRAY AND NANCY K. REID"
150 DEFDBL B-H,L-Z
160 DIM T(51)
170 E = 1D-17
180 CK = 0
190 PRINT
200 PRINT" 1) N^X"
210 PRINT" 2) LOG(N)"
220 PRINT" 3) EXP(N)
230 INPUT "SELECT PROGRAM : CHOOSE 1,2 OR 3"; Q
240 PRINT
250 IF Q = 1 GOTO 300
260 IF Q = 2 GOTO 760
270 IF Q = 3 GOTO 1300
280 GOTO 200
290 REM ************************************************
300 REM PROGRAM FOR POWERS OF N
310 PRINT "BASE MAY BE ANY POSITIVE DECIMAL NUMBER OR QUOTIENT,EXCEPT 1"
320 PRINT "N AS A DECIMAL (1)"
330 PRINT "N AS A QUOTIENT (2)"
340 INPUT "SELECT TYPE OF N : CHOOSE 1 OR 2"; C
350 PRINT
360 IF C = 1 GOTO 390
370 IF C = 2 GOTO 400
380 GOTO 310
390 INPUT "ENTER N AS A DECIMAL"; N : GOTO 430
400 INPUT "NUMERATOR OF N ="; NUM
410 INPUT "DENOMINATOR OF N ="; DEN
420 N = NUM/DEN
430 N0 = N
440 IF N0<=0 THEN PRINT"ONLY POSITIVE BASES : REDO":GOTO 310
450 IF N0=1 THEN PRINT"BASE CANNOT BE 1 : REDO":GOTO 310
460 PRINT
470 PRINT "EXPONENT MAY BE ANY DECIMAL NUMBER OR QUOTIENT"
480 PRINT "X AS A DECIMAL (1)"
490 PRINT "X AS A QUOTIENT (2)"
500 INPUT "SELECT TYPE OF X : CHOOSE 1 OR 2"; R
510 PRINT
520 IF R = 1 GOTO 550
530 IF R = 2 GOTO 560
540 GOTO 470
550 INPUT "ENTER X AS A DECIMAL"; X : GOTO 590
560 INPUT "NUMERATOR OF X ="; NU
570 INPUT "DENOMINATOR OF X ="; DE
580 X = NU/DE
590 X0 = X    'CHECK INPUT POINT
600 XL = 87/(ABS(LOG(N)))
610 IF ABS(X0) > XL GOTO 630
620 PRINT:GOTO 820:'TO LOG PROGRAM AS SUBROUTINE
630 PRINT "EXPONENT OUT OF RANGE : REDO" : GOTO 470
640 LN = X * LG
650 N = LN
```

```
660 PRINT:GOTO 1370:'TO EXP PROGRAM AS SUBROUTINE
670 EX = S
680 PRINT N0;"TO THE POWER";X0;"=";EX
690 IF CK=1 THEN PRINT"SHOULD AGREE WITH YOUR ORIGINAL BASE"
700 IF CK = 1 GOTO 730
710 PRINT : INPUT "DO YOU WANT TO CHECK (Y/N)";A$
720 IF A$ = "Y" GOTO 1770
730 PRINT:PRINT
740 RUN 150
750 REM ******************************************************
760 REM PROGRAM FOR NATURAL LOGARTIHMS
770 PRINT "INPUT MAY BE ANY POSITIVE NUMBER < 6E+37"
780 INPUT "N ="; N
790 REM CHECK INPUT POINT
800 IF N<=0 THEN PRINT"ONLY POSITIVES: REDO":GOTO 770
810 IF N>6E+37 THEN PRINT"TOO LARGE: REDO":GOTO 770
820 Z = N
830 IF Q=1 THEN PRINT"NOW IN LOGARITHM SUBROUTINE"
840 Z0 = Z
850 IF Z = 1 THEN LG = 0 : GOTO 1090
860 IF Z <=.2 GOTO 880
870 GOTO 900
880 Z = 1/Z
890 H = 1
900 IF Z >= 5 GOTO 1000
910 GOSUB 1160
920 IF Q = 1 GOTO 640 : IF Q = 3 GOTO 1700
930 PRINT "LOG("Z0") ="; LG
940 IF CK = 1 THEN PRINT "SHOULD AGREE WITH YOUR ORIGINAL ARGUMENT"
950 IF CK = 1 GOTO 980
960 PRINT : INPUT "DO YOU WANT TO CHECK (Y/N)";B$
970 IF B$ = "Y" GOTO 1770
980 PRINT : PRINT
990 RUN 150
1000 U = 1.6094379124341#
1010 Z = Z/5
1020 M = M + 1
1030 IF Z > 2 GOTO 1010
1040 GOSUB 1160
1050 LG = M * U + LG
1060 M = 0 :REM RE-INITIALIZE
1070 IF H = 1 THEN LG = -LG
1080 IF Q = 1 GOTO 640 : IF Q = 3 GOTO 1700
1090 PRINT "LOG("Z0") ="; LG
1100 IF CK = 1 THEN PRINT "SHOULD AGREE WITH YOUR ORIGINAL ARGUMENT"
1110 IF CK = 1 GOTO 1140
1120 PRINT : INPUT "DO YOU WANT TO CHECK (Y/N)";B$
1130 IF B$ = "Y" GOTO 1770
1140 PRINT : PRINT
1150 RUN 150
1160 REM  SUBROUTINE TO CALCULATE LOGARITHM
1170 PRINT : PRINT "LOGARITHM CALCULATION IN PROGRESS"
1180 Y = (Z - 1)/(Z + 1)
1190 T(1) = Y : V = Y
```

```
1200 KM = 50
1210 FOR K = 1 TO KM
1220 PRINT "LOG LOOP #" ; K
1230 T(K+1) = T(K) * Y * Y * (2*K -1)/(2*K +1)
1240 IF ABS(T(K+1)) < E GOTO 1270
1250 V = V + T(K+1)
1260 NEXT K
1270 LG = 2 * V
1280 RETURN
1290 REM ************************************************************
1300 REM PROGRAM FOR EXPONENTIAL
1310 PRINT "INPUT MAY BE BETWEEN -86 AND 86, INCLUSIVE."
1320 INPUT "N =";N
1330 REM CHECK INPUT POINT
1340 IF N > 87 OR N < -87 THEN PRINT "N OUT OF RANGE : REDO" : GOTO 1310
1350 IF N > 87 OR N < -87 GOTO 1310
1360 IF Q = 3 THEN PRINT "CALCULATION IN PROGRESS"
1370 D = N
1380 IF Q=1 THEN PRINT"NOW IN EXPONENTIAL SUBROUTINE"
1390 D0 = D
1400 KM = 50 : L = 0 : W = 0 : I = 0
1410 IF D < 0 THEN D = -D
1420 IF D > 2 GOTO 1440
1430 IF D <= 2 GOTO 1490
1440 I = 1
1450 W = 148.4131591025766#
1460 D = D - 5
1470 L = L + 1
1480 IF D > 1 GOTO 1440
1490 T(1) = 1 : S = 1
1500 REM -------START OF LOOP
1510 PRINT : PRINT "EXPONENTIAL CALCULATION IN PROGRESS"
1520 FOR K = 1 TO KM
1530 PRINT "EXP LOOP #" ; K
1540 T(K+1) = T(K) * D/K
1550 S = S + T(K+1)
1560 F = T(K+1)
1570 IF S = 0 GOTO 1610
1580 IF D <> 0 GOTO 1600
1590 IF D = 0 GOTO 1630
1600 IF ABS(F) <= E GOTO 1630
1610 NEXT K
1620 REM -------END OF LOOP
1630 IF I = 1 GOTO 1650
1640 IF I <> 1 GOTO 1680
1650 FOR A = 1 TO L
1660 S = S * W
1670 NEXT A
1680 IF D0 < 0 THEN S = 1/S
1690 IF Q = 1 GOTO 670
1700 PRINT "EXP("D0") ="; S
1710 IF CK = 1 THEN PRINT "SHOULD AGREE WITH YOUR ORIGINAL ARGUMENT"
1720 IF CK = 1 GOTO 1750
1730 PRINT : INPUT "DO YOU WANT TO CHECK (Y/N)";C$
```

```
1740 IF C$ = "Y" GOTO 1770
1750 PRINT : PRINT
1760 RUN 150
1770 REM ************************************************************
1780 REM CHECKING SUBROUTINE
1790 CK = 1 :REM INDEX OF CHECK
1800 PRINT"CHECK:  ";
1810 IF Q = 1 GOTO 1840
1820 IF Q = 2 GOTO 1880
1830 IF Q = 3 GOTO 1900
1840 X = 1/X0
1850 N = EX
1860 N0 = N
1870 GOTO 590
1880 N = LG
1890 GOTO 1330
1900 N = S
1910 GOTO 790
1920 END
```

Programming Notes

Screen addressing is an important part of computing. There is no standard, although there are two popular methods which have gained acceptance. BASIC running under MS-DOS generally uses the LOCATE R,C command, where R is the row and C is the column. This command will position the cursor at the screen location called for by the LOCATE. The next PRINT statement (even if it is several lines of code away), will then print at that location. The LOCATE positions start at 1,1 which is the upper left corner of the screen. An attempt to LOCATE at 0,1 or 1,0 will give a "Position not on Screen" error.

The other popular positioning format is the PRINT@. This positioning device is used like this: PRINT@721,"whatever". This one is used on all the earlier Tandy models, as well as MBASIC which runs under XENIX. The counting scheme here runs from 0 through 79 (assuming an 80-column screen), and position 80 then becomes the first print position on the second line.

You will undoubtedly run into programs printed in some magazine or other (even this one) using one or the other of these print locating conventions. If you are not used to seeing one of them it will look foreign and forbidding, but don't let that stop you. There is a way to convert from one form to the other.

To change LOCATE Row,Col to PRINT@: PRINT@ will equal (Row—1)*(Col—1).

To change PRINT@ to LOCATE Row,Col: Divide the PRINT@ number by 80 and add 1 for the Row. Add one to the remainder for the Col.

That's all there is to that. You might even write a little program to do the math for you, and even have it print out a list of the line numbers and what needs to be changed to what. If you are writing a program that needs to run with both conventions you can write a subroutine that does it and simply change one variable at the start of the program to tell it which machine it should convert for.

The above all assumes that we are converting from one 80-column screen to another. What if you are trying to convert from a 64-column machine to an 80-column one? It will still work, but you will not be utilizing all of the 80-column screen space.

Going from an 80-column screen to a 64-column one is another matter entirely. Mostly because it simply won't fit. In this case you would need to juggle things around to try and make everything fit on the smaller screen. The 64-column machines usually only have 16 lines, and that makes matters even worse.

Some of the MBASIC's that run under CP/M don't even have a PRINT@ or a LOCATE. They use an old convention for terminals that uses an ESCAPE +Y sequence to unleash their cursor. It is so bulky to use that it is normally set up as a user defined function (DEF FN) at the beginning of the program, then called with the print location coordinates.

Download

The CodeWorks Download System

Before we get into using the system, we ought to explain what the system is *not*. It is not a bulletin board system (BBS). It does not have the ability for people to carry on bulletin board type conversations. It is not an electronic mail system. You cannot send messages to other subscribers and your ability to tell the editor anything is limited. It is not an electronic information service. You won't find news, sports or weather here.

The CodeWorks download system is a system with limited capability to feed back information to the system operator. As it starts operation (full service operation will occur on or about the 1st of January 1986), we may have some problems, so bear with us. Any new software invariably has its faults.

The System

The system was designed and developed in the C programming language. It runs under the XENIX operating system and uses its services, but it does not use the normal XENIX access control or utilities.

In designing the system, a lot of hooks have been left in for special features and enhancements which can be added in the future.

If there is enough interest in telecommunications, followup articles on the system and telecommunications in general will include such things as basic system design, terminal programs to call the system with, automating your interaction with the system, etc. Let us know what you would like to see.

Using the System,

The download system has two sections. For those who are not subscribers there is a limited demonstration section available which allows a preview of the magazine and the download features. If you are not a subscriber but want to become one, you can sign up while here or ask for a sample issue of the magazine.

The major section of the system is for subscribers and provides for issue by issue downloading of programs and updates or corrections.

The system is menu driven. This means that everything you can do will be controlled from a menu of one sort or another. This greatly simplifies use of the system for beginners. An advanced user mode is available to subscribers which allows use of the system without the menu displays. Subscribers who are interested can learn about this mode on line.

In order to get on the system, you first have to get yourself set up on your computer and call us. You should set your computer to **8 bits, No parity, 1 stop bit, full duplex and 300 baud.** Download at 1200 baud is not available now, but it will be later if there is enough interest in it.

Once you have set up your computer, call (206) 475-2356. If the computer line is up, it will answer the phone. It will be unavailable sometimes (backups in progress, new software setups, etc.) but it will be up normally within a few hours. Since it runs unattended, the system could crash during an evening or weekend without anyone being able to restore it. In that case you may have to wait for the next business day before it will be up. If you find the number continually busy, let us know because that will tell us it's time to add another dial-up line.

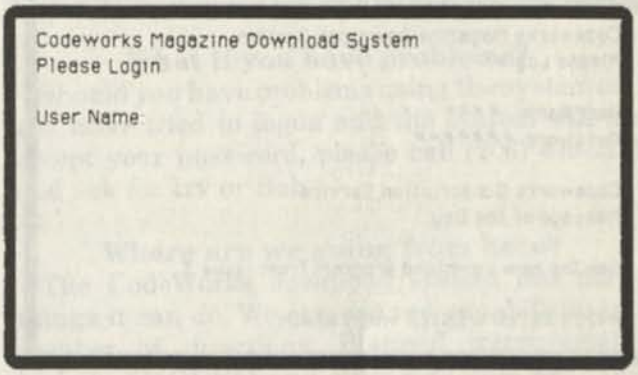


Figure 1

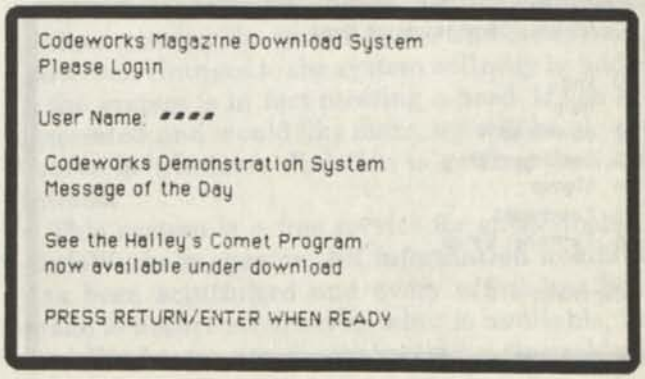


Figure 2

Once you have an answer from the system, your screen should look like that shown in figure 1. If you are logging in to use the demonstration system, type DEMO (upper or lower case, it doesn't matter) and press RETURN or ENTER. You will notice that your letters echo back only as pound signs,(#) so no one can look over your shoulder and see what you have typed. You will be welcomed to the system and then see a "Message of the Day" (MOTD) with information of interest (figure 2). Finally, you will be placed in the demonstration menu system which we will go into in a moment.

If you are a subscriber and want to use subscriber function, then you should enter your subscriber name and number (the way it appears on your mailing label) as one word and press RETURN. For example, if your name is "Sampson" and your subscriber number is 1325, then you type: sampson1325. Use upper or lower case or mixed, it makes no difference. Your subscriber number, by the way, is near the upper right of your mailing label.

Next you will be prompted for your password. If this is your first time on the system, the password you type in will be registered in the system as your normal password. You will be asked to type it in a second time for verification. Remember your password! If you forget it we cannot give it to you because it is encrypted by the system and even we can't get to it.

Once you have entered your password, you will be welcomed to the system and see the message of the day. Finally, you will be placed in the subscriber menu.

The Demonstration menu

The demo menu looks like figure 4. Each option available to you has a number before it. To choose that option, type the number and press RETURN or ENTER. Let's go over the options quickly to see how they work.

0. END(logout) The logout option will sign you off the system and hang up the phone. You should then hang up your phone.

1. Help The help option will display one or more screens of information about the operation of the system. You can leave the screens by typing the ESCape character. Pressing RETURN or ENTER will continue cycling the screens until you have seen them all.

2.Download The download option allows you to access the demo download menu. You will be given a choice of files available for download (figure 5). You select one by typing its number and pressing RETURN or ENTER. The system will retrieve the file and ask you to press RETURN or ENTER when you are ready to receive it. While the system is waiting, you should prepare your terminal

```
Codeworks Magazine Download System
Please Login

User Name: ****
Password ********

Codeworks Subscription Service
Message of the Day

See the new download program from issue 3

PRESS RETURN/ENTER WHEN READY
```

Figure 3

```
Codeworks Demonstration Menu

0  End
1  Help
2  Download
3  Send me more
4  Signup
5  Comments
6  Terminal Setup

Command =>2
```

Figure 4

```
Demo Downloads Available

0: End
1: halley.bas     Halley Comet Program
2: test.bas       Test Download Program

Command =>
```

Figure 5

```
Demo Downloads Available

0: End
1: halley.bas     Halley Comet Program
2: test.bas       Test Download Program

Command =>2
PRESS ENTER/RETURN WHEN READY
10 REM test program one
20 PRINT "HELLO CODEWORKS SUBSCRIBER"
PRESS ENTER/RETURN WHEN READY
```

Figure 6

program to receive the download and then press RETURN or ENTER. After the download is complete, the system will again stop and wait for you to press RETURN or ENTER. At this time you should terminate the download function on your computer.

3 Send me more (ask for info) This option will ask you for your name and address so we can send you a sample issue. Simply type in the information when prompted.

4 Signup The signup option will take your name, address and credit card information if you would like to become a subscriber. Only VISA and Master Card are accepted. There are no "bill me's". If you wish to subscribe and not use your credit card you will need to write directly to us.

5 Comments Demonstrations users are allowed to enter up to two lines of comments about the system.

6 Terminal setup The Default terminal characteristics for all users of the system is 80 character lines and 24 line screens. If you have a different size screen, you can use this option to set its size. This will only apply to this session however.

```
Codeworks Subscription Menu

0  End
1  Help
2  Download by Issue
3  Letters to the Editor
4  Change Password
5  Terminal Setup
6  Expertise Level

Command =>2
```

Figure 7

The Subscriber Menu

Subscribers have similar but more versatile options available to them once identified and logged in. The subscriber menu looks like that shown in figure 7 with the following options:

0 End (logout) The logout option is just like that for demo users, it signs you off and hangs up on you.

1 Help As with the demo menu, help lets you look at information about how to use the system.

2 Download The download section for subscribers is far more versatile than that for demo users. Instead of being put into a demo area, you will be given a list of issues to choose from. Choosing an issue by number will display the items available for download from that issue. As with the demo system, you can download a file by choosing it and loading it into your computer.

3 Letters In the letters section you can enter up to 20 lines of feedback to the editor about anything of interest. Items which are deemed appropriate will appear in the "Forum" section.

4 Password The change password section will prompt you first for your current password (to safeguard you, should you leave your system unattended while connected) and then your new password (with one repeat to be sure it is correct). This will be stored as your new password for access to the system.

5 Set Terminal As a subscriber, you can set your terminal parameters permanently or for only the current session. All users are created with default parameters (24 line screen, 80 column width), so be sure to set yours if that doesn't match. You can also set your terminal type from among those available. Typing HELP will display the list of terminal types.

6 Set Expertise There are three levels of expertise for subscribers: novice, normal and expert. In the novice mode, all menu displays are shown in full detail. In the normal mode you will see only the commands available, while in the expert mode you will be prompted for the command only. You can change your mode at any time by typing "novice" or "normal" or "expert". Using the expertise command will allow you to make your level permanent.

What if you have problems?

Should you have problems using the system or if you have tried to logon and the system will not accept your password, please call (206) 475-2219 and ask for Irv or Bob.

Where are we going from here?

The CodeWorks download system has many things it can do. We can expand capabilities in a number of directions. Planned improvements include XMODEM and other protocols available for downloads, auto-login sequences for special programs, special programs for your computer to make accessing the system easier and much more. However, changes to the system will only be added if the system is in fact meeting a need. If you are interested and would like more, we will be glad to evolve in whatever direction is getting the most interest.

This system is a free service for subscribers to CodeWorks magazine. All information available has been scrutinized and every effort has been made to assure accuracy in what is available, but CodeWorks accepts no responsibility for problems using the system. ●

Wood.Bas

An example of "best fit" programming

Staff Project

This is a rather long and involved article, best started on a long winter night. It traces the development of the program from start to end. Even if you are not a woodworker, watching the program go through the attempts to find a best fit is fascinating.

At the core of most programs you will find a rather small bit of code that is a formula or an algorithm surrounded by bunches of housekeeping code. That formula or algorithm, of course, is the heart of the program. But what do you do when there is no formula or algorithm? This is an example of just such a problem.

There exists a class of problems which have no defined solution other than a brute force one of trying all possible combinations until a best fit is achieved. If the number of possibilities you are considering is small, say four to five or less, then trying all the combinations is workable. But what happens when the number of items is ten or more? The number of possibilities of rearranging n items is n factorial. With just ten items the number of ways to rearrange them is over three million, six hundred thousand!

Assuming that you could eliminate at least half of these possibilities out of hand, you still wouldn't have enough space in memory to keep even a single digit to signify which of the attempts was better than another. Aside from that the time required to make all those attempts would be prohibitive.

The type of problems we are concerned with here are generally called "best fit" problems. An example of it would be the "knapsack" problem. In this hypothetical problem a hiker is preparing to climb a mountain. He has a knapsack with a defined capacity and various items he can take with him. The items vary in size, weight, and importance to his survival. He is limited by both size (the capacity of the knapsack) and weight (how much can he carry safely.) What is the best combination of items to take?

Our problem has to do with the cutting of wood from plywood sheets. (It could just as well be used to cut cloth material or parent sheets of paper stock, but we will concern ourselves with the plywood situation.) The sheets are normally 48 inches wide and 96 inches long. The grain of the veneer runs parallel to the length of the sheet. Many different hardwood veneers may be obtained, and the veneer may be on one side only (called type A1), or on both sides (called type A2.) Most lumber dealers will not cut a full sheet if you only need part of one so you must purchase a full sheet when only a fraction of one is needed. Most of the better grade hardwood sheets cost from about $50 to $65 per sheet. Making mistakes at these prices can be costly. Our problem, and programming project, is to design a program that will take any given woodworking project and produce a detailed cutting diagram using a best fit method that does not use brute force and which also keeps track of the grain of the wood. In addition it should also take saw kerf into account and automatically reduce the size of cutoff pieces by the kerf, but only on the edges being cut.

It may surprise you to know that after all the years people have applied saws to wood there is no rule or formula to determine if n pieces of wood with varying dimensions of x and y can be cut from a given larger piece.

Preliminary Planning

Without a rule to follow it seemed the only way to start was to ask how one would do it manually. You probably start by sketching the finished project on a pad and then deciding on overall height, width

and depth. This is usually followed by determining the size of each individual piece, taking into account any dados, miter corners and irregular shaped pieces. Then follows the drawing of a rectangle twice as long as wide and trying to fit your required pieces into it (or them), all the while keeping track of the desired grain of the wood. You can wear out an eraser very quickly doing this and finally decide that this is as good as it is going to get. So you head for the shop and start cutting. If you have made even a small mistake it can cost you another sheet or two of expensive wood and another trip to the lumber yard. The less fastidious of us, at this point, would probably change the original design to fit the mistakes and the available stock wood.

The way we handle this problem manually is an interesting process. Most of us probably don't even think about it that much - we just do it - and usually back ourselves into a corner. We learned a lot about our own thinking processes as we went ahead with this project. It was both exciting and frustrating.

To start we decided to apply the most simple logic we could think of. We knew we would need at least three arrays: one to hold the dimensions of the required pieces, one to hold the stock pieces, and a third to act as an output buffer to hold both the required piece and the stock piece it came from along with a cut code of some sort. That third array would be necessary to hold the information until the time came to actually print the cutting diagrams on paper.

The logic was suggested by Terry Dettmann and went like this: Take the largest piece you need first and cut it from the smallest available piece and put the leftover pieces back into stock. Then do it again with the next largest piece. With that sage advice, Terry left it up to the rest of us. It made sense, and we tried it and it worked, but it needed a lot more.

At this point we had not considered grain at all. It also became apparent that once a cut is made in any direction on a sheet of wood with a circular saw, that it must continue all the way across the length (or width) of the piece. To illustrate: starting with a new sheet and a requirement for a 70 inch long and 22 inch wide piece leaves you with a choice of which way to cut first. If you cut across the width first you have two pieces left over, one 26 by 48 and the other 70 by 26. If you make the horizontal cut first you leave two pieces again, this time a 96 by 26 and a 26 by 22. Obviously, if there is only one cut to make you simply must live with whatever is left over. But the two-cut problem loomed as a major obstacle, since it became a critical factor in determining whether or not the remaining required pieces could be cut from the sheet.

Since the type of cut needed to be identified, we coded all the possibilities, including a no-cut situation as follows:

Cut 1 - exact fit
Cut 2 - one cut, horizontally
Cut 3 - one cut, vertically
Cut 4 - two cuts, 1st vertical, 2nd horizontal
Cut 5 - two cuts, 1st horizontal, 2nd vertical.

The next question we ran into sounded ridiculous. How do you define "big?" You can use length, width, square area, semi-perimeter, hypotenuse or the ratio of one side to the other. In our first attempt, we tried all of these, with interesting results. Ratio tells more than you want to know about the shape of the piece but loses length and width information. Hypotenuse didn't help at all. It turned out that on some groups of pieces to cut the "big" question didn't make much difference. On others it made subtle differences and on yet others the difference was significant.

It became increasingly apparent that not only was the type of cut chosen important, but so was the *order* of the pieces. Determining "big" with square area seemed the most logical choice, but it turned out to be a dud. In the end, the choices of ordering that had the highest success rate were simply length and length plus width (semi-perimeter.) In the final version of the program we use both, and additionally, sort width within length. It increased our success rate by five to six percent.

To measure effectiveness of different methods we created a random piece generator and tacked it onto the front of the program. This allowed us to leave the computer unattended for hours, or even overnight, and the line printer would spew out all sorts of pertinent data for each attempt. The data was then scrutinized, especially the failures, for some common factor we could then trap. We found only that there were not too many common factors.

How close is close enough?

It's a lead pipe cinch that if the sum of the square area of all the required pieces is larger than the square area of the piece from which they are to come, it will not work. That's simple enough, but what if the total square area of required pieces is less? Will it work every time then? The answer, of course, is that it may or may not work. Some combinations of pieces, even with two pieces and less than a third of the square area of the stock piece, are clearly impossible. Others, with a total square area exactly equal to the stock piece and as many as eleven required pieces, fall apart on the first attempt.

We decided to define success like this: when the total square area of the required pieces is less than the stock piece and the program finds the way to cut those pieces without resorting to an additional sheet. Additionally, we defined it a success when the cut or fit is clearly impossible and an additional sheet is required. This left many attempts undetermined as to success or failure, with no definite way to tell if the problem was solvable and the program just couldn't do it, or the problem had no solution. In the final program when this occurs it lets the operator know that the solution is undetermined. But is it a success or a failure? We have yet to come up with a way to tell.

Program notes

The program starts out by setting the loop counters as integers which significantly increases the speed of execution. The three arrays, mentioned earlier, are established next. They are the required piece array R(x,x), the stock piece array S(x,x) and the output array O(x,x,x,x,x). The R array is counted with integer I and its terminal value is NR. The S array is counted with J and its terminal value is NS. The O array is counted with K and its terminal value is NO. The R(50,3) simply states that the array can have up to fifty elements, each containing three positions. Length, width and grain direction will occupy the three positions. In the S array we only need two positions since the stock is always carried with the same grain orientation. The O array needs five positions, two for the stock piece length and width, two for the required piece length and width and one for the cut code. Notice that the grain orientation is ignored after the first few lines of the program, but more on that later.

The next several lines print the heading of the program and instructions on the screen. Kerf, or no kerf, may be selected at this point. In line 320 the length and width of the stock sheet is initialized. From here on we will call the length of a ply sheet LA and the width LB. This way you can change these values to fit paper or cloth, or even some of the plywood that is 4 by 12 feet instead of 4 by 8 feet.

During input of the required pieces extensive error trapping occurs. This is necessary because if you could ask for a piece longer than LA the program would continue to pull in new sheets indefinitely, looking for one that fits.

In line 420 we solve the whole problem with grain. At this point, if grain 2 (parallel to width) was specified, we simply switch the dimensions of the piece and act as though it were a grain of 1. Now, since all grain is 1, we can forever forget about which way the grain runs. That really sounds illegal, but it isn't. It just results in some lengths being less than width. If that really bothers you a lot you can simply call the dimensions X and Y instead of length and width. It all comes out right in the end.

Later on in the program we will establish a rule that says we will make attempts at the problem that equal two times the number of pieces plus one. The attempt counter is called FL. SQ will be the variable that denotes the total square area of the required pieces and LF is derived from SQ and tells us how many sheets should be required. LF is the "expected sheet counter." These three variables are all initialized to zero in line 440.

The very next thing that happens is that the R array is scanned from beginning to end and the total square area of the required pieces is determined and set in variable SQ. This happens in lines 460 to 480. In line 490 we find out what the minimum number of sheets will be. The expected sheet counter, LF, will hold this number. As mentioned earlier, you can't buy less than a full sheet, so the equation in line 490 produces the next highest integer if the square area is even a small fraction over that of a standard sheet.

Sort into descending order

In lines 500 through 620 we do the first sort of the required pieces. The same sort routine is used to sort two different ways, depending upon the state of the attempt flag, FL. The very first time, the attempt number is zero, so we fall through line 540 and sort on length plus width. On the second attempt, we will jump around the length plus width line and sort on length only. What happens on subsequent attempts? We don't care since the program never comes back to this point after the second attempt.

(For more detail on how these sorts work see Issue 2, page 26. They are simple bubble sorts and are described fully in that article. Also note that if your version of BASIC supports the SWAP command you can make good use of it in this program in every case where variables are exchanged. As listed, the code will work even if you have the SWAP command and choose not to use it.)

Sort width within length

Now that the list is in order by length (or length plus width) we want to sort widths within lengths. This is not really difficult to do. It happens in lines 630 through 720. If length X is not equal to length X+1, we ignore the switching in lines 680 and 690 and go on to X+1 and X+2. What this says is that only if two lengths are equal do we look for width.

The next two sections of code from lines 730 through 810 come into play only on subsequent attempts when FL is 2 or more. We are still on attempt 0, so let's jump around these routines and come back to them when they will be used.

Compare required to stock pieces

Starting at line 920 we initialize the output array counter (K) to 1. We then start a very big loop that finally ends way down in line 1560. All sorts of things happen inside this "Big I" loop and it may be well to remember that during each pass through this loop we are taking a required piece and trying to find a piece of stock from which to cut it. We will also get a new sheet if there is no stock piece that fits, and determine the way to cut it and tuck the cutoff pieces neatly back into the stock array (minus the kerf when and where necessary.) Before we leave the loop we will also sweep the sawdust off the floor. (Oops, that means eliminate zero length or width pieces from the stock array.) At the same time, we will sort the stock array into ascending order. The output buffer (O) will be stuffed with all the appropriate dimensions and cut codes and finally we will go back to get the next required piece. That's just an overview of what happens inside the "Big I" loop, now back to line 940.

The "Big I" loop starts at line 940. It takes one piece at a time and runs it through the mill. The next thing we do is set the stock array counter (J) to 1. Remember that NS is the terminal counter for the stock array? Well, it's at zero now because there is no stock in the array. We do lots of things in line 870. Let's look at them.

First, J is greater than NS because NS is zero and we just set J to 1. This makes the remainder of the line valid. The first element of the stock array now receives a brand-new sheet of wood, full size and ready to go because S(J,1) is equal to LA (which is 96) and S(J,2) is equal to LB (which is 48). NP is a new variable whose only purpose in life is to keep track of the number of new pieces we haul in. Since we just got one we increase NP by 1 (it was zero to start.) Now, since the stock array has one piece in it, we must increase the count in NS, since NS is supposed to tell us how many pieces are in stock. Variable V is set equal to 1 at this time because we just got a new sheet.

You may well ask why we need V when NS apparently does the same thing. The reason is that NS is an accumulating counter, while V just goes "BANG" when a new sheet arrives and then dies later. Variable V will haunt us a little further down the road. Now the program tells us to go back to line 950.

But we just came from here, no? Yes, and we just set J equal to 1 again too, but this time when the program comes to line 970, J is not greater than NS (equal, but not greater) so it ignores the remainder of line 970 and goes on to line 980.

Line 980 is a decision line. It says that if the actual number of sheets used is greater than what we expected (LF) and the number of attempts at this point is not equal to two times the number of required pieces plus one, to go somewhere else. Right now neither of these are true statements, so we drop right through them to the next line which just happens to be another decision line.

Line 990 says that if we have not just pulled in a new sheet (which is not true now) or if the attempt counter is 4 or more (which also is not true now) we should go to line 1080. We will get back to 1000-1060 later when they come into play, but for now let's see what's happening at 1080.

Check for fits

Starting at line 1080 and continuing through line 1180 we compare the length and width of the piece we want to cut to the length and width of the available stock.

You may wonder why, all of a sudden, we are counting the stock array (S) with a U counter instead of J. Remember that we are still inside the "Big I" loop, and within that loop we are within the J loop. Yes, the J loop is real even though there is no FOR...Next statement for it. It is a homemade loop so that we can increment and decrement the counter at will and jump in and out of it whenever we want to. In this case, we use U to count the S array so that if we get all the way through the comparisons without finding what we want, the J value will still be intact for the next pass. Otherwise, if we find the piece we want in stock we will use it by setting J equal to U and then jumping out of the loop.

Let's put an actual example into the R array. Let's say the first required piece is 72 inches long and 22 inches wide. As shown earlier, we just pulled a new 96 by 48 inch piece into stock. Now let's go through the comparison routines.

If we can find a piece in stock that exactly fits what we are looking for it should get highest priority, right? So, in lines 1080 and 1090 we look for exact fit. If we find it, U will tell us which piece in the array it was, so we set J equal to U and go on to the following routines at 1200.

This was not the case in our example piece though. An exact fit was not found, so the next few lines look for equal length and enough width in the stock piece to allow us to get our required piece. Again we don't find it because 72 doesn't equal 96. The next few lines look for widths that are equal and a length that will be greater than what we need. Again, 22 does not equal 48, so we get to the last comparison.

The last comparison simply says to take anything that will fit. But don't let this fool you, we are going to give it every chance to make a better choice than just taking any old piece. Remember earlier when we said that we sort the stock array

into ascending order? Well, that comes into play right here. If none of the other comparisons work out, the next piece it will look for will be the smallest piece from which the required piece can be cut.

In our example case the comparison will work out in lines 1170 and 1180 because 72 is less than 96 and 22 is less than 48. But let's leave that for a minute and look at what would have happened had it not found a stock piece large enough.

Having gone through all the comparisons and not finding a piece large enough, line 1190 will increment J by 1 and go back to line 970 where J will once again be larger than NS and will cause a new sheet to be put into stock. (And if you can't get your piece out of a full sheet there has to be something wrong!)

Back to our example. In line 1170 we found that we could get a 72 X 22 piece out of a 96 X 48 inch piece. So we identify J with the place in the S array where we found the piece (at this point there was only one, but there will be more presently), and go to line 1210. Note that all of the comparison loops, if successful, go to line 1210. Note also that each of the comparison loops go through the entire stock array when looking for a fit.

At line 1210 we stuff the output buffer (O) with the length and width of the required piece and the length and width of the stock piece. That's four out of five. The remaining output buffer slot yet to fill is the cut code, which we will get to presently. Note that the K loop is another homemade loop (it counts the output (O) buffer array. Note also that when the comparison was found, line 1190 was not energized and so J still tells us where in the stock array our piece was found.

Let's examine lines 1220 through 1390. At this point we know the piece we want and the piece from the stock array it will come from. What we want to do now is to find out how to cut the stock piece and put any leftover piece or pieces back into the stock array. To keep from handling too many confusing array designations, line 1220 defines L as the length of the stock piece, W as the width of the stock piece, LC as the required piece length (length to cut) and WC as the required piece width (width to cut).

Suppose you are in a lumber yard and there is a series of bins, each with the space to hold one piece of plywood. The entire series of bins is labeled S and each individual bin is marked J1, J2, etc., to the last bin. Let's say that bin J1 has a piece of wood in it and the rest of the bins are empty. Since J1 is the last bin to be occupied, a sign reading "NS" is hung onto it. This, of course, is exactly what our stock array looks like at this point. (In our array, however, each bin would be subdivided into two sections, one to hold the width and the other to hold the length, which in the real world is rather impossible.) Now back to line 1230, where we will try to see if our piece of wood can be cut with only one cut.

Line 1240 says that if the required piece is exactly equal to the stock piece then we will simply remove the piece of wood from the stock array bin and leave it empty. We will then also designate the cut code (C) as 1.

Line 1250 says that if the length of the required piece is equal to the length of the stock piece only one cut is required and it will be parallel to the length. This leaves one piece of wood with the same length it had before, but with a width that is equal to its original width less the width of the required piece and less the saw kerf. In this case we will still have a piece of wood in stock bin 1, but it will have new dimensions. The cut code is set to 2 for this situation. Line 1260 performs the very same operation, only this time the cut is vertical, the width remains the same, the length changes and the cut code is set to 3.

None of the above fit our present example situation, so we get to line 1280, which says that if the attempt number is odd and we just pulled in a new sheet and the big I loop is at 1, then go to 1350 to make a 5 cut. We meet all those conditions except the attempt number (which is now at 0), so we ignore this line for now. We will also ignore lines 1290 through 1330 for now. They are special 5 cut situations, none of which apply to our example.

The 4 cut line

This brings us to line 1340, which is the line that determines the size of the two leftover pieces with a 4 cut. We are going to saw across the 48 inch width of our sheet at the 72 inch mark first. The piece left over from this cut will be put back into the bin where the 96 X 48 inch sheet came from (J1). Its length will be the original length (96 inches) less the length of the required piece (72 inches), less the saw kerf (SK). Its width will remain unchanged at 48 inches. This happens in the first two statements in line 1340. The next part of line 1340 says that J is now equal to NS plus 1. The NS sign was hung on bin one previously, and there is a piece of wood there, but now J is equal to 2, which is the second bin. Note that no matter how many bins are filled, the NS sign will always be on the last one that has something in it and the bin following that one will always be empty (making a good place to stuff our cutoff pieces of wood.)

Our second cut will be made on the 72 inch long, 48 inch wide piece left over from the first cut. This cut will be made at the 22 inch mark, leaving a piece 72 inches long and 26 inches wide (less the kerf.) The next part of line 1340 says that bin 2 will

hold this leftover piece, and the next thing we need to do is to move the NS marker to the second bin and set the cut code to 4. There are now two pieces of wood in stock; we have the required piece taken care of (it's in the first two positions of the output buffer. We put it there before the cut was ever made.) The last thing line 1340 says to do is to go to line 1360. Here, we finally stuff the cut code into the output buffer and move NO (the output buffer terminal marker) up one.

Show the man you are working!

The computer can do all of what has happened up to now in about a second or less. It's time to prove to the operator that we are earning our money, so let's show something on the screen. Line 1370 says that if V equals 1 (it does, remember that in line 970 we pulled in a new sheet and set V to 1) then print "New Sheet" on the screen. Then show what is being cut from what and with which cut code. That ought to keep him happy for a while. Now, since we have one complete cut in the output buffer, let's increment the output buffer counter (K) by one for the next piece and set that pesky V back to zero so it won't show a new sheet again until there actually is one. Line 1390 does all of that.

Clear out the deadwood

If the stock array had more pieces and we had found an exact fit somewhere, that array location would have been zeroed by line 1240. In computing, of course, there is a vast difference between zero and nothing. Zero is a discrete value, lying between 1 and -1, while nothing is the absence of anything. In lines 1410 through 1450 we loop through the stock array and if any location has a zero in it we take the very last item in the array (at NS) and stuff it into where the zero was. Then we move the NS marker up one position. Is this really necessary? Not really, you will never find a fit for a required piece with a zero length stock piece, but later when we print out the cutting diagrams and the leftover piece list it would be embarrassing to show zero dimension pieces.

Sort the stock array

As mentioned earlier, if the comparison loops do not find an exact fit or an exact length or width fit, the next piece that comes up that fits will be used. Well, we don't want to use a larger piece when a smaller one would do. It cuts down on the success ratio. So we sort the stockpile after each piece is cut and the leftovers are put back into stock. This sort is almost identical to the ones described earlier, except that this time we sort in ascending order (smallest to largest, "big" this time being determined by length plus width.)

We have been inside the "Big I" loop for a long time, but now everything is taken care of for the first required piece, so at line 1560 we finally do a NEXT I and go back to do the second required piece.

Results

Everything that has happened since we got into the I loop happens for every required piece. Assume, for the moment, that all required pieces have been cut. The I loop is finally done and instead of doing a NEXT I we arrive at line 1570. There is only one subroutine in this program. It resides at line 160 and its purpose is to clear the screen and home the cursor.

Lines 1580 through 1850 print the results of the program on the screen. The results are evaluated in these lines and appropriate messages are generated to give the user information about the run just completed. If the run was unsuccessful, the attempt FL is incremented in line 1730 and another attempt is made. The option to print the cutting diagrams or do another run is given in line 1830. Let's go through the cutting diagrams first and then go back and pick up those loose ends we left dangling earlier.

Print cutting diagrams

The routines from line 1860 through the end of the program at line 2810 print the cutting diagrams for each piece of the project. It begins by asking for a name for the project, which it then prints at the heading. A line concerning the direction of the grain is printed next, kerf (if it was specified) is printed next and another line indicating first and second cut is then printed.

From this point on, starting at line 1940, we loop through the output buffer, one item at a time, and print a scaled down picture of the stock piece along with the cutting information. As we did earlier, to get rid of confusing array dimension names, we change them to L, LC, W, and WC. This happens in line 1970. The information about the stock piece and the required piece is printed immediately prior to each drawing.

Lines 2010 through 2040 adjust the size of the drawing so they will fit on an 80-column printer. The values in lines 2010 and 2020 were chosen to make a roughly proportional drawing that is approximately 4 X 8 inches, giving a scale of approximately 1 inch to each foot. Lines 2030 and 2040 were necessary to keep the printer from adding a line feed when the length of the piece to cut (required piece) was too close to either extreme of the stock piece. On very small pieces there will be extreme distortion but it can't be helped and is due to the resolution of most printers.

Earlier in the program we used variable C to hold the cut code. It was changed to the fifth output array element later (because we had to use C again for the next piece.) In line 1970 we pull it out of the array and make C out of it again. Since C tells us

what cut is required, we use it in an ON..GOTO statement in line 2050 to tell which section of code to go to to get the proper drawing. Since we are still inside the K loop that started in line 1940, each section of code that does the drawing returns to line 2160, which is the NEXT K line.

After the last K item is printed line 2170 sends us to line 2550 where we print the number of full sheets used, the total square inches of the required pieces, a list of the required pieces, the list of pieces left over and finally, the output array is printed along with the cut codes.

No attempt has been made to produce these drawings on standard 8 X 11 paper. They were designed primarily for continuous feed roll paper with no page breaks (set your printer to have the number of lines to print and the number of lines per page to be equal.) The idea is to get something to hang on your shop wall and use as a cutting guide. Now let's get back to some of those leftover items.

The overall view

Having gone through the program once to see how one piece is cut should have given a rough idea of how it works. But there are various paths the program takes to cover all sorts of exceptions. Here is a brief overview, followed by details of code we didn't cover the first time through.

The program organizes the required pieces first into descending order by length plus width and then by width within length. This all happened during attempt number 0. In attempt 0, it goes directly to the comparison routines, gets new sheets as required, cuts them and puts leftover pieces back into stock and goes through the I loop as many times as there are pieces to cut. If the number of new sheets used (NP) is equal or less than the number of pieces it expected to use (LF), it prints the results on the screen and asks if you want a paper printout. End of program. But what happens when it can't make it on the first attempt (sheets used exceeds sheets expected)?

Every time a new sheet is put into stock in line 970 a check is made to see if the new NP is greater than the number of sheets we expected. It does this on every attempt except the very last one, when we will print out what we have in spite of the fact it didn't work. By putting this test at this point we can immediately abort the current attempt if it isn't going to work and go on to the next attempt. It saves a lot of time. If the test at 980 is true, we go to 1710 where we reinitialize the three arrays, print the new attempt number on the screen, increment the attempt counter and in line 1740, if the attempt number is 0 or 1, go back to the sort routine in line 510. If the attempt number is 2 or more, we go back to line 730 instead and bypass the sorts.

Assume we are now on attempt number 1 (actually the second attempt, since we start counting at 0.) The program will take us back to line 510 where we will sort again, but this time the sort will be by length only. We will again sort width within length and go directly to the comparison routines. Nothing much different here yet, except that the sort was a little different. But if we get through lines 1240 through 1260 (which, by the way, always have first priority), line 1280 will force a 5 cut on the first piece of a new sheet.

Now let's say that it still didn't work out and aborted again in line 980. This time when we increment the attempt counter it will be at 2 and line 1740 will send us back to line 730 instead of to the sort routines. Our attempt counter number is even this time and so the statement in line 740 will allow us to proceed to the lines immediately following it.

The recirculate routine

On every even attempt number after the first two attempts the code at lines 760 through 810 will move the last item of the required piece array into the first position and move all the remaining pieces down one. We do this by using the zero element of the array (previously unused.) Before entering the loop, we simply make position zero equal to whatever is in the last position (NR). Then, going backwards through the loop, we make the second-to-last item the last, the third-to-last item the second-to-last, etc., until we reach the top of the loop, where the data in the zero position becomes the data in position one. This effectively circulates the data in the array by one position. Now we go through the I loop and the comparison routines again for all the pieces in the required list. If we are still not successful, the 5 cut logic will operate on the next try and after that the recirculate logic will operate again and two more attempts will be made with the reordered list.

The impossible fit routine

On the next to last attempt, the code from 840 through 900 comes into play. What it says in effect is: "We have given this dude every opportunity to fall apart and it still won't yield, so let's see if we are dealing with impossible cuts or fits, and if so, relax our success requirements by adding one to our expected sheet counter (LF)." Here we use a FOR...NEXT loop which compares the first item in the list to every other item, then the second to all the remaining items, then the third to the rest, etc. If either of the two logic statements in lines 870 and 880 is true, variable FT is set to 1 and the expected sheet counter (LF) is incremented by 1. This is only true when the expected sheets is 1 to begin with. Two pieces which are impossible to fit or cut from one sheet can usually be cut from two sheets by taking one piece from each of the two

parent sheets. Surprisingly enough, this section of code comes into play more often than we had imagined.

The move 48 inch routine

The section of code from 990 to 1060 did a lot to help the success rate of the program on multiple sheet problems. What is does is force full width (48 inch pieces) to be cut immediately when a new sheet is put into stock. If there is more than one such piece it saves the next one until the next full sheet is put into stock. This way, it doesn't take both out of one sheet and possibly ruin that sheet for anything else we may need. Line 990 sets it up. V has to be 1, meaning that we just pulled a new full sheet into stock and the attempt counter must be at 0, 1, 2 or 3. The logic being that if it can't get it in four passes it probably was not meaningful and we will let the chips fall where they may. Notice the loop from 1010 to 1060. We are using Q to count the loop and it starts counting at the *current* I and goes to the end of the list. Notice also that as soon as it finds a 48 inch piece, it exits the loop, so it only moves the first 48 inch piece into the next cutting position.

The 5 cut logic

The cut that receives the highest priority is the perfect fit where no cut is required at all. Next come the single cut requirements where you have no choice in how to cut in any case. The tough choice is when two cuts are required. Which way do you cut first? These are the 4 and 5 cut problems.

Left to its own devices, the program will opt for the 4 cut (see line 1340.) The 5 cut is not necessary that often but when needed there is no other solution without it. Lines 1290 through 1330 examine five conditions that can force a 5 cut. They represent almost 10% improvement in the success rate. We will go through the first one only to give an idea of how they work. Line 1290 says that if the length of the required piece we are currently looking at plus the length of the *next* required piece is equal to the length of the current stock piece, and the widths of the two required pieces is the same, then force a 5 cut. The rest of the 5 cut logic statements look at the next *two* required pieces to determine the cut on the *current* piece. Obviously, if this logic could be extended, we wouldn't need the remainder of the program.

Hints on using the program

The program may be used not only to determine the cuts on an existing design, but to optimize a new design. It allows you to play the "what if" game, which gave rise to the nickname, "BalsaCalc."

The program cannot handle pieces shaped other than square or rectangular. Consequently, such pieces should be reduced to their smallest rectangle or square prior to using the program.

In some cases, where the grain of the wood is especially well matched and you want to get a set of drawer fronts or doors with a matching grain pattern, treat the two pieces as one in the program. It will prevent the program from taking the two pieces from different areas of the stock. Don't forget to allow for the saw kerf if you do this.

Saw kerf may be changed to suit your needs in lines 290 and 300. These are the only two places in the program where it is defined. Likewise the length and width of stock sheets is defined in line 320. Should you, for some reason, get a bunch of sheets at a discount because the edges are rolled over, you can temporarily change line 320 to take the smaller dimensions into account.

Printers usually buy paper in large parent sheets. (Paper, by the way, also has grain.) We used the program to cut parent paper sheets into smaller ones. It worked, but if you intend to do that on a regular basis, consider changing the printout scaling since most paper sheets are considerably smaller than a sheet of plywood.

Enter your dimensions carefully. There is no provision to edit them. However, if you should put in a wrong length just give it a ridiculous width (like 0) and the program will ask you to start that item all over again.

How well does it work?

After much testing, we can say with reasonable assurance that it will cut better than 90% of all one-sheet problems. Multi-sheet problems seem to average around 85%. The average number of attempts for one-sheet problems we tried is 3. The time for one-sheet problems was about 3 minutes or less. We tried real-world problems that fell apart on the 2nd or 3rd attempt, but on the other side of that coin we also had a four-sheet problem that took over two hours and came up with a five-sheet answer. If all this sounds too slow, consider what would happen if we tried 10 factorial pieces and it couldn't find a solution. It would take something like sixty *days!* We think the improvement, in spite of the less than 100% accuracy, is significant.

This is the type of program you can spend years tinkering with. There is room for improvement. If you should happen to get turned on and make any significant improvements please let us know.

A question still unanswered

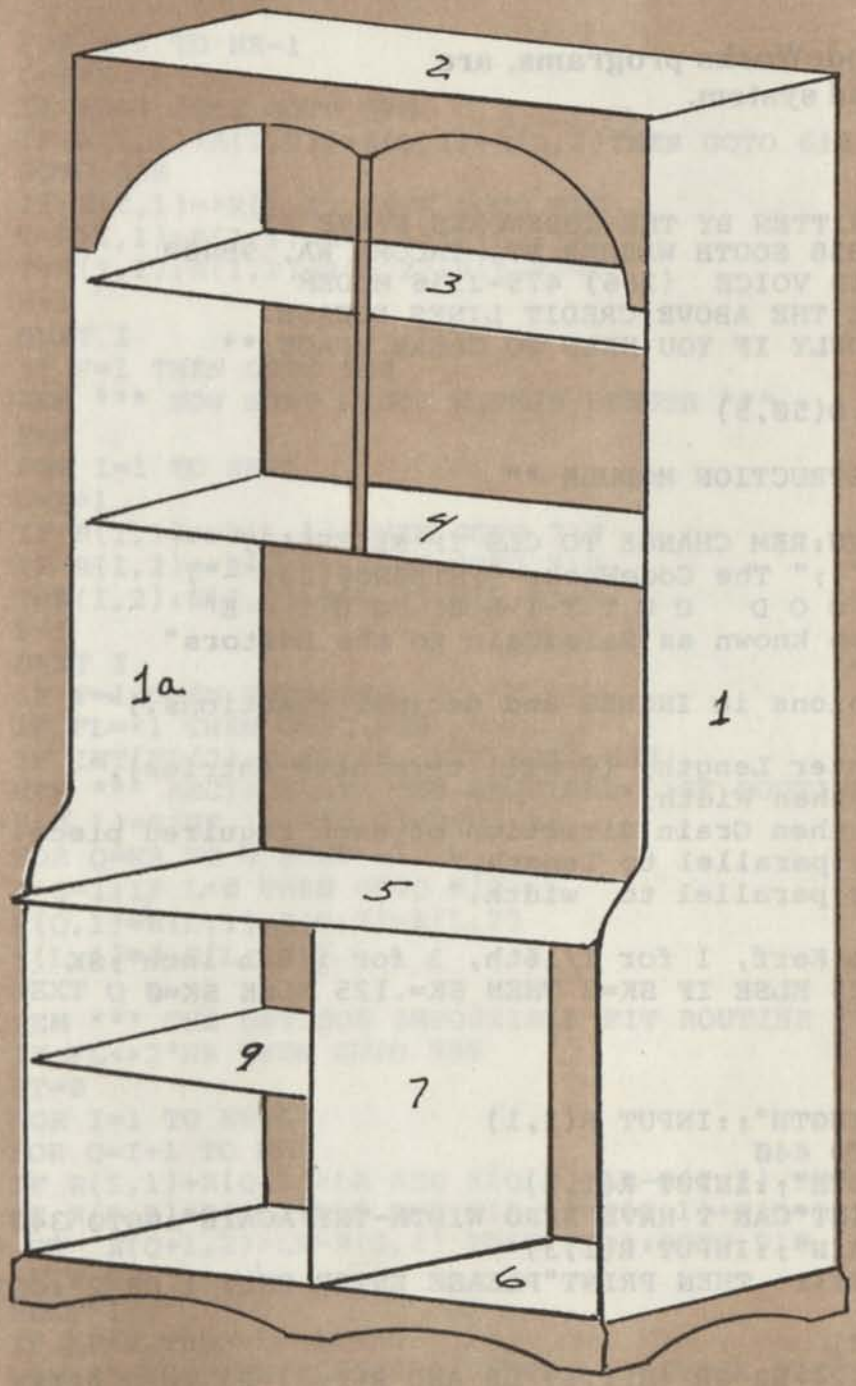
In spite of the work done on this program, the big question is still begging for an answer. Given any number of pieces with dimensions of x and y, and assuming that cuts must be made all the way across the stock piece, how can you determine whether or not they can be cut from a larger sheet with dimensions of xx and yy? Martin Gardner, where are you when we need you? ■

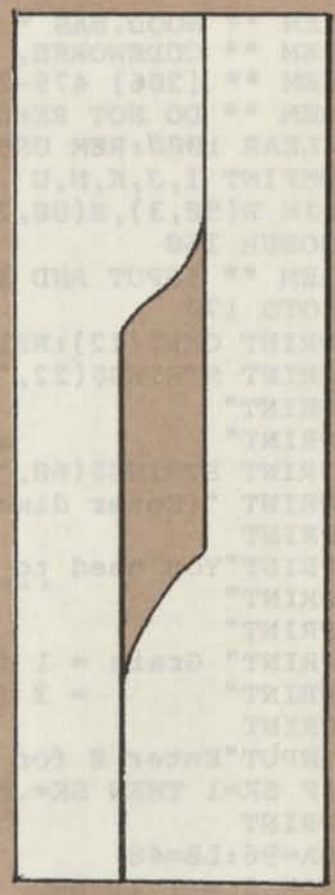
This is a photographically reduced version of the actual output for the sample run.

```
PROJECT ID IS: SAMPLE RUN

GRAIN ALWAYS RUNS PARALLEL TO THIS DIRECTION ----->>>

KERF OF  .125 INCHES IS REMOVED FROM CUTOFF PIECES.
  1 1 1 INDICATES 1ST CUT, 2 2 2 INDICATES 2ND CUT


PIECE #  1
STOCK PIECE IS  96 INCHES LONG AND  48 INCHES WIDE
PIECE TO CUT IS  21 INCHES LONG AND  11 INCHES WIDE / CUT CODE = 5

PIECE #  2
STOCK PIECE IS  74.875 INCHES LONG AND  11 INCHES WIDE
PIECE TO CUT IS  14.625 INCHES LONG AND  10.25 INCHES WIDE / CUT CODE = 4

PIECE #  3
STOCK PIECE IS  96 INCHES LONG AND  36.875 INCHES WIDE
PIECE TO CUT IS  70 INCHES LONG AND  19.875 INCHES WIDE / CUT CODE = 4

PIECE #  4
STOCK PIECE IS  70 INCHES LONG AND  16.875 INCHES WIDE
PIECE TO CUT IS  31.5 INCHES LONG AND  12 INCHES WIDE / CUT CODE = 4

PIECE #  5
STOCK PIECE IS  60.125 INCHES LONG AND  11 INCHES WIDE
PIECE TO CUT IS  30 INCHES LONG AND  11 INCHES WIDE / CUT CODE = 3

PIECE #  6
STOCK PIECE IS  30 INCHES LONG AND  11 INCHES WIDE
PIECE TO CUT IS  30 INCHES LONG AND  8.75 INCHES WIDE / CUT CODE = 2

PIECE #  7
STOCK PIECE IS  36.375 INCHES LONG AND  16.875 INCHES WIDE
PIECE TO CUT IS  30 INCHES LONG AND  7.5 INCHES WIDE / CUT CODE = 4

PIECE #  8
STOCK PIECE IS  30 INCHES LONG AND  9.25 INCHES WIDE
PIECE TO CUT IS  30 INCHES LONG AND  7.5 INCHES WIDE / CUT CODE = 2

PIECE #  9
STOCK PIECE IS  25.875 INCHES LONG AND  36.875 INCHES WIDE
PIECE TO CUT IS  21 INCHES LONG AND  29.375 INCHES WIDE / CUT CODE = 4

NUMBER OF FULL SHEETS USED =  1
TOTAL SQ INCHES OF REQUIRED PIECES = 3889.53

LIST OF REQUIRED PIECES
LENGTH          WIDTH
-----------------------
21        X     11
14.625    X     10.25
70        X     19.875
31.5      X     12
30        X     11
30        X     8.75
30        X     7.5
30        X     7.5
21        X     29.375

LIST OF LEFTOVER PIECES
LENGTH          WIDTH
-----------------------
14.625    X     .625
8.25      X     16.875
21        X     7.375
30        X     1.625
30        X     2.125
31.5      X     4.75
4.75      X     36.875

THE CUTTING ORDER AND CUT CODES ARE:

21        X 11          OUT OF 96        X  48          CUT CODE 5
14.625    X 10.25       OUT OF 74.875    X  11          CUT CODE 4
70        X 19.875      OUT OF 96        X  36.875      CUT CODE 4
31.5      X 12          OUT OF 70        X  16.875      CUT CODE 4
30        X 11          OUT OF 60.125    X  11          CUT CODE 3
30        X 8.75        OUT OF 30        X  11          CUT CODE 2
30        X 7.5         OUT OF 36.375    X  16.875      CUT CODE 4
30        X 7.5         OUT OF 30        X  9.25        CUT CODE 2
21        X 29.375      OUT OF 25.875    X  36.875      CUT CODE 4
```

Sample Run

Use Kerf option 2 and the required pieces shown in the figure on this page (directly to the right of here), and you should get the same solution shown on this page. The hand drawn numbers on the sample output were put there to show which pieces correspond to the drawing on the facing page.





Detail of sides 1 and 1a

The two doors, 8 and 8a, for the lower front are not shown here. They come out of one piece of wood so that the grain matches. This cabinet actually was built by us out of one sheet of birch plywood (the back and the trim on top come from a different sheet of quarter inch ply.) If you typed the program in correctly and use the required pieces on the opposite page, the solution to this problem should come on the sixth attempt out of 19 possible.

This, as well as other CodeWorks programs, are available on the download system.

```
100 REM ** WOOD.BAS * WRITTEN BY THE CODEWORKS STAFF **
101 REM ** CODEWORKS, 3838 SOUTH WARNER ST. TACOMA WA, 98409
102 REM ** (206) 475-2219 VOICE  (206) 475-2356 MODEM
103 REM ** DO NOT REMOVE THE ABOVE CREDIT LINES PLEASE.
105 CLEAR 1000:REM USE ONLY IF YOU NEED TO CLEAR SPACE **
110 DEFINT I,J,K,N,U
120 DIM R(50,3),S(80,2),O(50,5)
130 GOSUB 160
140 REM ** INPUT AND INSTRUCTION MODULE **
150 GOTO 170
160 PRINT CHR$(12):RETURN:REM CHANGE TO CLS IF NECESSARY **
170 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
180 PRINT"            W O O D   C U T T I N G   G U I D E"
190 PRINT"           also known as BalsaCalc to the Editors"
200 PRINT STRING$(60,"-")
210 PRINT "(Enter dimensions in INCHES and decimal fractions.)"
220 PRINT
230 PRINT"You need to enter Length, (0 will terminate entries),"
240 PRINT"              then Width,"
250 PRINT"              then Grain direction of each required piece."
260 PRINT" Grain = 1 for parallel to length,"
270 PRINT"       = 2 for parallel to  width."
280 PRINT
290 INPUT"Enter 0 for no Kerf, 1 for 1/16th, 2 for 1/8th inch";SK
300 IF SK=1 THEN SK=.0625 ELSE IF SK=2 THEN SK=.125 ELSE SK=0
310 PRINT
320 LA=96:LB=48
330 FOR I = 1 TO 49
340 PRINT"PIECE #";I;"LENGTH";:INPUT R(I,1)
350 IF R(I,1)=0 THEN GOTO 440
360 PRINT"           WIDTH";:INPUT R(I,2)
370 IF R(I,2)=0 THEN PRINT"CAN'T HAVE ZERO WIDTH-TRY AGAIN":GOTO 340
380 PRINT"           GRAIN";:INPUT R(I,3)
390 IF R(I,3)>2 OR R(I,3)<1  THEN PRINT"PLEASE ENTER ONLY 1 OR 2":GOTO 3
80
400 PRINT STRING$(33,"-")
410 IF R(I,1)>LA OR R(I,2)>LB OR (R(I,1)>LB AND R(I,3)=2) THEN PRINT "CA
N'T BE DONE - TRY AGAIN":GOTO 340
420 IF R(I,3)=2 THEN T=R(I,1):R(I,1)=R(I,2):R(I,2)=T
430 NEXT I
440 NR=I-1:FL=0:SQ=0:LF=0
450 REM ** FIND THE TOTAL SQ AREA OF REQ. PIECES **
460 FOR I=1 TO NR
470 SQ=SQ+R(I,1)*R(I,2)
480 NEXT I
490 LF=INT((SQ-1)/(LA*LB))+1:REM ** ESTABLISH EXPECTED SHEETS **
500 REM ** SORT REQ PIECES INTO DESCENDING ORDER ***
510 F=0
```

```
520 FOR I=1 TO NR-1
530 L=I+1
540 IF FL=1 THEN GOTO 570
550 IF R(I,1)+R(I,2)=>R(L,1)+R(L,2)THEN GOTO 610
560 GOTO 580
570 IF R(I,1)=>R(L,1) THEN GOTO 610
580 T=R(I,1):R(I,1)=R(L,1):R(L,1)=T
590 T=R(I,2):R(I,2)=R(L,2):R(L,2)=T
600 F=1
610 NEXT I
620 IF F=1 THEN GOTO 510
630 REM *** NOW SORT WIDTH WITHIN LENGTH ***
640 F=0
650 FOR I=1 TO NR-1
660 L=I+1
670 IF R(I,1)<>R(L,1) THEN GOTO 710
680 IF R(I,2)=>R(L,2) THEN GOTO 710
690 T=R(I,2):R(I,2)=R(L,2):R(L,2)=T
700 F=1
710 NEXT I
720 IF F=1 THEN GOTO 640
730 IF FL=<1 THEN GOTO 930
740 IF INT(FL/2)-FL/2<>0 THEN GOTO 830
750 REM *** RECIRCULATE THE REQUIRED LIST ROUTINE ***
760 R(0,1)=R(NR,1):R(0,2)=R(NR,2)
770 FOR Q=NR TO 0 STEP -1
780 L=Q-1:IF L<0 THEN GOTO 810
790 R(Q,1)=R(L,1):R(Q,2)=R(L,2)
800 R(L,1)=0:R(L,2)=0
810 NEXT Q
820 REM *** THE OBVIOUS IMPOSSIBLE FIT ROUTINE ****
830 IF FL<>2*NR THEN GOTO 930
840 FT=0
850 FOR I=1 TO NR-1
860 FOR Q=I+1 TO NR
870 IF R(I,1)+R(Q,1)>LA AND R(Q,2)>LB-R(I,2) THEN FT=1:GOTO 910
880 IF R(I,2)+R(Q,2)>LB AND R(I,1)+R(Q,1)+R(Q+1,1)>LA AND R(Q+1,2)>LB-R(
I,2) OR  R(Q+1,2)>LB-R(Q,2) THEN FT=1:GOTO 910
890 NEXT Q
900 NEXT I
910 IF LF<2 THEN LF=LF+FT
920 REM ** COMPARE REQUIRED PIECES TO STOCK PIECES ***
930 K=1
940 FOR I=1 TO NR
950 J=1
960 REM ** IF THERE IS NO STOCK - PULL IN A NEW SHEET ***
970 IF J>NS THEN S(J,1)=LA:S(J,2)=LB:NP=NP+1:NS=NS+1:V=1:GOTO 950
980 IF NP>LF AND FL<>2*NR THEN GOTO 1710
990 IF V<>1 OR FL=>4 THEN GOTO 1080
1000 REM *** MOVE NEXT 48" PIECE SO IT CUTS FROM A FULL SHEET ***
1010 FOR Q=I TO NR
1020 IF R(Q,2)<>LB THEN GOTO 1060
1030 T=R(I,1):R(I,1)=R(Q,1):R(Q,1)=T
1040 T=R(I,2):R(I,2)=R(Q,2):R(Q,2)=T
```

```
1050 GOTO 1080
1060 NEXT Q
1070 REM ** LOOK FOR EXACT FIT IN THE STOCKPILE ***
1080 FOR U=1 TO NS:IF R(I,1)=S(U,1) AND R(I,2)=S(U,2) THEN J=U:GOTO 1210
1090 NEXT U
1100 REM *** LOOK FOR EQUAL LENGTHS ****
1110 FOR U=1 TO NS:IF R(I,1)=S(U,1) AND R(I,2)=< S(U,2)THEN J=U:GOTO 121
0
1120 NEXT U
1130 REM *** LOOK FOR EQUAL WIDTHS ****
1140 FOR U=1 TO NS:IF R(I,2)=S(U,2) AND R(I,1)=< S(U,1) THEN J=U:GOTO 12
10
1150 NEXT U
1160 REM *** TAKE ANYTHING THAT FITS! ***
1170 FOR U=1 TO NS:IF R(I,1)=<S(U,1) AND R(I,2)=< S(U,2) THEN J=U:GOTO 1
210
1180 NEXT U
1190 J=J+1:GOTO 970
1200 REM ** STUFF THE CUT BUFFER WITH ALL THE DIMENSIONS ***
1210 O(K,1)=R(I,1):O(K,2)=R(I,2):O(K,3)=S(J,1):O(K,4)=S(J,2)
1220 L=S(J,1):W=S(J,2):LC=R(I,1):WC=R(I,2)
1230 REM *** CHECK FOR SINGLE CUT SITUATIONS *****
1240 IF L=LC AND W=WC THEN S(J,1)=0:S(J,2)=0:C=1:GOTO 1360
1250 IF L=LC THEN S(J,2)=W-WC-SK:C=2:GOTO 1360
1260 IF W=WC THEN S(J,1)=L-LC-SK:C=3:GOTO 1360
1270 REM ** DETERMINE WHETHER 4 OR 5 CUT AND SIZE OF LEFTOVERS ***
1280 IF INT(FL/2)-FL/2<>0 AND V=1 AND I=1 THEN GOTO 1350
1290 IF R(I,1)+R(I+1,1)=S(J,1) AND R(I,2)=R(I+1,2) THEN GOTO 1350
1300 IF R(I+1,1)+R(I+2,1)=S(J,1) AND R(I+1,2)=R(I+2,2) THEN GOTO 1350
1310 IF R(I+1,1)=R(I+2,1) AND R(I,1)=R(I+1,1) AND R(I+1,2)=R(I+2,2) THEN
 GOTO 1350
1320 IF R(I+1,1)+R(I+2,1)=S(J,1) AND R(I+1,2)+R(I+2,2)=<S(J,2) THEN GOTO
 1350
1330 IF R(I+1,1)+R(I+2,1)>R(I,1) AND R(I+1,2)=R(I+2,2) THEN GOTO 1350
1340 S(J,1)=L-LC-SK:S(J,2)=W:J=NS+1:S(J,1)=LC:S(J,2)=W-WC-SK:NS=NS+1:C=4
:GOTO 1360
1350 S(J,1)=L:S(J,2)=W-WC-SK:J=NS+1:S(J,1)=L-LC-SK:S(J,2)=WC:NS=NS+1:C=5
1360 O(K,5)=C:NO=NO+1:REM ** ADD CUT CODE TO CUT BUFFER ***
1370 IF V=1 THEN PRINT TAB(10);"--- New Sheet ---"
1380 PRINT"CUTTING "O(K,1);O(K,2);"from";O(K,3);O(K,4);"cut code";O(K,5)
1390 K=K+1:V=0
1400 REM ** ELIMINATE DEADWOOD FROM STOCKPILE **
1410 F=0
1420 FOR J=1 TO NS-1
1430 IF S(J,1)=0 THEN S(J,1)=S(NS,1):S(J,2)=S(NS,2):NS=NS-1:F=1
1440 NEXT J
1450 IF F=1 THEN GOTO 1410
1460 REM ** SORT THE STOCKPILE INTO ASCENDING ORDER **
1470 F=0
1480 FOR J=1 TO NS-1
1490 L=J+1
1500 IF S(J,1)+S(J,2)=<S(L,1)+S(L,2) THEN GOTO 1540
1510 T=S(J,1):S(J,1)=S(L,1):S(L,1)=T
1520 T=S(J,2):S(J,2)=S(L,2):S(L,2)=T
1530 F=1
1540 NEXT J
```

```
1550 IF F=1 THEN GOTO 1470
1560 NEXT I
1570 GOSUB 160
1580 PRINT TAB(15);"----- R E S U L T S -----"
1590 PRINT"AREA REQUIRED =";SQ;":AREA LEFT OVER = ";NP*(LA*LB)-SQ;"SQ.IN
."
1600 PRINT"MINIMUM SHEETS FOR THIS PROJECT BY SQ. AREA = ";INT((SQ-1)/(L
A*LB))+1
1610 PRINT"YOU NEED ";NP;" FULL SHEET(S) WITH A TOTAL AREA =";(LA*LB)*NP
1620 PRINT"Kerf =";SK;"Inches"
1630 PRINT"THE CUTTING ORDER FOLLOWS (Grain runs parallel to length)"
1640 PRINT
1650 PRINT"Length";TAB(10);"Width    OUT OF";TAB(29);"Length";TAB(40);"W
idth";TAB(50);"CUT CODE"
1660 PRINT
1670 FOR K=1 TO NO
1680 IF O(K,3)=LA AND O(K,4)=LB THEN D$="New sheet->" ELSE D$=""
1690 PRINT O(K,1);TAB(10);O(K,2);TAB(19);D$;TAB(30);O(K,3);TAB(40);O(K,4
);TAB(50);O(K,5)
1700 NEXT K
1710 IF NP>LF AND FL<>2*NR THEN NP=0:NS=0:NO=0:GOTO 1720 ELSE GOTO 1750
1720 PRINT:PRINT"Attempt--> ";FL+2;" of ";2*NR+1
1730 FL=FL+1
1740 IF FL=>2 THEN GOTO 730 ELSE GOTO 510
1750 PRINT
1760 IF SQ=NP*(LA*LB) THEN PRINT "ATTEMPTS =";FL+1;":This is an EXCELLEN
T solution!":GOTO 1820
1770 IF NP<>LF THEN PRINT"ATTEMPTS =";FL+1;":A POOR solution or IMPOSSIB
LE fit  or cut"
1780 IF NP<>LF THEN PRINT"I can't tell which, it's up to you."
1790 IF NR>4 AND LF<>NP THEN PRINT"You might try to combine similar piec
es and do better.":GOTO 1820
1800 IF LF=NP THEN PRINT"ATTEMPTS =";FL+1;":This is a SATISFACTORY solut
ion."
1810 IF SQ=<(LA*LB)*NP AND FT=1 THEN PRINT"Extra sheet is due to an impo
ssible fit or cut."
1820 PRINT
1830 INPUT"Do you wish to print the cutting diagrams (Y/N)";A$
1840 IF A$="Y" OR A$="y" THEN GOTO 1860 ELSE RUN 100
1850 END
1860 REM *** PRINT CUT LIST ROUTINE ***
1870 INPUT"ENTER THE NAME OF THIS PROJECT";B$
1880 LPRINT "PROJECT ID IS: ";B$
1890 LPRINT" "
1900 LPRINT"GRAIN ALWAYS RUNS PARALLEL TO THIS DIRECTION ----->>>"
1910 LPRINT" "
1920 IF SK<>0 THEN LPRINT"KERF OF ";SK;"INCHES IS REMOVED FROM CUTOFF PI
ECES."
1930 LPRINT"  1 1 1 INDICATES 1ST CUT, 2 2 2 INDICATES 2ND CUT"
1940 FOR K = 1 TO NO
1950 LPRINT" ":LPRINT" "
1960 LPRINT "PIECE # ";K
1970 L=O(K,3):W=O(K,4):LC=O(K,1):WC=O(K,2):C=O(K,5)
1980 LPRINT"STOCK PIECE IS ";L;"INCHES LONG AND ";W;"INCHES WIDE"
1990 LPRINT"PIECE TO CUT IS ";LC;"INCHES LONG AND ";WC;"INCHES WIDE";" /
 CUT CODE =";C
```

```
2000 LPRINT" "
2010 L=L*.82:LC=LC*.82
2020 W=W*.44:WC=WC*.44
2030 IF LC=<6 THEN LC=6
2040 IF LC > (.94*L) THEN LC=(.94*L)
2050 ON C GOTO 2300,2370,2480,2190,2070
2060 REM *** CODE 5 CUT -2 CUTS REQUIRED 1ST HORIZ 2ND VERT ***
2070 LPRINT STRING$(L,"-")
2080 FOR M=1 TO WC
2090 LPRINT"!";STRING$(LC-2,">");TAB(LC);"2";TAB(L);"!"
2100 NEXT M
2110 LPRINT STRING$(L,"1")
2120 FOR U=WC TO W
2130 LPRINT"!";TAB(L);"!"
2140 NEXT U
2150 LPRINT STRING$(L,"-")
2160 NEXT K
2170 GOTO 2550
2180 REM *** CODE 4 CUT -2 CUTS REQUIRED 1ST VERT 2ND HORIZ ***
2190 LPRINT STRING$(L,"-")
2200 FOR M=1 TO WC
2210 LPRINT "!";STRING$(LC-2,">");TAB(LC);"1";TAB(L);"!"
2220 NEXT M
2230 LPRINT STRING$(LC,"2");TAB(L);"!"
2240 FOR U = WC TO W
2250 LPRINT "!";TAB(LC);"1";TAB(L);"!"
2260 NEXT U
2270 LPRINT STRING$(L,"-")
2280 GOTO 2160
2290 REM *** CODE 1 CUT -  NO CUTS REQUIRED ***
2300 LPRINT" "
2310 LPRINT"************************************"
2320 LPRINT"*                                  *"
2330 LPRINT"*  EXACT FIT - NO CUTS REQUIRED    *"
2340 LPRINT"*                                  *"
2350 LPRINT"************************************"
2360 GOTO 2160
2370 REM *** CODE 2 CUT  - ONE HORIZ CUT REQUIRED ***
2380 LPRINT STRING$(L,"-")
2390 FOR M=1 TO WC
2400 LPRINT "!";STRING$(L-2,">");TAB(L);"!"
2410 NEXT M
2420 LPRINT STRING$(L,"1")
2430 FOR U=WC TO W
2440 LPRINT "!";TAB(L);"!"
2450 NEXT U
2460 LPRINT STRING$(L,"-")
2470 GOTO 2160
2480 REM *** CODE 3 CUT - ONE VERT CUT REQUIRED ***
2490 LPRINT STRING$(L,"-")
2500 FOR M = 1 TO W
2510 LPRINT"!";STRING$(LC-2,">");TAB(LC);"1";TAB(L);"!"
2520 NEXT M
2530 LPRINT STRING$(L,"-")
2540 GOTO 2160
```

```
2550 REM **** PRINT REQUIRED PIECE LIST AND LEFTOVER STOCK ****
2560 LPRINT" "
2570 LPRINT"NUMBER OF FULL SHEETS USED = ";NP
2580 LPRINT"TOTAL SQ INCHES OF REQUIRED PIECES =";SQ
2590 LPRINT" "
2600 LPRINT"LIST OF REQUIRED PIECES"
2610 LPRINT "LENGTH";TAB(15);"WIDTH"
2620 LPRINT STRING$(20,"-")
2630 FOR I=1 TO NR
2640 LPRINT R(I,1);TAB(10);"X";TAB(15);R(I,2)
2650 NEXT I
2660 LPRINT" "
2670 LPRINT"LIST OF LEFTOVER PIECES"
2680 LPRINT"LENGTH";TAB(15);"WIDTH"
2690 LPRINT STRING$(20,"-")
2700 FOR J= 1 TO NS
2710 LPRINT S(J,1);TAB(10);"X";TAB(15);S(J,2)
2720 NEXT J
2730 REM PRINT THE CUT BUFFER ******
2740 LPRINT" "
2750 LPRINT" THE CUTTING ORDER AND CUT CODES ARE:"
2760 LPRINT" "
2770 FOR K=1 TO NO
2780 LPRINT O(K,1);TAB(10);"X";TAB(12);O(K,2);TAB(24)"OUT OF";TAB(30);O(
K,3);TAB(40);"X";TAB(42);O(K,4);TAB(55);"CUT CODE";O(K,5)
2790 NEXT K
2800 PRINT"DONE"
2810 END
```

Programming Notes

This note may be something we said before, but bears repeating. A St. Louis reader called to say he had problems with the files in our Card.Bas program from Issue 2. It turned out that he was using a Tandy Model III and used the file as we presented it in the program listing. When he went to look at the file from the DOS Ready prompt, it came back and said "File Access Denied". Most Tandy machines (except the newer MS-DOS types) use a slash for the file extension. A little-used feature of the Tandy DOS is that you may specify a file password like this: filename.pwd/BAS. The "pwd" then becomes the password, and the extension is still BAS. If you don't specify the password, you cannot read your file back. Since MS-DOS, CP/M and others use the ".BAS" convention it looks like a password to the earlier Tandy machines.

We would like to ignore extensions in the magazine but they are too handy to ignore.

"THEY'RE COMPUTER BYTES."

The Shell Sort

2nd in a series on Sorts

Staff article

In the last issue we discussed the Bubble Sort. One of the problems with that sort is that the time it takes goes up almost exponentially with the number of items it has to sort. We can make a significant improvement in sort time with the Shell sort. The Shell sort we are presenting here is not really a true Shell sort. It is a modified version of the real thing. The real thing uses a stack and other devices which make it work better but complicates the code considerably.

The basic idea behind this sort is to divide and conquer. Instead of "bubbling" an item all the way down the line, this one simply picks the item up and moves it directly to where it will be in a better position. It picks an approximate mid-point in the file (see line 220 of the listing) and then if the number in the first position of the second half is smaller than the number in the first position of the first half, it swaps them. (See figure 1, the first swap switched the 1 in position 6 with the 92 in the first position.) A little study of figure 1 will show graphically what happens during this sort. Notice the 98 in the third position. It stays there for two passes then jumps over to position eight and stays there for four passes. In the next pass, it is moved directly to position 10, where it actually belongs.

In the program, the actual sort takes place between lines 210 and 360. It can be done in fewer lines by making multiple line statements, but we left it open so it would be easier to follow. Line 300 is not essential to the sort; it is included for demonstration purposes only.

At the beginning of the program we generate a batch of random numbers to sort. You may need to un-remark the Randomize in line 110 if your machine needs it. Just before we go into the sort we get the system time in TIME$. After the sort is done we get it again so that the time to sort can be observed. If you don't have TIME$ it will be treated just like any other string variable and will return a zero. If your BASIC supports the SWAP command you can use it in line 290.

The speed of this sort, when compared to the Bubble sort, is interesting. On fifty or less items you can hardly see the difference, but when the number of items gets into the 200 and over range this sort really shows its stuff. In Issue 2 we presented CARD.BAS. When we originally wrote it we used a Bubble sort. It took well over two hours to sort 200 records. We reduced that time to about twenty minutes with the Shell sort (and further reduced it to about four minutes by sorting pointers instead of strings.) You may want to revisit the sort article in Issue 2 and compare some times between the Bubble sort and this one. If you do, don't forget to try various numbers of items using both sorts. The difference should amaze you.

```
HOW MANY NUMBERS TO SORT? 10
 92  10  98  75  60   1  55   8  36  75
  1  10  98  75  60  92  55   8  36  75
  1  10   8  75  60  92  55  98  36  75  SWAP 1 - 92
  1  10   8  36  60  92  55  98  75  75  SWAP 8 - 98
  1  10   8  36  60  92  55  98  75  75  SWAP 36 - 75
  1  10   8  36  55  92  60  98  75  75  SWAP 10 - 10
  1  10   8  36  55  92  60  75  75  98  SWAP 55 - 60
  1  10   8  36  55  75  60  92  75  98  SWAP 75 - 98
  1  10   8  36  55  75  60  92  75  98  SWAP 75 - 92
  1   8  10  36  55  75  60  92  75  98  SWAP 1 - 1
  1   8  10  36  55  60  75  92  75  98  SWAP 8 - 10
  1   8  10  36  55  60  75  75  92  98  SWAP 60 - 75
  1   8  10  36  55  60  75  75  92  98  SWAP 75 - 92
SWAPS =  11
ENDING TIME = 00.45.03
START  TIME = 00.45.00
```

Figure 1

```
100 REM ** MODIFIED SHELL SORT DEMO **
110 'RANDOMIZE :'IF YOU NEED TO
120 INPUT"HOW MANY NUMBERS TO SORT";N
130 DIM A(1000)
140 FOR I=1 TO N
150   A(I)=RND(100):'USE YOUR BRAND OF RND HERE
160   PRINT A(I);
170 NEXT I
180 PRINT
190 S$=TIME$:'IF YOU HAVE NO TIME$ DON'T WORRY
200 PRINT   :'IT WILL JUST PRINT 0 FOR TIME
210 M=N
220 M=INT(M/2)
230 IF M=<0 THEN GOTO 370
240 J=0
250 K=N-M
260 I=J
270 L=I+M
280 IF A(I)=<A(L) THEN GOTO 340
290 A(T)=A(I):A(I)=A(L):A(L)=A(T):SW=SW+1
300 FOR Q=1 TO N:PRINT A(Q);:NEXT Q:PRINT" SWAP";A(I);"-";A(L)
310 I=I-M
320 IF I<1 THEN GOTO 340
330 GOTO 270
340 J=J+1
350 IF J>K THEN GOTO 220
360 GOTO 260
370 SS$=TIME$
380 FOR X=1 TO N
390   PRINT A(X);
400 NEXT X
410 PRINT"SORT IS COMPLETE"
420 PRINT:PRINT
430 PRINT"SWAPS = ";SW
440 PRINT
450 PRINT"ENDING TIME = ";SS$
460 PRINT"START  TIME = ";S$
```

Parting Shots

The Staff at CodeWorks wishes each of you a happy and prosperous NEW YEAR

"I HEAR HE'S ONE OF THE FASTEST COMPUTER PROGRAMMERS AROUND!"

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Address correction requested

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

CodeWorks

Issue 4 Mar/Apr 1986

CONTENTS

CodeWorks

Issue 4 **Mar/Apr 1986**

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

Authors: We constantly seek material from contributors. Send your material (double spaced, upper/lower case, please) 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. Cartoons and photographs are welcome. 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 will receive all issues for the subscription year. Not available outside United States zip codes. VISA and Master Card orders are accepted by mail or telephone (206) 475-2219.

CodeWorks is published bimonthly in Jan, Mar, May, Jul, Sep and Nov. It is printed in **The United States of America. Bulk rate postage 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

After months of diligent work we finally got our download system up and running. We were putting the finishing touches on it at Christmas, and thought that we were over the first hurdle. Of all things, the week after New Year's we had a lightning storm. This in a part of the country where we rarely even get them in the summer. The storm left our computer intact, but our hard disk didn't come through it well. The morning after the storm the hard drive sounded like a cement mixer.

The entire bubble on the drive had to be replaced. That, of course, meant that the entire system needed to be re-initialized as well. Somewhere around the third week of January, we finally got back to where we were before the storm. There were a few more quirks for a week or so, and by the second week of February we had most of them taken care of. Thanks to all of you for reporting problems and suggestions to us.

We had a good time putting together the Math.Bas program in this issue. The idea sort of evolved from something else we were doing. One thing led to another and the program was born. Even though the math may be old hat, watching the action is interesting.

One of our main programs in this issue is the payroll program. All the while I was working on it I kept thinking that the same structure could be used for other, similar, things. Basically it is just another form of a database. Well, one thing led to another and soon we were all thinking of devising an NFL football forecasting program. Bob, our resident circulation guru, knows all the players and all the scores. He brought in a stack of sporting magazines with all the stats and we have been fumbling through them looking for something significant. It's too early to make any definite promises, but with any luck we should have a program ready to publish in our Sep/Oct 86 issue, just in time for the new season. There is a problem with all those statistics, but maybe we could put them on the download each week — hmm. Well, just a thought.

You have probably noted that CodeWorks does not continue an article or program from the front of the magazine to the back. That is mostly due to the fact that I don't like articles that do that. Besides, we have no need to draw your attention to the back pages since we do not have advertisers. Our only restriction on where things go is where the color pages fall. We only use one additional color and it falls on specific pages, so we need to lay the magazine out to take advantage of them. This issue fell together rather tightly, and when all was said and done there were a couple of pages left over. Terry suggested using them for an overlay for the Card program from our Nov/Dec 85 issue to make the Check program in this issue. It worked out well, except that we needed another half page or so for the text, which happened to be a few pages further on and was scheduled for programming notes. I finally broke down and continued the article there, much as I didn't like to, and hope you don't mind either. The missing notes will get first priority in the next issue.

What with the lightning storms, holidays and all, we have let our author file slide a bit. I noted that and took care of it. From here on, we will mark the calendar and take care of it on time.

We hope you like the issue, and will see you next time.

— Irv

Forum

An Open Forum for Questions & Comments

I noticed a reader asked for an article on PEEK and POKE and I would like to second his request. I understand what the two commands are, but how and why do you use them? Why should I want to POKE a number into memory?

I approve of your idea to mix beginning and advanced programs. Your first issue with the Trend Plotter was useful. I am a diabetic and thus must get blood out of my fingers and measure it to control my insulin shots. By using your program and modifying it to fit my needs, my doctor claims he has a better idea of what is going on in my body. I was using a stat program which gave the same information but (he likes this better). Frankly, it has been a real medical help for me...

James C. McCord
Fairbanks, AK

That's a rather unexpected use of the Trend Plotter program. We are very happy it is of help to you. In regards to the PEEK and POKE: Both these BASIC commands provide for direct access to memory. They are very machine specific because the memory usage and layout varies considerably from computer model to computer model. PRINT PEEK(memory location) will return a number between 0 and 255. Or you can say: A=PEEK(memory location): PRINT A. Memory location is a decimal number. PEEK will tell you what value is currently being held in any memory location. POKE will allow you to change a memory location to a value between 0 and 255. POKE will get you into all sorts of trouble if you don't know what you are doing. The syntax for POKE is: POKE memory location, value. For example, POKE 18037,65 will put the number 65 in memory location 18037. If that location happens to be where BASIC resides, you might get some rather unexpected reactions. The CHR$ value of 65 is the capital letter A. If you try to POKE into Read Only Memory (ROM) the POKE will be ineffective - it simply will not accept it. POKE comes in very handy when you want to stuff a machine language routine into high memory. The data to be entered is put into BASIC data statements. Then, with a FOR...NEXT loop, you can read the data and using POKE put it into the appropriate memory locations. It's an interesting subject. The use of these commands varies from machine to machine, but an article on their use in general may not be a bad idea. Give us a while to get it together.

...Your WOOD program with article is beautiful! Well worth the space it took to print, and then some. It deserved top billing instead of relegation to the back pages... One troublesome item noted: why not use other than variable "O" because of its sighting conflict with zero?... The OUTLINE program is utilitarian and truly clever. I am predicting many readers will have lots of trouble "deciphering" it, mainly because your explanatory material is not in the nature of operating instructions. As of this writing, I'm still in the dark about how to use the EDIT and LOADING functions...

Waldo T. Boyd
Geyserville, CA

In our listings, since day one, the zero is always slashed, the "O" is not. If in doubt, you can check a line number, where you know the number cannot contain an "O" and see what the zero looks like. Some people have the same problem with the 1 and the I. In the WOOD program, the "O" stands for Output Array. But you are probably right, we should stay away from variables that could easily be misread. Your remarks in reference to OUTLINE at first left me with a very large question mark over my head. Then it finally dawned - you expected it to be a working program. It is not. It is an example used to illustrate the idea presented. It could have been any program at all, we just needed a structure to show how to implement the idea. - Irv.

I'm a subscriber to CodeWorks, and I'm puzzled. Today in the mail, I received my second copy of the Premier Issue! An error, obviously. Currently I'm expecting my copy of the Jan/Feb 86 issue. Perhaps it will be along shortly - I hope. Just thought I'd bring this to your attention. Good magazine - I like it.

Ed M. McDonough
Sturbridge, MA

Since early last fall, we have been promoting various lists of computer owners. They come from different list brokers and the possibility of your name being on more than one list is likely. I know it sounds crazy, but the time it takes to go through a big list to find names we already have as subscribers is prohibitive. We spot check for it, but don't actually look for each and every one. If you get an extra issue, please give it to someone you know who may be interested. And while on that subject, in the first two issues we asked you all to tell your friends, and that we needed as many subscribers as we could get. It turns out you did it very well. We have received many new subscribers who told us they were referred to us by a current subscriber. You have all done well, keep it up, and thanks! - Irv.

Issue 2, page 16, says that Mr. Norris' program to convert direct access to sequential files would appear in Issue 3. It didn't. Do you have a copy of it to spare? Many

thanks.

Edward Engberg
Santa Barbara, CA

The programs were apparently lost on the production room floor (they got overlooked by yours truly.) But never fear, I found the omission and in this issue you will find the other three programs. Also, we have put them (all five) together in one program with the first numbered in the 100's, the second in the 200's, etc. They will be on the download system as one program from this issue called NORRIS.BAS. After you have downloaded the program, you can take it apart and save each section as a separately named module.

...Your realistic approach to programming for personal enjoyment is as welcome as a fresh spring breeze. I wonder if you intend to cover a subject known as a B-Tree, Binary Tree, Balanced Tree. Take your pick. It is an alternative to sorting and I understand is conducive to handling extremely large files. I've been looking for some good sound source coding on it, but so far all the writers I've read have presented the algorithm, but not much else... Also, in studying the Staff Projects you have published, I noticed that often you drop out of a FOR...NEXT loop without closing the NEXT. I was always under the impression that you should never leave a NEXT hanging. It will (possibly) foul up things unexpectedly later on. I am looking forward to the next edition with a high degree of anticipation.

Art Phillips
Arvada, CO

I've had a chat with Terry Dettmann, who says he is working on a merge sort that uses the binary tree. Just as soon as he finishes it and we get it all checked out you will see it in these pages. And yes, we jump out of loops. It becomes a form of a "Do Until" loop. I have always heard you shouldn't do it, but can't figure out why. It would make sense if the loop counter would be returned to a zero value on closure but it doesn't. If you jump out ahead of time the loop counter is at some value between the "from" and "to" values. If you close the loop it is at the terminal value plus 1. In any case, thank you. You have given us an interesting lead into an article on loops: FOR...NEXT, WHILE... WEND, and the rest of them. We'll work on it.

...I am pleased to see that you are publishing a sensible magazine that is designed to help developing programmers...I feel very isolated in that when I have a question about a programming problem, I can very seldom find anyone that understands what I am talking about. I will probably be asking you some of these questions from time to time.

What I would like to see you do is publish a number of modules that could be merged, as desired, into a program for a specific purpose. It would be interesting to have readers submit their ideas for modules for a menu, sorts, searches, editing, right and left justification of lines of text, etc. This would probably turn up a number of new ideas in programming, like the one in (Issue 2) that suggested using STRING$(X,13) instead of a FOR...NEXT loop for multiple line feeds. (It was new to me.) I look forward to many months of enjoyment through CodeWorks and hope to communicate with your on-line computer shortly.

E. L. Stanley
Clarkston, WA

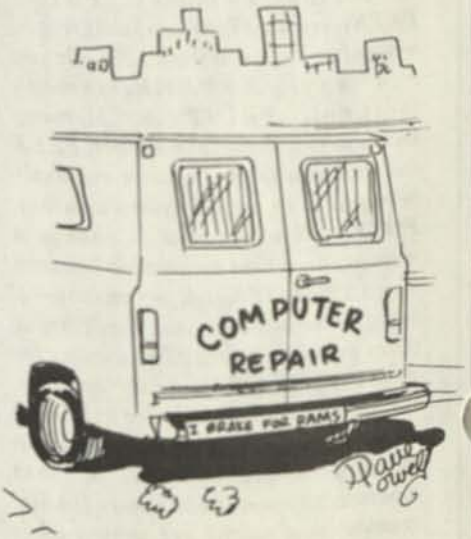
You are probably right, but BASIC's lack of local and global variables, among other things, make it somewhat difficult to implement. Still, not a bad idea. Perhaps we could designate all the variables starting with "Z" as reserved for common subroutines? It is worth some serious thought.

I am just now reading the Sep/Oct 85 issue and am delighted with it. Can you develop a routine so a printer will add a line feed in between each line listed when listing out a program? It would sure make it easier to write editing notes in the blank spaces..

William H. Fox
Oroville, CA

Most printers have a switch setting that will allow you to add a line feed in addition to the one sent from the computer at the end of a program line. If you have MS-DOS, it probably will do the line feed even when you have the printer set not to. (In that case there is usually a file called LF which you can use to include or exclude the line feed - LF OFF turns it off, LF ON turns it on.) If you can do neither of these to get your line feed, try saving your program in ASCII - SAVE "filename",A and then load it with your text editor and let it print it out. Most text editors/word processors will allow you to set the line spacing. These are not the only or best ways to do it; there must be several others.

A reader from Norwalk, CT has written to tell us he has converted our program CARD.BAS into a key inventory program. The problem he had was that there were 80 keys issued to different people and there were 12 levels of security. Not all keys fit all locks. He has reworked the program to account for the keys and who has them, as well as to tell when any given security level key needs to be reordered. If you have a similar problem and would like to contact him directly his name and address is: Arthur J. Avery, 1 Redbird Lane, Norwalk, CT 06854



Check.Bas

Organize your Checks for Tax Time

Staff Project. Sorting out the checks you wrote last year is a snap with this simple overlay to a program we published previously. It summarizes your expenses by category and gives you a sorted list of all checks.

Need a starting point for income tax time? Got that shoebox full of receipts in the bottom drawer? How about finding all those checks you wrote and forgot about? Getting your figures organized and ready for that form 1040 can be pure drudgery. Maybe this will help you.

Check.bas is a program that will let you enter all of your checks (either from the stubs or from the cancelled checks themselves.) It doesn't matter if they are not in order, the program will put them into any order you want. All you need do is enter who the check was for, what the check was for, the check number and the amount. You can then sort on any of these items and print out a neat list. More important is the fact that you can also summarize the checks by category and print subtotals by category and a grand total.

You may well ask where these few lines of code get all of that ability. Well, they don't have it all by themselves. These lines are merge lines that fit right over the top of a program we published in Issue 2, called Card.bas, in Nov/Dec 1985. It is a mini-database program, and these lines, we'll call them CHECK.MRG (for Check Merge), will make it all possible.

There are a couple of ways to do this. Let's start by saying that you must have Card.bas stored on your diskette with the exact line numbers shown in Issue 2 and nothing added or deleted. Then you type in the lines in this program (CHECK.MRG) as though it was a program all by itself. When done, save this program as an ASCII file — SAVE"CHECK.MRG",A. Then load your Card.bas program and at the ready prompt type MERGE"CHECK.MRG" and enter. Follow the directions in Issue 2 for initializing Card.bas, and when you have done so, you will find that it is no longer Card.bas but is now called Check.bas. Keep

Continued on page 25

Program CHECK.MRG

```
100 REM ** THIS IS CHECK.BAS * WRITTEN FOR CODEWORKS MAGAZINE **
102 REM * THIS IS A MERGED VERSION OF CARD.BAS FROM ISSUE 2
110 V=500: REM SET LIMIT OF # OF CHECKS HERE
120 DIM A$(V),B$(V),C$(V),D$(V),Q$(V),P(V)
160 OPEN "O",1,"CHECK.DAT"
230 '
240 '
250 '
260 '
330 OPEN "I",1,"CHECK.DAT"
420 '
430 '
440 '
450 '
570 A$="1-Who to  :"
580 B$="2-Category:"
590 C$="3-Ch Num  :"
600 D$="4-Ch Amt  :"
610 '
620 '
630 '
```

This program is available on the download as CHECK.MRG. An already merged version of Card.Bas (now called CHECK.BAS) is also available on the download.

```
640 '
670 PRINT"              C H E C K   F I L E   P R O G R A M"
680 PRINT"   organize your checkbook by category for income tax time"
750 PRINT "5 - PRINT Checks or Summary"
940 '
950 '
960 '
970 '
1020 PRINT A$(P(I));TAB(20);B$(P(I));TAB(40);C$(P(I));TAB(50);D$(P(I))
1030 '
1270 '
1280 '
1290 '
1300 '
1440 ON X GOTO 1450,1460,1470,1480
1490 '
1500 '
1510 '
1520 '
1610 '
1640 IF X=>5 OR X=<0 THEN GOTO 1620
1660 ON X GOTO 1670,1680,1690,1700
1710 '
1720 '
1730 '
1740 '
1995 REM *** PRINT LIST OR SUMMARY ROUTINE ***
2020 PRINT "ENTER 1 - TO PRINT THE CHECK SUMMARY BY CATEGORY"
2030 PRINT "ENTER 2 - TO PRINT THE ENTIRE CHECK REGISTER"
2040 PRINT:X$="$$###,###.##"
2060 ON X GOTO 2080, 2230
2080 '
2090 '
2100 FOR I=0 TO L1-1
2110 XX=LEN(B$(P(I)))
2120 L=I+1
2130 IF B$(P(I))=B$(P(L)) THEN CS=CS+VAL(D$(P(I))):GOTO 2170
2140 CS=CS+VAL(D$(P(I))):LPRINT B$(P(I));STRING$(29-XX,"-");
2150 LPRINT TAB(30);USING X$;CS
2160 CT=CT+CS:CS=0
2165 '
2170 NEXT I
2180 LPRINT" "
2190 LPRINT"GRAND TOTAL ----------> $";CT
2200 LPRINT" "
2210 LPRINT"TOTAL NUMBER OF RECORDS EXAMINED ";I
2220 GOTO 650
2230 REM ** PRINT ENTIRE CHECK REGISTER MODULE **
2240 FOR I=0 TO L1-1
2250 LPRINT A$(P(I));TAB(20);B$(P(I));TAB(40);C$(P(I));:LPRINT TAB(50)
;USING X$;VAL(D$(P(I)))
2260 NEXT I
2270 LPRINT" "
2280 LPRINT"TOTAL NUMBER OF RECORDS PRINTED ";I
2290 GOTO 650
```

Precomp.Bas

A BASIC Precompiler using Labels

J. Melvin Jones, Warwick, NY. If the lack of labels and the use of line numbers in BASIC bother you, here is a precompiler that allows the use of labels and does not need line numbers. Now you can write code with your word processor.

BASIC, as it is currently implemented, is an almost ideal language for hacking out programs interactively with a minimum of actual planning and previous specification. The problem, however, is the assumption that the programmer can assign and remember a specific line number for each and every statement in the program. While this is not difficult for programs up to about 100 lines, modern BASIC implementations often allow programs of considerably longer length and, correspondingly, complexity.

Thus, most programmers, myself included, must surrender the "off-the-cuff" nature of BASIC and diagram more complex programs in advance, thereby predicting the line numbers of various sections of code. Furthermore, for really large programs, the programmer usually must keep a scratch pad handy to document the hundreds of line numbers that he will need to remember at some future time.

Finally, when it comes time to revise or update the program the programmer must find a set of line numbers that is unused and patch his changes in wherever they will fit, resulting eventually in the "spaghetti code" so commonly found in BASIC programs.

The solution is, at least conceptually, obvious. Rather than assigning each statement a specific number, and therefore an irrevocable identity, it is preferable to assign mnemonic labels only to those lines which are referenced and, by doing so, delimit *meaningful* sections of code and allow the easy insertion and deletion of code without disturbing the integrity of the program.

Unfortunately, BASIC as specified does not allow this and, understandably, most compilers and interpreters don't either. This is the justification for the BASIC Source Code Precompiler.

This is how it works. First, using your favorite text editor or word processor, write your program using labels instead of line numbers. To do this, reserve the first eight columns of each statement as the Label Field. If the statement is to be labelled, left justify a sequence of up to eight characters in this field. A label may consist of any combination of ASCII characters *except* the space. If the statement is not labelled, fill the first eight characters with spaces (i.e., leave the first eight spaces in the line blank, by using the space bar or the TAB key.)

Whenever and wherever you would normally use a line number in the program, use the label of the statement referenced instead. Examples would be GOTO, GOSUB, IF...THEN...ELSE, and ON...GOTO statements. Whenever you use a label in a program line, make sure that it is surrounded by spaces, unless, of course, it is the last item in the line, in which case it can be followed immediately by the carriage return that terminates that statement.

Once the program is ready to be tested, save it in a sequential ASCII file and run the BASIC program PRECOMP.BAS. When asked for the source file, specify the file which contains your program (the version using labels.) When asked for the object file, name the file which is to contain the version of your program with line numbers (and therefore executable by your BASIC interpreter or compiler.) The BASIC Source Code Precompiler will make two passes through the source file, assign line numbers wherever needed, and write the resulting BASIC program into your sequential ASCII object file.

Finally, load and run the object file under your regular compiler or interpreter. If you need to make repairs or revisions, you can either make them to your original source file (recommended) or simply patch the resultant program (simplified by the Symbol Reference Listing made available by the Precompiler.)

Limitations? Only that you not assign a label

identical to a BASIC keyword or a variable name defined in your program. The Precompiler will not change occurrences of your labels within quotation marks (lexical constants) or in REM statements. No checking is performed to determine if a label is identical to a BASIC keyword or variable name, so any occurrence of an identical match will be converted to the corresponding line number. To avoid this problem, if it becomes a problem, I would suggest that you precede the troublesome labels with a dollar sign ($). Since no BASIC keyword or variable name usually begins with $ the label will be unique and never be confused with any other token.

This program was written for the Tandy Model 4 operating under TRSDOS Version 6.01.01 and BASIC version 1.00.00, but should easily be convertible to almost any other BASIC dialect or machine. The only requirement is that the target BASIC interpreter or compiler must be able to load and execute programs stored in ASCII files. I have never run across a system that could not, although some require that you set a special flag in the file header. Careful examination of the manual for the target system should provide all the necessary information.

(Ed. note: To make this program more transportable, we removed two or three WHILE...WEND loops and converted them back to straight coding. In addition, since many of our readers have computers that do not use SPACE$, we converted that to STRING$. The program, as presented here, has been successfully run on IBM-PC, Sanyo 555, Tandy Models 1000, I, II, III, 12 and 16. It was also checked and runs under CP/M MBASIC.) ■

```
100 REM ** PRECOMP.BAS * Written for CodeWorks Magazine, 3838 South
110 REM ** Warner St. Tacoma, WA 98409 (206)475-2219
120 REM ** For program details and instructions see CodeWorks Issue 4
130 REM ** Please do not remove the above credit lines.
140 ' ------------------------------------------------------------'
150 '          BASIC Source Code Precompiler by Jeffrey M. Jones
160 '
170 '  This program reads a prepared ASCII BASIC source file from
180 ' diskette, evaluates labels, and then numbers the program as
190 ' required by the BASIC interpreter. The executable code is then
200 ' written to a specified disk file.
210 '------------------------------------------------------------'
220 DIM L$(1024), L(1024)
230 CLS
240 PRINT"BASIC Source Code Precompiler. (Version 1.00.00)"
250 PRINT"Written by Jeffrey M. Jones, 4 October 1985."
260 PRINT
270 LINE INPUT"Source File: ";S$
280 LINE INPUT"Object File: ";O$
290 PRINT
300 PRINT"Pass 1..."
310 OPEN "I",1,S$
320 L=100
330 P=1
340 IF L>65500! OR P>1024 OR EOF(1) THEN GOTO 450
350   LINE INPUT #1,A$
360   GOSUB 1020
370   IF(LEN(A$)<8) OR (A$=STRING$(LEN(A$)," ")) THEN GOTO 440
380   IF LEFT$(A$,8)=STRING$(8," ") THEN GOTO 430
390   IF INSTR(A$," ")>8 THEN L$(P)=LEFT$(A$,8)
400   IF INSTR(A$," ")<9 THEN L$(P)=LEFT$(A$,INSTR(A$," ")-1)
410   L(P)=L
420   P=P+1
430   L=L+10
440 GOTO 340
```

```
450 CLOSE 1
460 IF (P>1024) THEN PRINT"** too many labels":END
470 IF (L>65500!) THEN PRINT"** too many lines":END
480 PRINT"Pass 2..."
490 OPEN "I",1,S$
500 OPEN "O",2,O$
510 L=100
520 IF EOF(1) THEN GOTO 870
530   LINE INPUT #1,A$
540   GOSUB 1020
550   IF LEN(A$)<9 OR (A$=STRING$(LEN(A$)," ")) THEN GOTO 860
560     L1$=RIGHT$(STR$(L),LEN(STR$(L))-1)+" "
570     A$=RIGHT$(A$,LEN(A$)-8)+" "
580     FOR I=1 TO P-1
590       IF INSTR(A$,L$(I))<1 THEN B$=A$:GOTO 800
600         P1=1
610         Q=0
620         E=0
630         B$=""
640       IF P1>LEN(A$) OR E=1 THEN GOTO 800
650       IF Q=1 THEN GOTO 700
660       IF MID$(A$,P1,LEN(L$(I))+2)<>" "+L$(I)+" " THEN GOTO 700
670         B$=B$+STR$(L(I))
680         P1=P1+LEN(L$(I))+1
690         GOTO 790
700       IF MID$(A$,P1,1)<>CHR$(34) THEN GOTO 730
710       IF Q=1 THEN Q=0 ELSE Q=1
720         GOTO 770
730       IF MID$(A$,P1,1)<>"'" OR MID$(A$,P1,3)<>"rem" THEN GOTO 770
740         B$=B$+MID$(A$,P1)
750         E=1
760         GOTO 790
770         B$=B$+MID$(A$,P1,1)
780         P1=P1+1
790       GOTO 640
800       A$=B$
810     NEXT I
820     A$=L1$+A$
830     A$=LEFT$(A$,LEN(A$)-1)
840     PRINT #2,A$
850     L=L+10
860 GOTO 520
870 PRINT
880 PRINT"Transformation completed."
890 CLOSE 1
900 CLOSE 2
910 PRINT
920 INPUT"Would you like a symbol reference list (Y/N)";A$
930 IF A$<>"Y" AND A$<>"y" THEN GOTO 1010
940   LPRINT"Symbol Reference Listing. Source: ";S$
950                                    Object: ";O$
960   LPRINT"
970   LPRINT
980   FOR I=1 TO P-1
990     LPRINT L$(I),L(I)
```

```
990   NEXT I
1000   LPRINT CHR$(12);
1010 END
1020 '
1030 ' Expand tabs. Note: Change CHR$(09) in the following lines to t
he
1040 ' tab character used in your system.
1050 '
1060 TC=INSTR(A$,CHR$(9))
1070 IF TC<1 THEN RETURN
1080 A$=LEFT$(A$,TC-1)+STRING$(8-((TC-1)-INT((TC-1)/8)*8)," ")+MID$(A
$,TC+1)
1090 GOTO 1060
```

Some text editors do not expand tabs into spaces when saving ASCII files. Also, some computers use space compression characters in ASCII files. If this is your case, omit lines 1020 through 1090 in the main program and use the following lines instead.

```
1020 '
1030 ' Expand space compression codes.
1040 '
1050 FOR TC=1 TO LEN(A$)
1060   IF ASC(MID$(A$,TC,1)>191 THEN A$=LEFT$(A$,TC-1)+
       STRING$(ASC(MID$(A$,TC,1)-192+MID$(A$,TC+1)
1070 NEXT TC
1080 RETURN
```

```
        PRINT"ENTER THE BEG BALANCE"
        INPUT BA
START   PRINT"ENTER CK DP OR CH"
        INPUT C$
        PRINT"ENTER AMOUNT"
        INPUT AM
        IF C$="CK" THEN GOTO POINT1
        IF C$="DP" THEN GOTO POINT2
        IF C$="CH" THEN GOTO POINT1
        PRINT"ERROR IN DATA ENTRY TRY AGAIN"
        GOTO START
POINT1  AM=-AM
POINT2  BA=BA+AM
        PRINT"AMOUNT",AM,"NEW BAL",BA
        GOTO START
```

Left. Our test sample program as written with WordStar. We used the tab and set it for 8 characters.

Below. The symbol reference listing produced for the program at the left.

```
Symbol Reference Listing.  Source: PRECOMP.TXT
                           Object: PRECOMP.OBJ

START          120
POINT1         210
POINT2         220
```

```
100 PRINT"ENTER THE BEG BALANCE"
110 INPUT BA
120 PRINT"ENTER CK DP OR CH"
130 INPUT C$
140 PRINT"ENTER AMOUNT"
150 INPUT AM
160 IF C$="CK" THEN GOTO 210
170 IF C$="DP" THEN GOTO 220
180 IF C$="CH" THEN GOTO 210
190 PRINT"ERROR IN DATA ENTRY TRY AGAIN"
200 GOTO 120
210 AM=-AM
220 BA=BA+AM
230 PRINT"AMOUNT",AM,"NEW BAL",BA
240 GOTO 120
```

Left. This is the executable code produced by Precomp.Bas using the sample program on the facing page.

Programming Notes

In our last issue we printed a programming note in reference to changing LOCATE to PRINT@. Hogwash! The idea was great, the math was terrible. I'd like to blame it all on the typesetter, but that was me - darn. Well, in the meantime, we have very carefully sat down and calculated (and checked) what the math should have been. Here it is:

To convert from an 80-column LOCATE Row,Col position to an 80-column PRINT@ location: PRINT@ = ((Row-1)*80)+(Col- 1).

To change from PRINT@,P to LOCATE Row,Col: Row = INT(P/80)+1 and the Col = P-((Row-1)*80)+1

Our thanks to JoAnn Blume of Seattle for not only pointing this out but providing the correct math.

Have you ever typed a number (outside the range -32767 to 32767) and found upon listing your program that BASIC has appended the exclamation mark after it? It just happens to have occurred in at least two programs listed in this issue. Why? Because they are usually numeric constants. Numeric constants are values input to a program that are not subject to change. Numeric constants cannot contain punctuation. The number 980,000 will not work but 980000 will. They are evaluated when they are entered, and if they are out of range, an error message is usually generated. The "!" declares a single precision number. In the Payroll program in this issue, for example, we have a line that says: FM=37800. Since it is larger than 32767, BASIC simply prints an exclamation point after it to indicate it is single precision. Some computers will do this automatically, others will not. When you type these programs into your computer, do not type in the exclamation point. Let the computer put it there if it really wants it. If you download our programs via the CodeWorks download, the program will come to you with the exclamation mark in it. It may or may not cause an error. If it does, simply remove it.

The "FIND" utility in MS-DOS is very handy. It will search an ASCII file and find every occurrence of a specified character string. If, for example, you wanted to see all of your remark lines in a program, save the program in ASCII and then from the DOS ready prompt, type: FIND "REM" filename. That will print all the lines with the REM on your screen. If you want a printed listing of those lines, do this: FIND "REM" filename > prn. It will then list the lines on your printer.

Piles to Files

Programs to Switch File Types

William L. Norris, Edmonds, WA. These are the other three programs which go with the first two published in our Nov/Dec 85 issue. All five are on the download for this month. The first one is numbered in 100's, the second in 200's, etc.

Program 3 — Transfer Direct Access to Sequential

```
300 CLS
302 PRINT"TO TRANSFER A DIRECT ACCESS FILE TO SEQUENTIAL"
304 CLEAR 25000: ' USE ONLY IF YOU NEED TO CLEAR STRING SPACE
306 PRINT"A$=";FRE(A$);"  ";"A=";FRE(A)
308 DIM A$(350)
310 INPUT"NAME OF DIRECT ACCESS FILE ";DA$
312 LINE INPUT"NAME OF SEQUENTIAL FILE ";SQ$
314 INPUT"LENGTH OF LINES (LRL)";L%
316 INPUT"IS LRL FIXED OR PREALLOCATED (Y/N) ";YN$
318 IF YN$="Y" THEN GOTO 320 ELSE GOTO 322
320 OPEN "R",1,DA$,L%:GOTO 320
322 OPEN "R",1,DA$
324 OPEN "O",2,SQ$
326 RD=LOF(1):PRINT"RD=";RD
328 INPUT"BEGINNING AND ENDING NUMBERS (B,E)";B,E
330 IF E-B+1>350 THEN PRINT"LIMIT IS 350":GOTO 328
332 FIELD 1,L% AS WD$
334 FOR Y=B TO E
336   GET 1,Y
338   X=X+1
340   A$(X)=WD$
342   PRINT#2,A$(X)
344   PRINT X;
346   PRINT WD$
348 NEXT Y
350 CLOSE
352 PRINT"LINES ";B;" TO ";E;" HAVE BEEN MOVED FROM ";DA$
354 PRINT"TO SEQUENTIAL FILE ";SQ$;" LINES (1 TO";X;")"
356 PRINT"LRL=";L%
358 INPUT"PRESS ENTER TO CONTINUE, 99 TO END";SY$
360 IF SY$="99" THEN GOTO 364
362 GOTO 300
364 END
```

Program 4 — Combine two Direct Access files

```
400 CLS
402 CLEAR 25000: ' USE ONLY IF YOU NEED TO CLEAR STRING SPACE
404 PRINT"TO COMBINE TWO DIRECT ACCESS FILES"
406 INPUT"NAME OF FIRST DIRECT ACCESS FILE";DA$
408 INPUT"NAME OF SECOND DIRECT ACCESS FILE";DB$
410 INPUT"LENGTH OF LINES (LRL)";L%
```

```
412 OPEN "R",1,DA$
414 OPEN "R",2,DB$
416 FIELD 1,L% AS WD$
418 FIELD 2,L% AS WC$
420 R=0:R1=LOF(2)
422 R=R+1:R1=R1+1
424 IF R>LOF(1) THEN GOTO 434
426 GET 1,R
428 LSET WC$=WD$
430 PUT 2,R1
432 GOTO 422
434 CLOSE
436 INPUT"PRESS ENTER TO CONTINUE, 99 TO STOP";SY$
438 IF SY$="99" THEN END
440 GOTO 400
```

Program 5 — Convert Sequential to Direct Access and Append

```
500 CLS
502 CLEAR 25000:   USE ONLY IF YOU NEED TO CLEAR STRING SPACE
504 PRINT"TO TRANSFER SEQUENTIAL FILE TO DIRECT ACCESS FILE"
506 LINE INPUT"NAME OF SEQUENTIAL FILE. ";SF$
508 INPUT"HOW MANY CHARACTERS PER LINE (LRL) ";L%
510 DIM A$(350)
512 OPEN "I",1,SF$
514 IF EOF(1) THEN GOTO 524
516 X=X+1
518 LINE INPUT #1,A$(X)
520 PRINT X;A$(X)
522 GOTO 514
524 CLOSE
526 L=X:PRINT"THERE ARE ";L;" LINES IN THE SEQUENTIAL FILE"
528 LINE INPUT"NAME OF DIRECT ACCESS FILE. ";DF$
530 INPUT"HAS FILE SPACE BEEN FIXED OR PREALLOCATED (Y/N)";DS$
532 IF DS$<>"Y" THEN GOTO 536
534 OPEN "R",2,DF$,L%:GOTO 538
536 OPEN "R",2,DF$
538 RD=LOF(2):R=RD
540 PRINT"THERE ARE ";RD;" LINES IN THE DIRECT ACCESS FILE ";DF$
542 FOR X=1 TO L
544    R=R+1
546    FIELD 2, L% AS WD$
548    LSET WD$=A$(X)
550    PRINT USING"####";R;
552    PRINT" ";WD$
554    PUT 2,R
556 NEXT X
558 CLOSE
560 PRINT"SEQUENTIAL FILE ";SF$;" HAS ";L;" LINES NOW IN DIRECT ACCES
S ";DF$
562 PRINT"WHICH HAS ";R;" LINES AFTER ADDITION"
564 PRINT"PRESS ENTER TO CONTINUE, 99 TO STOP";:INPUT SY$
566 IF SY$="99" THEN END
568 GOTO 500
```

Beginning Basic

A loop is one of BASIC's most powerful tools. It automatically gives you the ability to perform a given operation a specified number of times. The most common form of a loop in BASIC is the FOR...NEXT loop. Take a look at the following program lines:

```
10 FOR I = 1 TO 10
20    PRINT I;
30 NEXT I
40 END
```

When you run this code, it will print:

1 2 3 4 5 6 7 8 9 10

The semi-colon in line 20, after the I, causes the print on the same line. Without it each number would be on a new line. In this program, I is called the *loop counter*. The counter of the FOR statement is incremented from its initial value until it reaches the final value. In between these two values, line 20 prints the *current* value of I on the screen. FOR I = 1 to 10 means that the value of I will repeat the loop until I is greater than 10. When I is incremented to 11, the NEXT I will be ignored by BASIC and the program flow will continue at line 40. In general, in a FOR...NEXT loop, if allowed to complete the loop the number of times specified by the "from", "to", the loop counter will always be one more than the terminal (or "to") value. What this says is that in this type of loop, we do the operation at least once, and then check to see if we have arrived at the limit.

To get the picture of how the loop works, here is the way the previous program would be done without using FOR or NEXT:

```
10 I = 1
20 IF I >10 THEN GOTO 50
30    PRINT I;
40 I = I+1:GOTO 20
50 END
```

This code will do the same thing as our previous example, and shows what really happens inside the FOR...NEXT loop.

What happens inside the loop does not necessarily have to be something connected to the loop counter. Most of the time it is because it is the most convenient way of handling things. Further, more than one statement can be included inside the loop - you can operate on several things at the same time.

In an incrementing loop, as in our first example, a step of plus 1 is automatically assumed. You have no need to tell BASIC this, it just does it. But if you only wanted every other item between 1 and 10 the FOR statement would have said: FOR I = 1 TO 10 STEP 2. You can also *decrement* a loop. In this case the FOR statement looks like this: FOR I = 10 to 1. But now, the -1 is not assumed, and you must always indicate the step. So now the statement should read: FOR I = 10 TO 1 STEP -1. Naturally, you can use a step of -2 if you want every other value instead of all of them. You can also start at a negative value, go through zero and end up at a positive value: FOR I = -5 TO 5 will give you the whole numbers between minus five and plus five.

All the values in the loop, the counter, upper and lower limits and the step size, must be single precision or integers.

Loops may be "nested" almost without limit. This means that inside one loop you can have another loop operating. The innermost loop will always repeat its given number of iterations before the outer loop changes once. This is true no matter how "deep" the loops are nested. There are many times, especially in handling arrays with two or more dimensions, where nesting is very desirable and easy to implement. Here is a simple example of nesting:

```
10 FOR I = 1 TO 3
20    FOR J = 1 TO 5
30       PRINT J;
40    NEXT J
50 PRINT
60 NEXT I
```

This will print 1 2 3 4 5 three times. The print statement in line 50 is necessary to keep the three groups of 1 to 5 from printing on the same line. By the way, indenting the lines inside a loop is very good practice, since it shows the content of the loop clearly when reading the code.

In spite of what we said earlier about these loops executing at least once (because it checks for the terminal count *after* it has gone through the loop once), Microsoft's BASIC, release 5.0 and later, has this note about loops: "The body of a FOR...NEXT loop is skipped if the initial value of the loop times the sign of the step exceeds the final value times the sign of the step." This represents a change from earlier releases of this BASIC and may be something to be aware of. ■

Sequential Files

How Buffers Connect Memory to Disk

Terry R. Dettmann, Associate Editor. When a computer is turned off the data in memory is lost. Storage on diskette makes it possible to regain access to important information. This article describes one of two methods of storing information on diskette. It has roots that go back to punched cards and magnetic tapes, but is still a very useful filing method. See Payroll.Bas in this issue for some practical application of Sequential files.

Most programs of any real significance are concerned with storing information for later use. Business programs store accounting information. Engineering programs often store data. Even Adventure programs sometimes store information needed about the scenario.

The way we store information is in data files. These are collections of information on some storage medium such as disk or tape. We are going to take a look at simple disk files and find out how to set them up as well as some hints on how to use them.

This article is not all-inclusive. You should be familiar with the material in your DOS manual, or at least have it handy to check on things we say and do. Make sure you read about how each BASIC statement is written for file handling. If you take the time, files can be a useful addition to your programming skills.

There are other types of files, one of them being the random access file. However, this article will only cover sequential files.

What is a File?

We all have some idea about what a file is, but let's be a little more precise so that we are all talking about the same things.

In the early days of computing, everything was done on punched cards which were stored in large filing cabinets designed specifically for them. Gradually, the usage of the word "file" transferred from the cabinet to the decks of cards themselves.

With the introduction of tapes, it became natural to refer to information on tape as a file. Most often, the information was stored as a sequence of card images meaning that the tape record was built exactly like a card deck.

Both of these methods of storage had one major quality in common, they were both sequential storage methods. That is, both cards and tapes were read one card (or data record) at a time. In order to look at the 124th record, it was necessary to first read the 123 records before it. Woe to the programmer who dropped a card box and lost the order!

With the introduction of disk storage, access to files could be done much faster than before. At first, files were still stored sequentially. But, the unique feature of the disk which makes it a really powerful tool is that another, better type of file can be designed.

On the first computers (and again on the first microcomputers) it was the responsibility of the programmer to keep track of the locations of his files. This required a great deal of programming to make sure that files were kept separate and that the location of each bit of information was known.

It wasn't long before someone recognized that the computer could just as well (probably better) do the work. This gave rise to the concept of the Operating System, or Disk Operating System (DOS). The operating system takes care of the where and the record keeping; we only need to worry about what is in the files.

Many people use sequential access data files because they are simple to use and do not require you to learn many new techniques; you just have to read and write from a new place.

Before we start talking about sequential files

though, we have to talk first about file buffers.

File Buffers

What is a file buffer? Why do we need them? How are they made and what do they look like?

A file buffer is a region of memory that is set aside to receive information from your program until an efficient sized packet is made for writing to an output file on the diskette. Or, the other way, a buffer is used to hold information from a diskette input file for your program to access.

Why have a buffer? The problem of buffers came up years ago to speed up access to information and make it more efficient from both the program side and the external storage side. The buffer exists to match the radically different access speeds of an input/output device to the computer itself.

It would be very slow indeed if every time we printed a string, each letter had to be transferred to diskette one at a time! To make the most efficient use of the disk, we should have large amounts of information ready to go to the diskette all at once. To make programming clear however, we want to get only small bits of information at a time.

The buffer matches these two radically different requirements as efficiently as possible. To do this, an area of memory is set aside to be an image of a section of the diskette. For disk read and writes with most Microsoft (and other) operating systems, this is 256 bytes (plus some system space for file information.)

Every time you write information to diskette by using a PRINT statement, you are actually putting the information in the buffer assigned to that file. Only when the buffer is full will the information be written to its appropriate place on the diskette.

To see this in action, try the experiment in program 1. It inputs lines you type in at the keyboard and writes them to diskette. By waiting for the disk to stop between each line, you will find that if you keep your lines short, the disk will not start on every line, even though each line is PRINTed to the diskette when it is read in.

Further, after 10 lines are in, the system will read in each of the lines one at a time. If you wait for the disk to stop between each read and each line doesn't fill a buffer, then the disk will not start for every line you ask to be displayed.

But where do buffers come from? Would you believe that something you do creates them, even though you may not know it? In some disk operating systems, you need to answer the "HOW MANY FILES?" question with a number or simply

Every time you write information to diskette by using a PRINT statement, you are actually putting the information in the buffer assigned to that file.

ENTER. Others come up automatically allocating three (or just one) files unless you specify otherwise. These all create the space necessary in memory for the correct number of file buffers, even if you don't use them. Check your DOS or BASIC manual for the exact way to specify the number of file buffers for your particular computer.

In order to use any file the file must first be opened. The OPEN statement is provided to do this in BASIC. By opening a file, we take one of the available file buffers and use it as a connector between the file and your program. The OPEN statement looks like this:

OPEN "mode",filenumber,"filename"

Where mode is I if the file is sequential and being used for input, or it is O if the file is sequential and being used for output. (That's "eye" and "oh", not one and zero.) Filenumber is the number of the buffer to assign to the file, and "filename" is the name of the file that is (or will be) stored on the diskette.

When a file is opened, the file named in the open statement is connected to the buffer and file information is funneled through there. When the buffer is used, either to put information into the file or to get it out, that information is first put into the buffer and then picked up by the program in INPUT statements or written to the diskette when the buffer is full.

When you are done with the file, you use the CLOSE statement to flush all information out of the file buffer and onto the disk. If you do not close a file properly, some information may still be in the buffer, including updating information for the directory if the file was increased in length.

Sequential access files work just like files on tape, except that a few extra commands are needed to use them. We first OPEN the file for use, then INPUT or PRINT information from or to the file as desired.

Problems which show up in reading a sequential file are invariably caused by incorrect *writing* of that file in the first place. The error is not where it seems to be.

To create a sequential file, we first OPEN the file for output. Let's say we want to create a file called "MAIL/SEQ". If we assign it to buffer 1, then we use this statement:

```
OPEN "O",1,"MAIL/SEQ"
```

This makes the file available to our program. To write to the file, we simply use the standard PRINT statement, modified to indicate which file we want:

```
PRINT#1,NM$,ADR$,CTY$,ST$,ZP$
```

Easy, right? Well, we goofed (did you notice?) A PRINT statement like this one writes each item in the print list to the file the same as if it were going to the printer or the screen.

No problem you say? It does not produce an error. But there is a problem when we try to input the items later. Look at Figure 1. Here is how the record looks for the strings given. Each of the strings we printed to the file will be separated from the others by spaces. *But when we input the strings later, BASIC will think this is only one string!*

In an INPUT statement, BASIC will add to a string until it finds a comma or the end of the record. But there are no commas here. To get around this, when using strings, we have to put the commas in the file explicitly, like this:

```
PRINT#1,NM$+",",ADR$+",",CTY$+",",
        ST$+",",ZP$
```

Notice that there are two ways to put the commas in, either add it to the string or put it in separately.

With numbers, we don't need the commas since a number is automatically stopped when a blank space is reached. But a number after a string still must be separated by a comma, like this:

```
PRINT#1,NM$+",",A
```

Without the comma, the number that should be A would be read into an INPUT statement with string NM$. That would cause an error when the system tried to read A since there is nothing left in the record.

One of the most confusing things about sequential files is that you can output almost in any form without error. Then when you try to read the information back in, you get "Input past end" or "Illegal function call" errors. *Usually, the problem is not in the INPUT lines where the error occurs, but is in the PRINT lines where you put it out to the diskette.* One way to see what you have output is to go to DOS level and use TYPE (or on some machines, LIST) the file name. Since the file is in ASCII, you can see the layout of the file on your screen.

After a file is created, it must be closed to get all the information out of the memory buffer and to the diskette. Then we can OPEN it for input.

To open a file for INPUT with sequential access, we use the statement:

```
OPEN "I",1,"MAIL/SEQ"
```

To bring in a record of information, we use the statement:

```
INPUT#1,NM$,ADR$,CTY$,ST$,ZP$
```

This will bring in one record from the file and put the information in the strings in the input list (remember this will only happen correctly if the commas are in the record on the diskette.)

It often happens that we wish to bring in things from diskette with commas. For example, in writing a text editor, we could make each line a string variable. But then there might be commas in the lines. To get everything in a record as one string variable, we use the command:

```
LINE INPUT#1,LN$
```

Everything in the record will be put in the string LN$. You can try this easier than you think by saving a BASIC program on disk in ASCII format (put a comma and an A after the last quotation mark of the filename.) Program 2 illustrates how to do this with BASIC program files. This works because BASIC programs can be stored on diskette as ASCII files.

On input from a file, we have a special problem. Where is the end of the file? If we read in more records than are in the file, our program dies with

an error from BASIC (usually INPUT PAST END). To prevent that, we could put the number of records to be found in the first record of the file, but that gets to be a bit cumbersome.

BASIC provides us with a simpler way to find the end of a file. We simply check for the end of file with the EOF function. You can imagine this as a true/false function. It is false until we reach the end of the file, then it is true. To use it, we check it just *before* we are about to input a record. If the file is at the end of file, EOF is true. In that case we skip to some other processing, otherwise we can read the record. Program 2 illustrates this use of the EOF function. Once we are done with a file for input, we close it, as we did the output file.

Whenever you open a file, a pointer is created which points to the first record in the file. If we opened the file for INPUT, this is fine, since we will read the records in order until we get to the one we want.

If the file is opened for OUTPUT, the only way to move the pointer is by printing new records to the file. By doing this, we are writing over the old records. In order to add to the end of a file, we either have to bring the whole file into memory and then write it back with our additions, or we have to copy it, record for record, to a new file which we keep open for output. Either way, it is workable but somewhat inconvenient.

The payroll program, in this issue, makes use of sequential files in a practical application. Notice in that program that we do not need to create as many file buffers as there are employees. Only one buffer is used for all files. This is because we never have more than one file open at once. The rule to remember is that you must have as many file buffers as you expect to have open files at the same time. ■

Sequential file Demo Program 1

```
100 REM ------------------------------------------------
110 REM  DEMO PROGRAM 1
120 REM  SEQUENTIAL FILES
130 REM
140 REM  FILENAME: PROG1.BAS
150 REM ------------------------------------------------
160 CLS:CLEAR 5000 : ' CLEAR ONLY IF YOUR MACHINE NEEDS TO.
170 SC=63:GR=45:MD=20
180 PRINT STRING$(SC,GR):PRINT TAB(MD);"FILE BUFFER DEMO"
190 PRINT STRING$(SC,GR)
200 PRINT:PRINT
210 PRINT TAB(10);"OPENING FILE TEMP.DAT"
220 OPEN "O",1,"TEMP.DAT"
230 PRINT TAB(10);"NOW INPUT 10 LONG LINES"
240 PRINT TAB(10);"WAIT FOR DISK TO STOP BEFORE ENTERING A NEW LINE"
250 PRINT
260 FOR I=1 TO 10
270   PRINT I;": ";:LINE INPUT LN$
280   PRINT #1, LN$
290 NEXT I
300 PRINT TAB(10);"CLOSING FILE"
310 CLOSE
320 REM ------------------------------------------------
330 PRINT TAB(10);"OPENING FILE FOR INPUT"
340 OPEN "I",1,"TEMP.DAT"
350 CLS:PRINT STRING$(SC,GR)
360 PRINT TAB(MD);"BUFFER DEMONSTRATION":PRINT STRING$(SC,GR)
370 PRINT:PRINT
380 PRINT TAB(10);"PRESS ENTER TO READ IN A LINE"
390 PRINT TAB(10);"WAIT FOR THE DISK TO STOP BETWEEN LINES"
400 PRINT
410 FOR I= 1 TO 10
420   C$=INKEY$:IF C$="" THEN 420 ELSE IF ASC(C$)<>13 THEN 420
```

```
430   LINE INPUT #1,LN$
440   PRINT I;": ";LN$
450 NEXT I
460 PRINT TAB(10);"CLOSING FILE"
470 CLOSE
480 END
```

Sequential file Demo Program 2

```
100 REM ------------------------------------------------------
110 REM   PROGRAM 2
120 REM   SEQUENTIAL FILES DEMONSTRATION
130 REM
140 REM  FILENAME: PROG2.BAS
150 REM ------------------------------------------------------
160 CLS:CLEAR 5000: ' USE CLEAR ONLY IF YOUR MACHINE NEEDS TO
170 SC=63:GR=45:MD=35
180 REM ------------------------------------------------------
190 PRINT STRING$(SC,GR):PRINT TAB(20);"READ AND DISPLAY FILES"
200 PRINT STRING$(SC,GR)
210 PRINT:PRINT
220 PRINT TAB(5);"ENTER THE NAME OF AN ASCII FILE (EITHER A"
230 PRINT TAB(5);"PROGRAM FILE SAVED WITH THE 'A' OPTION OR A"
240 PRINT TAB(5);"FILE WRITTEN TO DISK BY ANOTHER PROGRAM)."
250 PRINT
260 PRINT TAB(10);:LINE INPUT"FILENAME: ";FF$
270 OPEN "I",1,FF$
280 I=0
290 IF EOF(1) THEN GOTO 340
300   I=I+1
310   LINE INPUT #1,LN$
320   PRINT I;": ";LN$
330 GOTO 290
340 CLOSE
350 END
```

With these strings:

```
NM$="CODEWORKS MAGAZINE"
ADR$="3838 S. WARNER"
CTY$="TACOMA"
ST$="WASH"
ZP$="98409"
```

the file will look like this:

```
CODEWORKS MAGAZINE    3838 S. WARNER  TACOMA  WASH    98409
```

Figure 1

Math.Bas

Making Mathematics come Alive

Staff Project. In the days of teletype output this program would have been impractical if not impossible. Because the video screen allows selective placement of the cursor we can now present a math formula as it would appear in a book or on a blackboard. From there it is just one step more to making it readjust to new values input from the keyboard.

Mathematical relationships chalked on a blackboard in school or in a textbook have a way of being inert. They tell the story, but as they get more complex it is difficult to see the effect of any given element of the formula. It seemed like a natural thing to put "live" formulae on a computer, and most of us do, but we no longer look at the formula, only the answers.

Most of us can be divided into two groups: those who intuitively "know" or "feel", and those who are analytical and get to step three by following rules through steps one and two. Consequently, seeing a mathematical formula in action may be a ho-hum experience to some and a revelation to others.

The subject of this article is live Algebra. This means that the formula is presented on the computer video screen almost exactly as it would be written on the old chalk board in the little red school house. On our screen though, you can push keys representing the elements of the formula and watch them change, along with whatever else in the formula is then effected by that element, including the answer.

The ability of the computer to do this has been there since the advent of the video screen. There is no tricky programming involved, just the careful placement of the formula on the screen and the use of the BASIC INKEY$ function. The formula is actually on the screen twice, first to show the variable names and their places in the formula and second to give those variables some actual values which can be changed from the keyboard. The first tells which keys to push to increment or decrement the values in the second. Upper case characters will always (by our convention) increase or increment the value, while lower case will decrease or decrement it. Fixed values such as Pi are not included in those values which can change, of course.

The program we present to demonstrate live Algebra is not long (about 120 program lines), and includes three diverse mathematical relationships. The first is the compound interest formula, next is a permutation formula and last is the formula for resonant frequency. They represent an example each from finance, math and electronics. They were chosen for that and also because each has a different form - and we wanted to show that most formulae can be represented like this.

Naturally, some stops and limits had to be incorporated into the program. Without them there are various errors, especially divide by zero and overflow errors. This was most apparent in the permutation formula, where N factorial had to be limited to N = 32. The number gets very big, very fast.

The objective of the program is to get the formula on the screen in its natural, static state showing which keys represent what values in the formula. Then, immediately below it, show it again with some initial conditions given to the variables. The keyboard keys corresponding to the variables should then increase or decrease the values and the second formula should change immediately, showing the new values as well as the new solution. In actual practice, we found that we needed a "speed" key as well, to act as a multiplier, to get values up or down faster. In the two cases where we used this, it is the F key, and provides a times 10 change every time it is pushed. It also shows on the screen, and does not have an effect on every variable but only on those which need it.

How it Works

The idea is basically simple. First we print the static formula on the screen. Then we assign some initial conditions to the variables so that it will

LINE NUM	LOCATE NUM	PRINT@ 80-COL SCREEN	PRINT@ 64-COL SCREEN
540	3,43	202	170
580	12,18	897	721
590	13,1	960	768
600	14,1	1040	832
610	15,1	1120	896
850	6,43	442	362
880	13,1	960	768
890	14,1	1040	832
1240	11,1	800	640
1250	12,1	880	704
1260	13,1	960	768

You may also need to un-remark the CLEAR statement and possibly change the ^ symbol to whatever your machine uses for exponentiation.

This table gives the line numbers of the program which need to be changed, and the values to change, for 80-column screens that use PRINT@ and for 64-column screens using PRINT@. Since most 64-column screens also use only 16 lines instead of 24 lines, the program was written so that no more than 16 vertical lines were used.

come up running with some real values. Next, we go into a loop that prints the formula again, using the initial conditions, and the answer is computed and printed on the screen. Still inside the loop, we use the INKEY$ function to trap the keys representing the values in the formula. If the key is upper case, we increase it a given amount and loop back to recalculate the formula and display it. If the key is lower case, we decrease the value, recalculate it and display it. The ENTER key is trapped as a way to get out of this loop and return to the main menu, where another formula may be chosen for display.

Program Details

Since all three parts of the program are essentially programmed the same way, we will concentrate on just one, the resonant frequency formula. Without getting terribly engulfed in electronics let us just say that a wire coil and an electrical capacitor connected properly will exhibit extremely low resistance (actually in this case, impedance) to some particular alternating current (frequency). Given values for the inductance of the coil and the capacitance of the capacitor, there will be one frequency which will pass with virtually no resistance while all other frequencies will be blocked.

The frequency which passes without resistance is called the resonant frequency. The inductance of the coil is measured in Henrys and the capacitance of the capacitor is measured in Farads. Both of these units were named after famous early pioneers in electricity, and are both so large that in actual practice neither a Farad nor a Henry is seen. The units that normally apply to radio frequencies are microhenrys (or millihenrys) and

microfarads or picofarads.

As a matter of interest, the all-American clock radio by your bedside probably has a fixed coil and a capacitor made up of little plates, half of which are stationary and the other half which mesh with the stationary ones without touching. The knob on your radio which changes the stations is connected to the movable plates of the capacitor. Changing the capacitance changes the resonant frequency of the coil—capacitor combination and lets your favorite AM wakeup station come through. Even though most radios have funny numbers (this is AM 71! Wake up!), the frequency for the standard AM radio is from about 0.5 megahertz to 1.6 megahertz, that's 500,000 Hertz to 1,600,000 Hertz. Hertz used to be called "cycles per second", which is what Hertz now stands for. Standard FM radio and television are much higher in frequency — radar and microwave are even higher. But let's cycle back to the program...

The resonant frequency portion of the program takes place from lines 350 through 670. The first thing we do is clear the screen. Then some values in line 360 are defined as double precision so that they won't jump into exponential notation and look funny in our formula. In lines 370 through 460 we print the formula and some information on the screen.

Line 470 defines Pi and gives our "speed" key an initial starting point at 10. The initial values for L (inductance) and C (capacitance) in our formula are set in lines 480 and 490. In lines 500 through 530 we set some limits. First, we don't let C get too small. Then we don't let L get too small and lines 520 and 530 set the lower and upper limits on our speed key, F. (The F key changes variable SP — for "speed"). In line 540 we run into the first information which is updated on the screen, the speed factor. For those of you who do not have MS-DOS machines, the locate 3,43,0 simply says to print at row 3, column position 43 and turn off the blinky cursor. The sidebar to this article will show the corresponding PRINT@ positions for both 64-column machines using PRINT@ and 80-column machines using PRINT@, as well as a couple of other notes for those machines. Incidentally, this program was originally written on a Sanyo 555 MS-DOS type machine, then transferred via RS-232 to a Tandy Model IV, where it was checked in both Model III and IV modes and the PRINT@ locations were calculated for both. While on the Model III and IV Tandy machines we noted that the values on the screen blinked on and off while they were being updated, while on the MS-DOS type machines they do not, they just change rather smoothly. Although we are not sure, we think this might be due to the fact that the Model III/IV have memory mapped video screens, while on the others the screen is treated like an output device.

Lines 560 and 570 do all of the actual computation of the resonant frequency formula. The following lines, down to 620, print the results on the screen. From lines 620 to 660, we use the INKEY$ to see what key was pressed, and then increment or decrement and use the speed multiplier if applicable. Line 670 closes the loop and sends us back to line 500 where, if there was a change, the whole mess is recalculated and the display is updated. Line 660 checks to see if the ENTER key (CHR$(13)) was pressed. If it was, we go back to line 100 and display the main menu again for another selection.

We should mention something about the exclamation points in lines 530 and 830. Don't type them in. Just type in the number and if your computer wants to put the exclamation point in, let it. Somewhere else in this issue there is a programming note about them. Also, if you should happen to download this program on the CodeWorks Download, the exclamation points may give you syntax errors. Simply remove them if they do.

The other two formulae work in the same manner. The compound interest formula is probably the most interesting one to play with. Now you can easily find the difference between compounding monthly or quarterly. It also gives credence to the old saying that "Them that got, get." Try some big principal numbers and see. (I think it was Einstein who once said that the miracle of modern man was compound interest.)

Finding a general way to put any formula on the screen would be nice, but it is rather time consuming and difficult. You can do the same thing with a loop or with one of the 'Calc programs but this one, we think, has a certain amount of educational value, especially for those of us who can't look at a formula and immediately "see" the relationships. ■

Math.Bas is available on the download system.

```
100 REM ** MATH.BAS ** CODEWORKS MAGAZINE, 3838 S. WARNER ST
110 REM ** TACOMA, WA 98409 ** (206) 475-2219 VOICE
120 REM ** (206) 475-2356 MODEM ** PLEASE DO NOT REMOVE THESE LINES.
130 REM * For conversion to 64 and 80 char screen PRINT@ and
140 REM * program notes see CodeWorks Issue 4
150 'CLEAR 1000: REM USE ONLY IF YOU NEED TO CLEAR STRING SPACE.
160 CLS
170 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
180 PRINT"           M O V I N G   M A T H   P R O G R A M
190 PRINT"              Algebra in action, or AlgeCalc?
200 PRINT STRING$(60,"-")
210 PRINT"  This program contains three different mathematical equati
ons"
220 PRINT"in which you can change various values interactively and"
230 PRINT"see the results while the computer solves the equation.
240 PRINT"  Pressing the upper case key for the letter variable in th
e
250 PRINT"formula will increase that value, lower case will decrease
260 PRINT"it. Pressing ENTER will return you to this menu.
270 PRINT
280 PRINT TAB(10);"1 - Compound Interest Formula"
290 PRINT TAB(10);"2 - Permutation Formula"
300 PRINT TAB(10);"3 - Resonant Frequency Formula"
310 PRINT"Your choice?";
320 X$=INKEY$:IF X$="" THEN GOTO 320
330 X=VAL(X$):IF X<>1 AND X<>2 AND X<>3 THEN GOTO 320
340 ON X GOTO 680,980,350
350 CLS
360 DEFDBL L,C,S,F
370 PRINT"              Resonant Frequency Formula"
380 PRINT
390 PRINT"Use F key to adjust rate of change, now = "
400 PRINT
410 PRINT"Where L is in Henrys, C is in Farads, 2xPI is constant and"
420 PRINT"f is the resonant frequency in cycles per second (Hertz)."
430 PRINT"          1"
440 PRINT"--------------------- = f (resonant)
450 PRINT"        /-----------
460 PRINT"2xPI x \/  L x C
470 PI=3.14159:SP=10
480 L=.0001
490 C=.0000001
500 IF C=<1E-09 THEN C=1E-09
510 IF L=<.000001 THEN L=.000001
520 IF SP<1 THEN SP=1
530 IF SP>100000! THEN SP=100000!
540 LOCATE 3,43,0:PRINT SP
550 S=L*C
560 S1=SQR(S)
570 F=1/(S1*(2*PI))
580 LOCATE 12,18:PRINT"1"
590 LOCATE 13,1:PRINT"---------------------------------- = ";:PRINT
USING"###,#####.###";F;:PRINT"  Hertz"
```

```
600 LOCATE 14,1:PRINT"           /-------------------------"
610 LOCATE 15,1:PRINT"2xPI x  \/ ";:PRINT USING"#.######";L;:PRINT" x
 ";:PRINT USING"#.#########";C
620 K$=INKEY$:IF K$="" THEN GOTO 620
630 IF K$="L" THEN L=L+.000001*SP ELSE IF K$="l" THEN L=L-.000001*SP
640 IF K$="C" THEN C=C+1E-09*SP ELSE IF K$="c" THEN C=C-1E-09*SP
650 IF K$="F" THEN SP=SP*10 ELSE IF K$="f" THEN SP=SP/10
660 IF K$=CHR$(13) THEN RUN 100
670 GOTO 500
680 CLS
690 PRINT"              Compound Interest Formula"
700 PRINT
710 PRINT"Where P = Principal amount, I = Annual Interest Rate, Y is"
720 PRINT"number of years and C = number compounding periods per year
."
730 PRINT
740 PRINT"Use F key to adjust rate of change, now = "
750 PRINT
760 PRINT"                          (Y x C)"
770 PRINT"         P x (1 + I) ^             = Future Value"
780 P=1000
790 I=.05
800 C=4
810 Y=10
820 I1=I/C
830 IF SP>100000! THEN SP=100000!
840 IF SP<1 THEN SP=1
850 LOCATE 6,43:PRINT SP
860 N=Y*C
870 FV=P*((1+I1)^N)
880 LOCATE 13,1,0:PRINT"                                  ";Y;" x ";C
890 LOCATE 14,1:PRINT USING"$$##,#####";P;:PRINT" x (1+";I;")^
    = ";:PRINT USING "$$###,#####.##";FV
900 K$=INKEY$:IF K$="" THEN GOTO 900
910 IF K$="P" THEN P=P+1*SP ELSE IF K$="p" THEN P=P-1*SP:IF P<1 THEN
P=0
920 IF K$="I" THEN I=I+.01 ELSE IF K$="i" THEN I=I-.01:IF I<.01 THEN
I=.01
930 IF K$="C" THEN C=C+1*SP ELSE IF K$="c" THEN C=C-1*SP:IF C<1 THEN
C=1
940 IF K$="Y" THEN Y=Y+1 ELSE IF K$="y" THEN Y=Y-1:IF Y<1 THEN Y=1
950 IF K$="F" THEN SP=SP*10 ELSE IF K$="f" THEN SP=SP/10
960 IF K$=CHR$(13) THEN RUN 100
970 GOTO 820
980 CLS
990 PRINT"              Permutation Formula"
1000 PRINT
1010 PRINT"How many ways can (R) items within a"
1020 PRINT"larger group (N) be arranged?"
1030 PRINT"P is the answer."
1040 PRINT
1050 PRINT"                N!
1060 PRINT"    ---------------------  = P
1070 PRINT"           ( N - R )!
```

```
1080 PRINT
1090 N=7
1100 R=5
1110 IF N<2 THEN N=2      : ' MUST HAVE 2 TO ARRANGE ANYTHING
1120 IF N>32 THEN N=32    : ' CAUSES OVERFLOW IF MORE
1130 IF R<1 THEN R=1      : ' MUST HAVE AT LEAST ONE ITEM
1140 IF R=>N-1 THEN R=N-1: ' CAN'T ARRANGE MORE THAN YOU HAVE
1150 T=N-R:N1=N
1160 FOR I=1 TO N1-1      : ' FACTORIAL TAKES PLACE HERE
1170 N1=N1*I
1180 NEXT I
1190 IF T<1 THEN T=1:GOTO 1230
1200 FOR I=1 TO T-1       : ' FACTORIAL TAKES PLACE HERE TOO
1210 T=T*I
1220 NEXT I
1230 P=N1/T
1240 LOCATE 11,1,0:PRINT"       ";N;"!       ";N1
1250 LOCATE 12,1:PRINT" ------------ = -------------- =";P
1260 LOCATE 13,1:PRINT"  (";N;"-";R;")!     ";T
1270 K$=INKEY$:IF K$="" THEN GOTO 1270
1280 IF K$="N" THEN N=N+1 ELSE IF K$="n" THEN N=N-1
1290 IF K$="R" THEN R=R+1 ELSE IF K$="r" THEN R=R-1
1300 IF K$=CHR$(13) THEN RUN 100
1310 GOTO 1110
```

Check.Bas from page 5

the original Card.bas for other things and save the new Check.bas to organize your checks.

There is yet another way to do it. We have put CHECK.MRG on the CodeWorks download and you can get it that way if you wish. Then go through the above procedure. If you don't want to mess around with any of the above, we also have a merged Check.bas on the download, ready to go without merging anything. Take your pick. You will also still find Card.bas (under the Issue 2 menu) there too.

Check.bas works just like Card.bas does, and all the discussion in Issue 2 applies here too. The number of fields has been reduced to four, however, and option 5 of the menu now says to print check lists or a summary of checks instead of simply printing reports. Most of the changes needed to be added to option 5.

It is quite important to think about the categories you will assign to your checks. For example, make one "Doctors" and another "House payment", and so forth. Be sure you use the same spelling every time you enter the same category. Try to get general categories that will conform to what you or your accountant will need in preparing your income tax forms.

After all your checks have been entered, sort the file on the category field and save it. Then run the program again and print your reports. Because you have sorted by category, all similar items will be adjacent. The program will check to see if the current item is the same as the previous one and if so, will add to a subtotal and keep a running total of all the subtotals. It will then print that category only once with the total of all items in it on your printer. After printing all the different categories, it will print a grand total.

The other report is a simple list of all the records you have entered. You can sort this one any way you like and print it out. Now if you or your accountant has any question on any check, this list will identify (by the check number) where it came from in your original checkbook.

That's all there is to it. You even get a grand total of all the checks you wrote last year — ouch! ●

Payroll.Bas

A Small Business Payroll Program

Staff Project. The diversity of rules, regulations and tax laws in the various states make a generalized payroll program a challenge. This program is a simple starting point from which you can customize one as you wish. It uses sequential files throughout (see related article in this issue on those files) and the structure of the program may lend itself to uses other than payrolls.

In the 1800's the stagecoach would race through the foothills with a strongbox strapped to its top. At least two tough looking hombres with rifles would ride "shotgun", keeping a sharp eye out for desperados, injuns and slickers. After arriving at the mining camp, the strongbox would be opened and all the workers would be paid in gold coin. They would then head for the nearest town, presumably to squander their gold on whiskey, women and poker - not necessarily in that order. Payrolls had their own problems then.

In 1913, immediately after the 16th amendment became effective, congress enacted a personal income tax, with a normal rate of 1% on incomes in excess of $3,000 for a single person and $4,000 for a married person, and surtax rates on taxable incomes in excess of $20,000 ranged from 1% to 6% The corporate rate then was 1%.

With minor exceptions, all employers were required to withhold tax on a current basis from employees' pay checks and transmit the sums to the government. By 1960, this country obtained 82% of its total revenue through income taxes.

The national system of social security established by the Social Security Act of 1935 was part of President Franklin D. Roosevelt's "New Deal." This resulted in another contribution from the employee, this time matched by the employer. Since the Internal Revenue Service had already been set up to administer the income tax program, the social security (FICA) deduction was conveniently included with the IRS required deposit and included on IRS form 941.

If it had all stopped there, writing a payroll program for a computer today would be a snap. But it didn't stop there. State agencies, labor unions and others all get their "cut" of a paycheck. Unfortunately (or fortunately, however you want to look at it) the state and local deductions are not as uniform as the federal deductions. Some states, Washington state is one, do not have a personal income tax. Even in states that do, the amount differs and the income against which the tax is levied differs. In addition, most income taxes, both state and federal, are levied on a graduated basis.

Many people who are not employers or directly connected with the collection of these taxes assume that the government gets paid on April 15th of each year. Nothing could be farther from the truth. An employer withholds income taxes and FICA amounts, and if the amount is less than minimal, deposits them in a designated bank quarterly. If the amount is more than minimal the deposit must be made monthly and if the amount amounts to anything, the deposits must be made on what the IRS calls an "eighth monthly basis." What that means in practical terms is that the deposit must be made eight times per month or twice per week.

What follows is a description of a payroll program that will handle up to ten employees. It is designed for small businesses, and has been used in various previous incarnations over a period of about eight years. Because of that, it gets right down to what you really need to fill out the required forms and keep acceptable records of employees. It provides each employee with a pay stub for each pay period, a list of employees and most important, a report of accumulated quarterly earnings and deductions needed to make proper deposits and complete the required quarterly forms. It also provides information for W-2 Forms at the end of the year and allows both quarterly and year-end clearing.

The Overall View

The program establishes a sequential disk file for each employee. Every time you do anything with an employee, that file is loaded from disk,

operated on and then written back out to the disk. One other sequential file is established, containing only the *file names* of all the active employee files. When you do the payroll, or clearing, it is this file which automatically points to and gets the proper employee files. It is also this file which will tell you who your employees are when you type in a wrong name.

The program is menu oriented, with options to do the payroll, edit a pay record, print reports, add or delete employees, clear records and end the session. Let's start at the beginning of the program and explain what's happening top to bottom.

Program details

In line 170 we dimension two arrays; E() is the number of elements in each employee record and E$() is an array of the file names of up to 10 employees.

Line 180 asks for a date which will be posted on the pay stubs and your reports. You can enter it any way you like. If your computer keeps the date internally (it's usually called DATE$), simply change line 180 to: 180 D$=DATE$. That way, you won't even be asked for the date, it will pick it up from the system. Lines 200 through 340 are remark lines that identify some important variables. In line 350 we read in the employee file name file, and then immediately write it out again. This happens many times in this program. Whenever we add or delete an employee we update this file right away so that during one session where you might add one employee and delete another the file is always exactly current.

Lines 370 through 400 establish some of the variables that are apt to change with the dictates of the IRS or the state you live in. Putting them up front makes them easy to find and change, and making variables of them makes the change global. Some of these have a nasty habit of changing every year. In line 400 you will find an exclamation point after the number 37800. It is not a sign of disgust, only the computer telling us that the number is greater than 32767. Don't type the exclamation point in; let the computer put it there by itself it it really wants it. Also, if you download this program on the CodeWorks download, you may or may not get a syntax error on that line. If you do, simply remove the exclamation point. (Our MS-DOS computers didn't care about it, but our Tandy Model III choked on it.)

In line 410 we define FNI(X) as a defined function. This is a rounding function, so we don't end up with paychecks with $125.100234 or something equally ridiculous.

Your company name and address is hard coded into lines 420 through 440. It will appear at the top center of your reports, and as a one line entry across the top of each pay stub.

The menu comes next and is straightforward. INKEY$ is used to limit the choices to only those available.

Do the Payroll Module

The first module we encounter is the payroll module. It goes from line 620 through line 1040, and takes one employee at a time (per the pay names file), calculates the current pay period, updates the quarterly and year-to-date totals, prints the pay stub and writes the updated record back to the disk. If you have five or less employees, the pay stubs will fit on one standard 8 x 11 sheet of paper; if you have more than five then two sheets will be required. Don't worry about it if you have a single sheet printer. The program will stop at each employee in any case to ask for hours worked and vacation, which will give you the opportunity to change the paper if necessary.

Entering zero for hours worked will skip over an employee. This was added so that if an employee took a week off without pay, for example, you can exclude him or her from the current payroll.

Let's go to line 720 and see what happens. N1 is always the number of names in the pay names file and because of what we said earlier, should always be the current number of active employees. Let's say that the first name in the pay names file is John. There will then be a file on disk with the filename John, and in that file his full name might be JOHN Q. PUBLIC. In line 730 we then equate E$ to E$(1). E$(1) equals JOHN and now E$ equals JOHN. We then go to the "open employee file and read subroutine", at line 2830. While we are there, we will open the sequential file for input and get John's full name, E1$ and his social security number, S$. We then go into a little loop and read in 19 items (from 0 through 18) of information concerning John. After that, we close the file and return. We now have John's record in memory.

We then ask how many hours John worked and if vacation was taken and then calculate his current pay. This all happens from 740 through 840. While in those lines, we also update his quarterly and year-to-date totals. In line 850 we go through a one line loop and round all the calculated values, using the defined function we set earlier in the program.

Line 860 prints your company name, address, city, state and zip in one line across the top of the pay stub. The following lines through 980 print

the remainder of the pay stub. We are done with John now, so in line 1000 we go to the subroutine that writes John's updated record back to the disk. Upon returning, we check to see if this was the fifth pay stub we have printed. If it was, we tell the printer to form feed to the next page. We then go on to get the next employee record from the pay names array E$(2).

After we have cycled through all the valid pay names, we end up at line 1030, which forces a form feed to the printer, and then we return to the main menu - the payroll is done.

You can now slice up the page of pay stubs and insert them into the pay envelope with the paycheck, or if you like, make a photo copy of it first for your records.

Edit or Review a Pay Record

This module allows you to examine a pay record and if necessary, make changes to it. In line 1080 we enter the file name of the person whose record we wish to see. The very next thing that happens is that we go to a subroutine at 3090 to see if this is a valid name. That subroutine compares the name we just typed in with those in the pay names file (remember that they were read into memory and into array E$( ) early on in the program?) and if an exact comparison is not found, it tells you so and gives you a list of the valid pay names and a chance to try again. This prevents mix ups and also a lot of "file not found" errors and "bad file name" errors.

Here would also be a good place to mention that you cannot have the same file name for more than one employee. If you have two people named Bob, for example, call one BOB and the other BOB1, or BOB plus the first letter of his last name.

Having found a valid employee, we then go (in line 1100) to the read employee record subroutine. The information is then displayed on the screen in line 1120 through 1240.

Lines 1250 through 1350 need some explanation. First off, if we choose to correct nothing, we enter zero in line 1250. The next line, 1260, will not operate because X2$ *is* now a null string. So in line 1270 we go through a loop to clear all the variables of the record we just looked at from memory. Yes, they are still there, and they are on the disk too, but since we have made no changes, there is no point in writing them back. The reason we have to clear this record from memory is that if you first look at one employee record and then go on to create a new pay record for someone else, the new record would pick up all the data from the record just viewed. Now, after clearing the data from memory, we go back to the main menu.

Now let's assume we want to make a change to the record. In line 1250 we enter the item number. Since XX is now equal to something other than zero, line 1260 will be skipped again, and so will line 1270. In line 1280 we check to see if the number is in the allowable range of numbers and if not, go back and ask for it again. Now in line 1290 we enter the correct information as X$. We use a string here because two of the items in each pay record are strings, the name and social security number. They are items 1 and 2. In lines 1300 and 1310, if XX was 1 or 2, we exchange X$ with the name or social security number. But the remainder of the items in the pay record are all integers, and they start with zero *after* the name and social security number. So in line 1320, if XX was greater than 2, we simply subtract 3 from XX and make that array element in the pay record equal to the *value* of X$.

Lines 1330, 1340 and 1260 allow us to make multiple changes to the record before writing it back to disk or going back to the main menu. If the answer in line 1330 is anything but Y or y, we write the record to disk in line 1350, then clear X2$ and go to line 1110 where the updated record is re-displayed. From there you can choose zero to get back to the main menu. If we do want to make more changes, line 1340 sends us to line 1250, where we select the number and make the change. If we had said we want to make a change and then at line 1250 select zero, line 1260 finally comes into play and forces us to change something.

Print Payroll Reports

The reports module is between lines 1360 and 1900. Two different reports are provided. The second is simply a list of your employees with their full names, social security numbers and those items which are fixed, like withholding percentage and medical deductions. The first report is the one that will be used more often, as it keeps a running total of amounts paid and withheld. It also calculates your accumulated income tax and FICA liability for the current quarter.

As we did earlier, the pay names array is read to find who the current employees are. Then each pay record is read in and the appropriate values are summed in variables T1 through T8. Tax liability is figured in line 1720, and consists of two times the withheld FICA plus the income tax withheld.

Depending on which state you live in, it also provides most of the information needed in filling out quarterly reports for state employment

security and for whatever form of state workman's compensation you have. There is certainly nothing exotic about the code in this section. It simply totals data and prints it out in report form.

Add or Delete Employees

The add employee module runs from line 1900 through 2220. The sub-menu in this section allows you to add, delete or return to the main menu. The usual error trapping at the menu is provided for. Making a new employee record consists of nothing more than entering six pieces of information. See lines 2010 through 2100. The rest of that new employee record will contain zeros until the first time he or she is included in a payroll, when the zeros will change to some other value.

After the basic information for the new employee is entered, we ask for a first (or file) name for that employee. In lines 2130 through 2150, we check the name just entered against those already in the pay names array. If the name is already being used, we get the chance to give another, different, name. If it is unique, we go on to line 2170, which sends us to the "open file and write" subroutine at 2750. That will place the new employee record on disk. But now we need to tell our little index file, the pay names file, that we have a file name to add to it. The three lines of code that do this are at 2180 through 2200. What they do is read the pay names file and look for the first blank space, then put our new name into that blank.

The other part of line 2190, where E$(I)="ONE" is for first time initialization of the program, when there are no names at all in the pay names file. If we try to run this program with no files initialized, we will get a "file not found" error. So the first time we run it, (and this is the only time we need to do this), we load the program, then type, in command mode, this statement: E$(1)="ONE":GOTO 2920 and ENTER. The program will then create the pay names file on disk. It will contain the word "ONE", and there will be a "return without gosub" error on your screen. Ignore the error and simply run the program. From here on in, you will never need to worry about it again.

Oh yes, at this point you should immediately go to the add employee menu and put at least one employee into the system. Because of line 2190, the "ONE" in the pay names file will then be replaced by that first employee file name. Keep this little procedure in mind. We will use it again in a slightly different way to re-activate a previously deleted employee later.

The delete section is covered from lines 2230 through 2390. It is very easy to do. We simply give the file name of the employee to delete, find it in the pay name file and change it to a null string. In line 2390, we then go to the subroutine which writes the pay names file to disk and when it finds the null string it skips over it so that the names will be contiguous in the file. It then reads the pay names file back into memory. Now that employee will no longer be included in payrolls or reports, but *his employee file is still on the disk with all its data.*

Let's suppose that Tom, Dick and Harry all work for you and that they have all been employed since January. In June, Harry decides to take off three months to go fishing in Alaska. When he goes, you carry him till the end of the quarter (end of June) by entering 0 for hours worked when you do the payroll. After you have cleared the quarter at the end of June, you can delete Harry, and don't worry about him until he comes back from Alaska in September. Then before you do the first payroll on which he will be paid, you load the program and in command mode (assuming you still have Tom and Dick), you type: E$(3)="HARRY":goto 2920 and Enter. Ignore the "return without gosub" error and run the program. Harry will be reinstated as an employee with all of his year to date information intact. Note that this is very similar to starting the program for the very first time. Only this time the E$() subscript is three, because Tom is 1 and Dick is 2. Always reinstate an employee at the end of the list of pay names, regardless of where he or she was in the line up before being deleted.

End of Quarter/Year Clearing

This module resides between lines 2400 through 2670. Once again, we read the pay names file and cycle through the pay names array to fetch the individual pay records. If the quarter is being cleared, only items 11 through 18 in each pay record are set to zero, since these are all quarterly items. If year-end clearing is selected, then items 5 through 18 are cleared, which clears the year-to-date totals as well as the current quarter. This selection is made in line 2570 and executed in line 2610.

Other than that, the clearing module is easy code to follow. To prevent clearing anything by mistake, the words (in capital letters) QUARTER or YEAR must be typed in to affect the clearing; otherwise line 2560 will send you back to the main menu. After all records for all active employees have been cleared line 2670 will return you to the main menu.

```
Magarac's Widget Company 1234 Tool Steel Road Skunk Hollow, WA 98000
THOMAS A. ARMSTRONG      123-45-6789   Pay period ending: 7 FEB 86

HOURS VacAvail  Taken    Rate       GrossPay          NetPay
  40     3        0      10.25       410              317.19

                                  -- Deductions --
                  FICA      FedTax     StateTax  Medical    WorkmnComp
Current Period --  28.91      41         16.4      5.4         1.1
Year to Date ----  57.82      82         32.8      10.8        2.2
YTD Gross $ 820
-------------------------------------------------------------------
Magarac's Widget Company 1234 Tool Steel Road Skunk Hollow, WA 98000
RICHARD C. ANDERSON      234-56-7890   Pay period ending: 7 FEB 86

HOURS VacAvail  Taken    Rate       GrossPay          NetPay
  36     1.9      1      12.65       455.4            330.08

                                  -- Deductions --
                  FICA      FedTax     StateTax  Medical    WorkmnComp
Current Period --  32.11      54.65      27.32     10.25       .99
Year to Date ----  67.78      115.37     57.68     20.5        2.09
YTD Gross $ 961.4
-------------------------------------------------------------------
Magarac's Widget Company 1234 Tool Steel Road Skunk Hollow, WA 98000
MARY BETH HUTCHINSON     345-67-8901   Pay period ending: 7 FEB 86

HOURS VacAvail  Taken    Rate       GrossPay          NetPay
  48     3.3      0      13.75       660              470.4

                                  -- Deductions --
                  FICA      FedTax     StateTax  Medical    WorkmnComp
Current Period --  46.53      79.2       52.8      9.75        1.32
Year to Date ----  85.31      145.2      96.8      19.5        2.42
YTD Gross $ 1210
-------------------------------------------------------------------
```

This is a page of pay stubs for Magarac's Widget Company after the second pay period. If you use the figures for these employees on the facing page, you can use this figure to check that you have entered the program correctly. During the first pay period each employee worked 40 hours and took no vacation.

Magarac's Widget Company
1234 Tool Steel Road
Skunk Hollow, WA 98000

Accumulated Pay Amounts Report for period ending 7 FEB 86

Hours	Gross	FICA	FedTax	WkComp	StTax	Med	NetPay
THOMAS A. ARMSTRONG			123-45-6789				
80	820	57.82	82	2.2	32.8	10.8	634.38
RICHARD C. ANDERSON			234-56-7890				
76	961.4	67.78	115.37	2.09	57.68	20.5	697.97
MARY BETH HUTCHINSON			345-67-8901				
88	1210	85.31	145.2	2.42	96.8	19.5	860.77
*** TOTALS ***							
244	2991.4	210.91	342.57	6.71	187.28	50.8	2193.12

Total 941 liability so far is $ 764.39

Above is a sample of the Amounts Paid/Withheld after the second pay period.

Magarac's Widget Company
1234 Tool Steel Road
Skunk Hollow, WA 98000

Employee List as of 7 FEB 86

Name	SS#	Rate	FedTax%	StTax%	Med Ded
THOMAS A. ARMSTRONG	123-45-6789	10.25	.1	.04	5.4
RICHARD C. ANDERSON	234-56-7890	12.65	.12	.06	10.25
MARY BETH HUTCHINSON	345-67-8901	13.75	.12	.08	9.75

This is what the Employee Information Report looks like.

End Session Module

The end session module is the shortest and easiest. It resides between lines 2680 and 2730 and consists of nothing more than a reminder to back up your files. It then simply runs into an END statement in line 2740 and gives the BASIC ready prompt.

Subroutines

There are four subroutines which are called repeatedly from the main program. The first is the "Write employee pay record" subroutine from lines 2740 through 2810. See the article on sequential files in this issue for more detail on how they work.

The next subroutine, from lines 2820 through 2900, is the "Read employee pay record" subroutine.

The next subroutine both writes and then reads back the pay names sequential file. Notice that there is no return or end at the end of the write portion; it goes directly to the file and then is read back immediately. This keeps the valid pay names as up to date as possible during operation of the program. Also note in line 2950 that when writing the pay names to disk, if a null string is found (presumably because of a deletion), the null is not written to the disk but goes to the NEXT I instead. This insures that the valid pay names are at the beginning of the pay names file with no gaps.

The Who is Real? Subroutine

The last subroutine of the program, starting at line 3080 and continuing through 3220, is used any time you ask for any employee pay name. It checks through the pay name file and looks for a match, and if it finds one, it returns to wherever you came from. In line 3100 it also checks to see if you simply pressed the ENTER key instead of entering a name. If you did press ENTER or an invalid pay name, line 3130 tells you so and then gets the valid names from the pay names array and prints them for you to see. It then asks which one you want, and will continue to do so until you get it right. This sounds dictatorial, but remember that we are messing with someone's pay here, and it should be done right.

Suggested additions

As mentioned at the very beginning of this article, making a program that applies to all states and situations would be formidable. Once you study this program you should be able to modify it sufficiently to take care of your special cases. One of the things that could be added would be a check writer. Here again, there are so many conventions and spacings on available check stock that you almost need to do your own. In the "do the payroll module", you could easily print the pay stub above a tractor fed check, then jump to a subroutine at the end of this program that would write the check itself.

Another feature that could be added is the computation of overtime. You could simply convert overtime hours to straight hours, but be careful. If your state computes anything on hours (ours does) then you would be off there.

W-2 form information is all contained on the last pay stub of the year. If you do not routinely make copies of your pay stubs, this is one you will want to have a copy of. ■

```
100 REM ** PAY.BAS ** FOR CODEWORKS MAGAZINE
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 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.
170 DIM E(18),E$(11) : ' E$() sets the max number of employees plus 1
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$;" worked this period";:INPUT HO:IF HO=0 THEN G
OTO 1020
750   PRINT"Did ";E$;" 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)+C
M
830   E(17)=E(17)+HO:E(11)=E(11)+GP:E(12)=E(12)+CF:E(13)=E(13)+CT:E(1
4)=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(N
P):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 TAB(2);HO;TAB(8);E(4);TAB(17);HT;TAB(25);E(0);TAB(35);GP
;TAB(50);NP
920   LPRINT" "
930   LPRINT TAB(32)" -- Deductions --"
940   LPRINT TAB(20);"FICA";TAB(30);"FedTax";TAB(40);"StateTax";TAB(5
0);"Medical";TAB(60);"WorkmnComp"
950   LPRINT"Current Period --";TAB(20);CF;TAB(30);CT;TAB(40);CS;TAB(
50);CM;TAB(60);CL
960   LPRINT"Year to Date ----";TAB(20);E(7);TAB(30);E(9);TAB(40);E(6
);TAB(50);E(8);TAB(60);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 Ø for none ";XX
1260 IF XX=Ø AND X2$<>"" THEN PRINT"You chose to change something, wh
ich number";:INPUT XX:IF XX=Ø THEN GOTO 125Ø ELSE GOTO 128Ø
1270 IF XX=Ø THEN FOR I=Ø TO 18:E(I)=Ø:NEXT I:GOTO 45Ø
1280 IF XX<1 OR XX>21 THEN GOTO 125Ø
1290 LINE INPUT"Enter the correct information ";X$
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 125Ø
1350 GOSUB 275Ø:X2$="":GOTO 111Ø
1360 END:REM ----------------------> Print Payroll Report module **
1370 CLS:PRINT TAB(1Ø);"   PAYROLL REPORTS"
1380 PRINT
1390 PRINT"Get your printer ready"
1400 PRINT
1410 PRINT"1 - Report of Amounts Paid/Withheld (IRS 94Ø info)"
1420 PRINT"2 - Employee Information Report."
1430 PRINT"3 - To return to main menu."
1440 PRINT
1450 PRINT"Your Choice";
1460 X$=INKEY$:IF X$="" THEN GOTO 146Ø
1470 X=VAL(X$):IF X<1 OR X>3 THEN GOTO 146Ø
1480 LPRINT TAB(2Ø);C1$
1490 LPRINT TAB(2Ø);C2$
1500 LPRINT TAB(2Ø);C3$
1510 LPRINT" "
1520 ON X GOTO 153Ø,175Ø,45Ø
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(1Ø);"Gross";TAB(2Ø);"FICA";TAB(3Ø);"FedTax";TA
B(4Ø);"WkComp";TAB(47);"StTax";TAB(56);"Med";TAB(65);"NetPay"
1610 LPRINT" "
1620 FOR I= 1 TO N1
1630   E$=E$(I):GOSUB 283Ø
1640   LPRINT E1$,S$
1650   LPRINT E(17);TAB(1Ø);E(11);TAB(2Ø);E(12);TAB(3Ø);E(13);TAB(4Ø)
;E(14);TAB(47);E(15);TAB(56);E(18);TAB(65);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 T1;TAB(1Ø);T2;TAB(2Ø);T3;TAB(3Ø);T4;TAB(4Ø);T5;TAB(47);T6
;TAB(56);T7;TAB(65);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);T
AB(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 10
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$="0" THEN GOTO 1920
2350 GOSUB 3090
2360 FOR I=1 TO 10
2370   IF E$(I)=E$ THEN E$(I)=""
2380 NEXT I
2390 GOSUB 2920:GOTO 350
2400 END:REM ---------------> End of Quarter/Year Clearing module **
2410 CLS:PRINT TAB(20);" QUARTER / YEAR END CLEAR "
2420 PRINT
2430 PRINT"Be sure you have printed your payroll reports for the"
2440 PRINT"quarter before clearing. Clearing the quarter will remove"
2450 PRINT"all quarterly data for ALL employees. Clearing the year"
2460 PRINT"will clear everything except basic employee data for"
2470 PRINT"ALL employees. Use Edit/Review option to verify clear."
2480 PRINT
2490 PRINT"At the end of the year, clearing the year will clear the"
2500 PRINT"last quarter as well. >> Print your reports first! <<"
2510 PRINT"To prevent inadvertent clearing, you must type in the"
2520 PRINT"word QUARTER or YEAR, otherwise, you will be sent"
2530 PRINT"back to the main menu.
2540 PRINT
2550 PRINT"CLEAR what: ";:INPUT X$
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$+",",S$+",",
```

```
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$,S$
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 10
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 10
3030   IF EOF(1) THEN GOTO 3060
3040   INPUT#1,E$(I)
3050 NEXT I
3060 CLOSE 1
3070 RETURN
3080 REM ------------------------------> Who is Real? Subroutine **
3090 FOR I=1 TO 10
3100   IF E$(I)=E$ AND 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 END:'of program.
```

This program, as well as the others in this issue, are available on the download system under menu item "Issue 4."

Puzzler #3

Making a Card Deck

It's almost been like "Name that tune" around here. We have been working on a poker playing program, and every time someone gets a bright idea about how to make up a deck of cards on the computer, someone else seems to come along and say they can do it in two less lines of code.

It started with about a dozen lines of code. Then it was reduced a few lines, and still later a few more lines. Somewhere along the way, someone mused about the fact that this would make a neat puzzler, so here it is.

Here is what we want to end up with: An array, let's call it B( ) that starts with B(1) equal to 102 and ends up at B(52) equal to 414. That is, the array will start at element one containing the number 102 and go through 114, then 202 through 214, 302 through 314 and 402 through 414. This way, the first digit of the number will designate the suit of the card and the second two digits will always be the value of the card from the deuce through the ace.

As an aside, let's say that there is no sense in carrying a card value throughout the program in string form; it gets messy very quickly. So the logical way to do it is to assign numbers to the cards and operate on all of them as integers. You only need to change the card suit and value to a string when it is printed out on the screen.

Since we obviously need to dimension B( ) at 52, we won't count that as a line of code. Given this problem, what is the least number of program statements it will take to create that B( ) array?

Send your solution to us and we will print and give credit to those who do it in the least number of program statements. ■

Download

The CodeWorks download system is always improving, sometimes from plans we have for the system and sometimes from improvements suggested by subscribers and others. Here are some changes that became active since the last issue.

Al Mashburn suggested that some users may have trouble logging into the system because they often hit RETURN a few times from habit. (Some systems use this as a way to tell what baud rate you are using.) This would throw you into the Demo menu system. We ignore these returns now and simply tell you to type in a name.

One subscriber (Richard Burwell) found that he couldn't use a six character password as we had said you could. We fixed that, so now you can. We have also made it possible to set your password from the main subscriber menu.

Tony Pepin, a local subscriber, found that some lines had been truncated in a download. We traced that to a problem in setting maximum line lengths and corrected it.

Greg Sheppard suggested that we have a terminal configuration option to set whether you want a Return, Line Feed, or both. It's on line now. He also suggested a flag to mark if you've already seen the Message of the Day during login. It's there now so after you have read the Message of the Day once, it will not bother you again until it changes.

Since some users have had trouble understanding what they could enter for terminal types, we have tried to make that clearer by printing a short description of the terminal on the terminal configuration screen and checking to make sure we understand the terminal type when you try to change it. If you enter a question mark when you have been prompted for your terminal type, you will get a readout of a terminal help file which will list all the terminal types the system can understand. (Watch out, there are *hundreds* of terminals listed.) We will spend more time in the issue on the details of terminal types.

There have also been some internal changes to tune up the system and make it faster. Thanks for your interest and support. ●

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Address correction requested

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

CodeWorks

Issue 5 May/June 1986

CONTENTS

CODEWORKS

Issue 5 May/Jun 1986

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

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

300 Baud, 8 bits, no parity, 1 stop bit

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

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

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

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

Editor's Notes

Do you realize how much computing power you have?

It wasn't that many years ago when I remember walking into the computing room of a large corporation. It was truly "twenty degrees cooler inside" and the floor was raised. The large room was teeming with cookie-cutter men, all wearing slate-gray suits, white shirts and black ties. Even their haircuts were identical.

There was row after row of identical tape drives. There were more rows of mechanical sorting machines and key punch machines. There were several line printers spewing paper. The centerpiece of this whole affair was a large circle of blue-gray cabinets. At the center of this circle there was a four cubic foot "brain" behind Plexiglas. Behind the glass you could see little lights winking on and off in mesmerizing patterns. The brain was surrounded with red velvet ropes connected to four highly polished brass posts at each corner. There was a very neatly lettered sign on it that proclaimed it to be a 128K Core Memory. Tours of the installation were commonplace. It was a focal point of corporate pride.

Outside the "machine room" was a vast area filled with partitioned cubicles. Each cubicle contained one person and stacks of computer printout. Occasionally you could see an empty cubicle, and further along one with two people, usually involved in some apparently meaningful discussion concerning programming the computer.

These were the programmers who fed the beast. The language then was FORTRAN. At Dartmouth, about this time, a couple of men were playing around with what was later to become BASIC. I can't recall for certain, but I think that the computer was an IBM 7090.

Several years later I again had the opportunity to see the same computer room. It now contained two IBM 360/65's driven by an IBM 360/50. The inner sanctum no longer had the holy of holies look. The computer operators now wore blue jeans and plaid wool shirts.

They queued-up tapes and hung paper or changed disk pacs. The sea of programmers in the outer room had shrunk. They had even moved one wall to take away some of the vacated space. They were now talking about memory with one-half megabyte. Bill Gates was still attending elementary school.

These days most of us have more computing power sitting right in front of us than the 128K monster mentioned earlier. We also have it at less than one-hundredth the cost. The sea of programmers has shrunk even further - to one person - you. But let's not forget that those early programmers laid the groundwork for most of what we do today, and they had to do it with memory and availability constraints.

It hasn't been that many years since the sum total of anything electronic was all radio. A couple of weeks ago, I had the opportunity to look at my own heart in action on a video screen. The valves were opening and closing, the walls were expanding and contracting and there was no blood obscuring the picture nor ribs getting in the way. An elaborate sonic transducer was placed on my chest to get the picture. It also picked up the heart beat sound and amplified it. A medical technician did it, and it was recorded on a video tape recorder. A doctor who knew what to look for saw it all later, at his convenience, on a television set. The computer that did it was not much more than we already have sitting on our desks. The recorder was an off-the-shelf VCR.

Yes, computing power has come a long way in a very short time. - Irv

Forum

An Open Forum for Questions & Comments

In the Mar/Apr 86 issue you had a question from Art Phillips about closing a For...Next loop. I found (the following) in a newsletter called *Northern Bytes* in the form of an article entitled *For/Next Termination* by Ray Greet taken from the Adelaide (South Australia) *Micro-User News*. The following is the content of the article:

"About twelve months ago I mentioned to some members that exiting from For/Next loops prematurely, without terminating the loop, was poor programming practice. My justification for making the statement was based on information obtained from the book *Microsoft BASIC Decoded*. This stated that when a loop construct is generated, a 16-byte "frame" is deposited on the system stack. From this I concluded that if a loop is not properly terminated then the construct frame would remain on the stack and so consume memory unnecessarily. The frame is a structure containing details of loop parameters necessary for control of the loop process (this frame, incidentally, expands to 20 bytes if a single precision index variable is used.)

A recent listing passed to me which contains examples of unterminated loops prompted me to research the matter further and report so as to clarify the situation as it would seem the processes are not fully understood.

Some tests have shown that my original assumption was mostly correct; the frame remains on the stack except if a RETURN statement, when used as part of the subroutine, is the abortion device.

Any frame that is left on the stack will remain there until such time as a subsequent invocation occurs using the same reference variable. The significance of this is: If a subsequent allocation makes reference to a variable which was used by an earlier unterminated stack frame, that stack frame will be re-initialized. Any such re-initialization recovers all stack space that may be pending but used after any such abandoned frame. This means, of course, that any other pending For/Next frames will be lost, thus creating a source for NEXT without FOR errors. However, if a GOSUB is in control, the frame search function will only search the stack as far back as the GOSUB control frame, so pending loops invoked prior to a GOSUB are protected.

It seems obvious that to avoid any possible confusion, good programming style should be adopted. Terminate all loop constructs. Some minor examples:

```
The wrong way:
10 FOR I=1 TO 100
20 READ C
30 IF C=0 THEN 60
40 ....
50 NEXT

The right way (1)
10 FOR I=1 TO 100
20 READ C
30 IF C=0 THEN I=100:GOTO 50
40 .....
50 NEXT

The right way (2)
10 FOR I=1 TO 100
20 READ C
30 IF C=0 THEN I=100:
      NEXT:GOTO 100
40 .....
50 NEXT
100 ....

The right way (3)
10 GOSUB 30
20 STOP
30 FOR I=1 TO 100
40 READ C
50 IF C=0 THEN RETURN
60 ...
70 NEXT
80 RETURN
```

The first right way example would be used when code continuation is required, the second if further branching is necessary. The abortive method used in example three is quite acceptable as the RETURN statement will automatically purge the abandoned For/Next frame from the stack. The method by which the interpreter can clobber loop conditions as described earlier is a design flaw. It would be more logical to flag as an error any attempt to open a loop with a variable already assigned to a loop. Whether this has been changed in later versions is unknown to me."

I hope this is of some benefit to you in trying to explain For...Next loops. Perhaps you should write an article about them. Your magazine has had some really useful articles in it to date. Keep up the good work.

John Bielot
Moodus, CT

Thank you. This is the first really good reason we have heard of for properly closing loops.

We received your sample edition a few days ago and have keyed in several (programs). I find those we have entered to be error free - something that can't be said for many printed BASIC programs I have labored through in the past. This letter was prepared using MAKER.BAS. We are very pleased also with CARD.BAS, but have added one line to prevent what could cause an inadvertent deletion. The situation arises after displaying a searched-for record on Item 4, and it is *not* the desired record. If the operator does not re-enter the 1-Continue looking option, variable X

still contains a 4 and simply pressing ENTER will delete the current record. To prevent this, add line 1325 X=1, thus defaulting to the continue routine. I have also added a small optional routine to slow down the Quick Scan routine. We are looking forward to receiving this year's editions.

W. G. King
Orland Park, IL

Nice fix for CARD.BAS. It would delete the record as you said. But how did you write the letter using MAKER.BAS? Does it have some capability we didn't know about?

...Since you are probably going to get a few letters, let me be the first to point out that, in the Expand Space Compression Codes section of PRECOMP.BAS (Issue 4, page 10), line 1060 should have read:1060 IF ASC(MID$(A$,TC,1))>191 THEN A$=LEFT$(A$,TC-1) + STRING$ (ASC(MID$(A$,TC,1))-192," ")+ MID$(A$,TC+1)
One other slight note, which your industrious readers will no doubt catch onto: the subroutine doesn't always expand *every* expansion character to a space. The For...Next construct (in most computers) only evaluates the expression LEN(A$) at entry into the loop and, therefore, scanning will stop just as soon as the *original* length of A$ is reached. For the application (i.e., PRECOMP.BAS), this is quite acceptable, and the computers that use space expansion usually have no problem having them in BASIC programs.

J. Melvin Jones
Warwick, NY

Note that this change only applies to the code inside the box on page 10 of Issue 4.

The other night I downloaded WOOD.BAS and made some changes to suit myself and the TRS-80 Model III... I have run into a problem with it. When you input exact sizes with no kerf, as when cutting paper stock, or when a plywood cut matches a leftover exactly, the program comes up with an Illegal function in line 1430 and crashes. It will also accept minus dimensions for leftover pieces. I would expect that others would run into the same problem and that a fix would be published in the next issue. Thanks again for a real magazine.

Frank M. Smiley
Newark, DE

We ported the program over to a Model III and found the exact problem you describe. Here is the fix. Add line 1425 IF NS=<1 THEN 1470. Somehow, on the Model III, NS gets to -1 when there are exact fits. Although we don't know for sure, this may be another of those little changes between Microsoft Versions 5.1 (prior and after). The other computers we have all end up with NS equal to 1 in that case. The minus dimension problem goes away with this fix also. The program currently on the download has been fixed to reflect this change. Thanks for bringing it to our attention.

Last week I wrote to you praising the program WOOD.BAS. I have not changed my mind about the program having now entered the code and run it. It is great! But I found one possible correction and then I have added some code to make it possible to save the leftovers and re-use them in another problem. The code is enclosed. *(See Figure 1).* The possible correction is in line 370. I suggest the GOTO be to line 360 instead of 340. If the length is ok, why re-do it just to get the width correct? The code to add saving and getting is as follows: Line 345 asks if you wish to load leftover inventory. If yes, GOTO 2900. Lines 1845-50 ask if you wish to save leftovers. If yes, GOTO 2830. Lines 2810-90 save leftovers larger than 2 inches on a side. Lines 2900-90 load the leftovers. The file name for the leftovers can be named using the type of wood, etc. Line 3000 is for convenience in saving the program after modification and test, always to the same file, just by typing in GOTO 3000. I trust you approve of the suggested modifications.

Larry Abbott
Wyomissing, PA

Yes, we do. We thought about it when we wrote the program, but left that out because the length of the program was already getting out of hand (not to mention the article describing it.)

I am a charter subscriber to CodeWorks and find the magazine refreshing and informative. I have over 20 years of business applications data processing experience as a programmer and system analyst but still find helpful hints in your articles. I did feel, however, that I should write to mention a disappointment I felt when reading your Mar/Apr 86 issue. I refer to the article regarding PAYROLL.BAS on page 26 and the accompanying samples of the reports produced for the Magarac Widget Company. If you notice, dollar amounts that have a zero ones column in the cents do not print the zero. Also, some of the decimal points in the columns do not line up. This can be corrected simply by using editing when printing. This is basic printing methods and I feel it should be practiced especially when you are printing a business application program. No company would put up with report formats like you've shown. I would guess that many junior data processing people read your magazine. Should you not teach them correct methods in producing output? Shame on you! I will continue to subscribe to your magazine and read it thoroughly. Keep up the good work.

Peter E. Huckel
Riverside, CT

You are right. Apparently something else had our attention while we were preparing that program for publication.

Have enjoyed the magazine since the premier issue, and read it cover to cover. I am a poor typist and short on patience so I hooked my MOD IV

to a modem and called the download the other night to get Card/Bas from issue 2 and avoid the hassle of being a typist. It downloaded fine and when I went to list the program so that it could be edited for TRSDOS 6.2 BASIC, I found it was password protected and I could find no way to get it out.

I am using a 2 disk Model IV with the terminal function of DeskMate, and can't imagine how a password got involved. I tried the download password, but with no avail. Normally, I'm not into passwords anyway so I'm not all that familiar with their use. Any suggestions that you may have will be greatly appreciated.

James R. MacMurray
Wellsville, NY

In Tandy language a file may be password protected by giving the file name an additional password extension like this: card/doc.psw The .psw then becomes a password. Most other computers use the .bas the same way that Tandy uses /bas. From the directory dump you sent along, I can see that Card/Doc is file protected. (It has a P under the Attribute column.) Here is what you may try to do: Get into BASIC, then load the file like this:

LOAD "CARD/DOC.BAS"

When you can list the program, be sure to change the file extension in two places near the beginning of the program to Card/Dat from Card.Dat, see issue 2 listing. Also see Programming Notes on page 37 of issue 3, where we discussed this same problem.

I have received your special sampler issue. I admit that I am not much of a programmer and in all probability the programs in your issues will not run on my Apple IIe unless modifications are made. For example, I have entered the Calendar.Bas program and find commands that are not in Applesoft BASIC.

1. Sometimes you LPRINT and later PRINT.
2. Command: PRINT USING ...
3. Command: ELSE
4. Command: LINE INPUT

I don't understand how I can use your magazine unless I can find information on how to modify or convert the commands that my Apple IIe does not use.

John P. Overton
Bartlesville, OK

We are aware of the difference between Applesoft and most of the other BASIC's. In fact, we purposely try not to solicit subscriptions from Apple owners because of it (unless they are using the Z-80 SoftCard.) There are several good books on the necessary conversions, notably those by Dr. David Lien, of Compusoft Publishing. If enough interest is shown, we have no problem in exploring these differences.

Thank you again for your input and interest.

Irv

Figure 1

```
435 INPUT"DO YOU WISH TO LOAD LEFTOVER INVENTORY? (Y/N)";A$:IF A$="Y"
 OR A$="y" THEN GOSUB 2900:PRINT
1845 INPUT"DO YOU WISH TO SAVE THE LEFTOVER INVENTORY? (Y/N)";A$
1846 IF A$="Y" OR A$="y" THEN GOTO 2830 ELSE RUN 100
1850 END
2810 INPUT"DO YOU WISH TO SAVE THE LEFTOVER INVENTORY? (Y/N)";A$
2820 IF A$="Y" OR A$="y" THEN GOTO 2830 ELSE RUN 100
2830 LINE INPUT"ENTER WOOD TYPE AS FILE NAME (8 CHARS)";A$
2840 FS$=A$+"/INV:1":OPEN "O",1,FS$
2850 FOR J=1 TO NS
2855 IF S(J,1)<2 OR S(J,2)<2 THEN GOTO 2870
2860 PRINT #1,S(J,1);S(J,2)
2870 NEXT J
2880 PRINT"ALL DONE...."
2890 CLOSE:END
2899 REM *** SUBROUTINE TO LOAD LEFTOVER INVENTORY ***
2900 LINE INPUT"ENTER WOOD TYPE FOR FILE NAME (8 CHARS)";A$
2905 U=0
2910 FS$=A$+"/INV:1":OPEN "I",1,FS$
2920 IF EOF(1) GOTO 2980
2930 U=U+1
2940 INPUT #1,S(U,1),S(U,2)
2950 GOTO 2920
2980 NS=U:CLOSE:RETURN
2990 END
3000 SAVE"WOOD/BAS:1":PRINT"WOOD.BAS SAVED ON 1":STOP
```

Busmod.Bas

The construction of a modeling program

Staff Project. This modeling program uses a puzzler from an earlier issue. It seemed like a perfect place to use it. Although our story has a "they all lived happily everafter" ring to it, you can make it as realistic as you want. See Programming Notes on page 16 for further ideas about this program.

Many people are surprised to learn that most of the decisions they make involve forecasting. Most forecasting required for decision making is handled judgmentally in an intuitive fashion. Subjective judgments are clearly not as accurate and effective as systematic approaches. When several interacting factors need to be considered the problem of seeing clearly is further compounded.

Models of real world situations can be constructed which describe reality to various degrees. Some problems can be put into a box and tied with a neat ribbon. Others defy being so contained and explained. If you own N shares of a given stock and it goes up you have certainly gained, if it goes down you lose. That is easy enough, but predicting the price of gold in the year 2010 is another matter entirely.

The "adjustable wrench" (if you will pardon the analogy) in mathematical models in the recent past is the 'Calc program: VisiCalc, Multiplan, Lotus, Symphony and Framework, just to name a few. These are all spreadsheet programs which permit you to look through a little window to view a portion of a larger area. The "one size fits all" approach of these programs makes them very adaptable.

Our approach in this article is to show that you can build a very specific program to solve a defined modeling problem. The rationale being that if you are modeling something close to your financial well-being, you will want to know the process intimately, understand it completely, and be able to make whatever adjustments to it with confidence. Continuing with the earlier analogy, we will attempt to apply a box-end wrench to the problem.

The program with this article allows you to start with any values whatever, even if they are wrong. Then, through experience and evaluation of single parameters, the process is refined until a workable model is obtained. The program works towards the optimization of bottom line profit.

A second feature of the program is an evaluation of the sensitivity of the data. In this mode, each of the elements of the formula is varied 10 percent while holding the remaining elements fixed. The percentage change to the bottom line is then calculated to find the effect of that 10 percent change. This tells the user of the program which items will need the most attention as well as how critical (relatively) each is to the whole scheme of things.

The applicable method to follow is to start with the obvious. In our case, it is that profit equals income less expenses. Then we break income down into all the factors that have an effect on it. We do the same for expenses. The amount of detail you go into is entirely up to you. If you want to account for pencils and paper clips you can, though in practical terms you wouldn't (unless you happen to be a retailer or wholesaler of those items.) After all the detailed items have been broken out of the formula and identified, we put them all back together again to arrive at the original P=I-E. However, now that we have identified each item within income and expenses, we can vary any single item to see the effect on profit.

In our sample program, the relationship between most of the items is linear. In reality, however, one cannot expect sales to continue at a constant level as unit price rises. Sales/Price curves are a subject in themselves. See the sidebar to this article for other ways to generate this curve. In the program we use a simple inverse square function to define it, and it is biased slightly so that it shows no reduction of sales at about the $100.00 level.

Program design

Our hypothetical friend, Joe Magarac, has developed an interesting and unique electronic circuit. He has plans to produce this board in his garage workshop and sell it, via direct mail, to computerists all over the country. Before he embarks on this little adventure, he wants to know not only if the venture will be profitable, but also

how profitable. He needs to have some idea ahead of time since there will be considerable up-front cost should he commit to going ahead with the idea. Before investing any cash, he designs a program that will help him see as many aspects of his proposed idea as possible.

After some thought, Joe finds that some expenses will be fixed and monthly. Others will depend on how many units he produces, which in turn depends on how much direct mail promotion he does. He also finds that there will be two postage costs, one for sending the direct mail and another in sending out filled orders. Further, there are two different printing costs, one for the direct mail brochure and another for the documentation which accompanies the product. After checking with the local printer, the post office, and having read a book or two on direct mail methods and procedures, Joe finally comes up with the main items he needs to be concerned with. In a few cases, he already has some actual, hard figures on cost.

Here is the way Joe broke down the items he would need to consider:

Material cost per unit: The most pessimistic figure was used here, i.e., no quantity price breaks were considered. Should the orders later be sufficient to warrant buying in larger quantities at lower prices, it would simply add to the profitability. The material cost per unit includes having a circuit board etched and drilled and the cost of all the individual components which will populate the board. This cost was easy to calculate, and rounded out to $58.00 per unit.

Labor cost per unit: Joe's friend and neighbor, Marvin Munche, agreed to work for Joe nights and weekends, and he would assemble and test the boards for $10.00 per unit, another easy cost to calculate.

Miscellaneous costs per unit: Rather than detail all the little costs and clutter the program, Joe created this catch-all category. It includes the documentation (printing) cost, the taxes, the packaging and postage cost per unit. Joe was proud of his unit, and felt his documentation should speak well for his product. This pushed the miscellaneous costs per unit to $5.87.

Fixed monthly overhead costs: This cost was primarily for equipment leases. Joe needed electronic test equipment, two computers (his board had to do with communications), a circuit board testing unit and various items of furniture for his Widget factory. The cost of these items surprised even him at $2275.00 per month.

Direct mail postage: This was the easiest cost to fix; the U.S. Postal Service did it for him. It is twelve and one-half cents per piece, bulk rate.

Direct mail printing per piece: Joe correctly followed the advice of the direct mail manuals he had read. They said that the direct mail piece was his only and best salesman; it had to do the job or all was lost. After contacting the local printer, Joe found he could get what he wanted at 42 cents per piece. Again, this cost would go down slightly with quantity, but Joe took the worst case figure for his model.

Direct mail cost of names: After ordering catalogs from mailing list brokers, Joe found that most selected mailing lists rented from $60. to $100 per thousand names. Samples of larger lists for testing purposes were available at the same price, but the minimum sample number was 5000 names. Joe averaged the prices he found in the catalogs and arrived at a seven and one-half cent per name figure.

Number of direct mail pieces sent per month: Other costs will be a function of this number. So will the number of orders and the income. This number is also a function of how much you can afford to invest. The book said that advertising was an investment, not an expense. Joe decided that the maximum number of direct mail pieces he could afford to send per month was 10,000.

Percent return on direct mail: This is one figure that Joe found very difficult to estimate. The books said that the national average return for direct mail was approximately three percent. What they didn't point out is the variation in return percentage compared to the price of what was being offered. Granted, Joe could live with a lower percentage of return since his price per unit would be rather high, but the only way to find out was to do it. He decided, conservatively, to set this figure at one percent and hoped he was not being too optimistic.

Sale price per unit: Before Joe started this exercise, he had envisioned a unit price of around $100. Having gone this far, he already realized that he could go broke fast at that price. He would have to consider what the competition was selling a similar board for. Could he beat their price and

still make it? What would happen if dealers handled his product? Could he afford the discount they would ask for? The answers to these questions were precisely why he needed the model. He took a wild guess and set his unit sale price at $200. The model would find the optimum price after it was programmed and running.

Joe decided that since there were only ten items it would be easy to write the program using data statements. The ten items appear at the end of the program, with both the name of the item and the current value. The last data item is a dummy string and a sentinel (-1).

Program description

The opening lines from 100 through 280 set dimensions for the variables, print the heading and the menu on the screen and provide for selection of menu options. Since the data statements will be read several times during the use of the program, a single RESTORE statement at line 290 serves to restore the data pointer. The data, string and integer in A$() and E(), respectively, are read in lines 300 through 330. The data sentinel is checked for in line 320, which when reached, sends the program flow to line 340. In line 340, variable LN is set equal to the number of data items. Ignore line 350 for now, we are going through option 1 of the menu first, therefore Z is equal to 1.

Lines 360 through 390 print the data names and values on the screen in two columns. Since we are going to display two items at a time, a step of 2 is necessary in line 360. Making L equal to I plus 1 in line 370 allows the printing of two items on one line, one with subscript I and the other with subscript L. The print line is formatted in line 380, with appropriate dashes and spaces.

Our friend, Joe, must be a reader of CodeWorks magazine. Just look at what he is intending to do. Do you remember Puzzler 1 from issue 2 and can you see it coming? He is planning to pick any of the data items to vary through a single For...Next loop. The tipoff is in line 500, where he makes E(X) equal to I. Now, back in line 410 he is asking which of the items we want to vary and makes that item number equal to X. Line 420 checks to see that X is in the proper range. Lines 430 and 440 set the start and end values for the item we wish to vary. Line 460 then prints the name of the item we are varying on the screen.

Line 470 is a print line that puts the headings of all our output variables on the screen. The actual For...Next loop which calculates the values starts at line 490. It starts at the start value we entered and ends at the end value, but the step for the loop is calculated to be the difference between the end and the start divided by 10. This will result in ten solutions to the formula being printed on the screen, no matter how large or small the starting and ending values we pick.The value we selected to vary will be set equal to I in line 500, and will change with the range we selected for start and end. All other subscripted E variables within the loop will remain at the value they had when they were read from the data statements. Let's look at how the profit formula in line 580 was broken into its component parts.

In order to figure costs (and income) we need to know how many units were sold. In line 510 we find US, the units sold, by taking E(3) the direct mail pieces sent and multiplying it by E(7), the percentage return. Now, in line 520, we need to modify the number of units sold by some kind of function that reduces the number of units sold as price rises. The unit price is E(10), and in line 520 we square the unit price, divide it by 3000 and add two. We then take the integer of it to get rid of the trailing decimal places and if the number is less than 1 we make it 1, the assumption being that at least one person will buy no matter how high the price. Division by 3000 in this line is a shaping constant. Making this value 1000, for example, will make the price curve drop more steeply, making it 4000 will keep it up before it begins to drop off. (See sidebar to this article.)

In line 530, we can now calculate the gross income by multiplying the now modified units sold with the unit price. Line 540 creates a new variable, E1 (not to be confused with E(1), which calculates all the costs that vary with the number of units sold. Line 550 creates variable E2, which accounts for all the direct mail costs. Line 560 calculates the total expenses. We can add E(9) directly here because it represents fixed overhead and does not need to be calculated. We do, however, want a column in our output that shows all the variable expenses. Line 570 does that for us. In line 580 we are back to our original formula which states that profit (P) is equal to gross income (R) less all expenses (E3). Line 600 then prints the appropriate values under the headings that were already printed in line 470. This repeats ten times, each time incrementing the value we have chosen to vary, and then waits for you to press return to go back to the main menu.

Sensitivity routine

When we choose option 2 from the main menu we go directly to line 650. After clearing the screen and printing some heading information on it, we arrive at line 720, which sends us in a GOSUB to line 290. At line 290 we restore the data and then read it into the A$() and E() arrays. In line 350, Z is now really equal to 2, so we return to line 730.

Before going further, let's explain what is supposed to be happening here. Joe doesn't want to stumble over dollars looking for pennies. He wants to know which parts of his operation have the most effect on the bottom line. That way, when he gets up in the morning, he knows what the most important thing to do today is. He is going to do it by using the lines of code from 510 through 580 to calculate the bottom line, then change each item, one at a time, by 10 percent and recalculate for the bottom line again. That way, he can find the percentage change to the bottom line for a ten percent change in each item, and determine which item needs his most immediate attention. All these calculations will be done at one time and the results stuffed into two arrays, P(X) and P1(X). It all happens in just a few lines between 730 and 790.

Line 730 sets up the loop to read from 1 to the total number of data items. This is followed by a GOSUB to line 510, where the formula is calculated and the profit (P) is set into P(X) upon return. Then, without incrementing the loop counter I, that same value is increased by 10 percent in line 760. Now we GOSUB to 510 again and recalculate. This time, upon the return, we put this value in P1(X). This continues for each data item until the loop is exhausted. At this point, the profit value for each item is in P(X) and the profit value based on the 10 percent increase is in P1(X).

Now all we need do is read these two arrays and determine the percentage difference between them for each item. This happens between lines 800 and 870. Line 810 figures the percentage and then takes the absolute value of it (eliminates the sign). Lines 820 and 830 determine what the sign of the percentage should be and assigns the string value of the sign to P$(X). Then in line 840, if the percentage is equal to or less than one, we don't want to print it, so we jump to the next I. Line 850 prints the percentage value on the screen, along with the string name of the item and the appropriate sign.

But why does Joe strip off the sign and then put it back on again? This question is left as an exercise for the reader. Hint: Remove the second part of line 810, remark 820, 830, 840 and change the PR in line 850 to PP.

Note that the sensitivity routine works with the

Generating Curves

If the sales/price curve in the accompanying article is not to your liking you can find a better one with this little program. The sales/price curve is, of course, not something that has been cast in concrete for all situations. It varies as a function of many factors. In the real world the curve can only be set by price-testing. There are some buyers who would not buy the product at any price. On the other end of the scale, there are some who would buy no matter how high the price (assuming it is something they really thought they needed.)

Price testing consists of offering the same product, at the same time, at different prices, to equal groups of potential buyers and tracking the results. The curve will be defined better when there are large enough numbers in each group, and with as many price breaks as possible or practicable.

The process is expensive and time-consuming. In our business model program choosing inverse square as the fall in buyers as price rises is not as close to reality as it could be, but is far more realistic than assuming no drop in buyers at all.

Program to generate empirical curves for Busmod.Bas

```
10 REM ** EMPIRICAL CURVE GENERATOR **
20 REM ** FOR 80-COL, 24 LINE SCREENS **
30 F$="###.#"
40 INPUT"ENTER SCALE OF CURVE (10 TO 110)";C
50 INPUT"ENTER SHAPING COEFFICIENT (50 TO 2000)";B
60 FOR I=100 TO 1150 STEP 50
70 A=(EXP(-I^2/B^2)*C)
80 PRINT STRING$(A/1.5,"-");:PRINT USING F$;A
90 NEXT I
```

Sample output of above program, using C at 100 and B at 600.

```
---------------------------------------------------------------- 97.3
---------------------------------------------------------------- 93.9
-------------------------------------------------------------- 89.5
------------------------------------------------------------ 84.1
---------------------------------------------------------- 77.9
-------------------------------------------------------- 71.2
------------------------------------------------ 64.1
---------------------------------------------- 57.0
------------------------------------ 49.9
------------------------------ 43.2
------------------------- 36.8
--------------------- 30.9
----------------- 25.6
-------------- 21.0
----------- 16.9
--------- 13.4
------- 10.5
----- 8.2
---- 6.2
--- 4.7
-- 3.5
-- 2.5
Ok

Run with C at 100 and B at 600
```

```
 1 - Material Cost/Unit  58        2 - Labor Cost/Unit  10
 3 - # Dir Mail sent  10000        4 - Dir Mail Postage  .125
 5 - Dir Mail Printing  .42        6 - Dir Mail $/Label  .075
 7 - % Return Dir Mail  .01        8 - MISC Costs/Unit  5.87
 9 - Fixed O'Head/Mo  2275         10 - Unit Sales price  200

WHICH ITEM # DO YOU WANT TO VARY? 10
FROM WHAT VALUE -- ? 50
TO WHAT VALUE ---- ? 850

VARYING --> Unit Sales price
STEP   PRICE #SOLD #DIRMAIL  FIXCOSTS  VARCOSTS  GROSS   EXPENSES  $ NET
 50      50   101   10000     2275      13660     5050     15935    -10886
130     130    96   10000     2275      13291    12480     15566    -3087
210     210    87   10000     2275      12626    18270     14901     3368
290     290    73   10000     2275      11592    21170     13867     7302
370     370    56   10000     2275      10336    20720     12611     8108
450     450    34   10000     2275       8711    15300     10986     4313
530     530     8   10000     2275       6790     4240      9065    -4826
610     610     1   10000     2275       6273      610      8548    -7939
690     690     1   10000     2275       6273      690      8548    -7859
770     770     1   10000     2275       6273      770      8548    -7779
850     850     1   10000     2275       6273      850      8548    -7699
PRESS RETURN FOR MENU?
```

Figure 1. This shows the effect on the bottom line by stepping unit sales price through a range. It appears that the optimum price is around $370. To get it more closely, you can run again and set the range from 360 to 380.

```
SENSIVITITY OF DATA. Only those terms causing
more than 1% change in bottom line are shown.

     $ NET       WITH A 10%
   CHANGES       INCREASE IN

   - 19  %       Material Cost/Unit
   - 4   %       Labor Cost/Unit
   + 29  %       # Dir Mail sent
   - 5   %       Dir Mail Postage
   - 18  %       Dir Mail Printing
   - 4   %       Dir Mail $/Label
   + 69  %       % Return Dir Mail
   - 7   %       Fixed O'Head/Mo
   + 61  %       Unit Sales price
```

Figure 2. This sensitivity run shows that unit sales price and percent return on direct mail will change the bottom line the most. This printout will not change unless new values are put into the data statements at the end of the program.

```
 1 - Material Cost/Unit  58        2 - Labor Cost/Unit  10
 3 - # Dir Mail sent  10000        4 - Dir Mail Postage  .125
 5 - Dir Mail Printing  .42        6 - Dir Mail $/Label  .075
 7 - % Return Dir Mail  .01        8 - MISC Costs/Unit  5.87
 9 - Fixed O'Head/Mo  2275         10 - Unit Sales price  350

WHICH ITEM # DO YOU WANT TO VARY? 7
FROM WHAT VALUE -- ? .01
TO WHAT VALUE ---- ? .015

VARYING --> % Return Dir Mail
STEP   PRICE #SOLD #DIRMAIL  FIXCOSTS  VARCOSTS  GROSS  EXPENSES  $ NET
 .01     350   61    10000     2275      10706    21350    12981    8368
 .0105   350   66    10000     2275      11075    23100    13350    9749
 .011    350   71    10000     2275      11444    24850    13719   11130
 .0115   350   76    10000     2275      11814    26600    14089   12510
 .012    350   81    10000     2275      12183    28350    14458   13891
 .0125   350   86    10000     2275      12552    30100    14827   15272
 .013    350   91    10000     2275      12922    31850    15197   16652
 .0135   350   96    10000     2275      13291    33600    15566   18033
 .014    350  101    10000     2275      13660    35350    15935   19414
 .0145   350  106    10000     2275      14030    37100    16305   20794
PRESS RETURN FOR MENU?
```

Figure 3. The unit sales price has now been fixed at $350. This run shows the effect of varying percent response to direct mail from 1 percent to 1.45 percent.

```
SENSIVITITY OF DATA. Only those terms causing
more than 1% change in bottom line are shown.

      $ NET      WITH A 10%
    CHANGES      INCREASE IN

    - 4   %      Material Cost/Unit
    + 27  %      # Dir Mail sent
    - 4   %      Dir Mail Printing
    + 32  %      % Return Dir Mail
    + 2   %      Unit Sales price
```

Figure 4. Changing the unit sales price (as in Figure 3) makes considerable difference in the sensivity of the other data. Compare this one with Figure 2 on the opposite page to see how things have changed.

values coded into the data statements, not with some value you may have entered in the first option of the program. As you optimize values in the data statements (by editing the line), run the sensitivity again. It is very interesting to see which items take on more or less importance with slight changes in other values. Another question for the reader: If the percentage change for all items could be brought to, or near, zero would that represent the optimum values for the model? If so, would they be realistic values?

Notes for other machines

The print lines in this program will not fit on 64 column screens. One way to fix this, crude but workable, is to change those print statements to LPRINT. That way, your output will be as shown in the examples, except it will be on your line printer. In any case, if you use the values now in the data statements, your output should be as shown in figure 1 with this article. You could also eliminate two columns from the printout, but that takes something away from the program. It would have been nice to have a 132 column screen for this program.

Modifications

The data statements are read from first to last and put into the E() array. It will be easy for you to add to or subtract from them. Keep the string portion of the data statement at 18 characters or less so they will fit on the screen.

If you want to put your own formula in you will need to change the data statements, the print statements in lines 470 and 500 and the calculation lines between 510 and 580. Don't forget that the last data statement must be DATA ,-1. Take whatever formula you want to use apart and then build it back up again. Keep in mind that you must define a variable before you can use it. If any part of the formula gets too messy to handle, use a new variable to represent it as we did in lines 540 through 560.

You can add your own step value for the main For...Next loop by adding a line at 445 to input the step. Then change line 490 to STEP (whatever variable you used at input.)

To avoid the necessity of editing lines to make permanent changes, you could input the data from a disk file instead of data statements. That way, any changes you make could be saved on diskette.

As with most of our programs, they could have been done in many ways and this is just one.

What about Joe?

Based on what Joe found out by playing with his program, he quit his job at Flash Electronics and is busy in his garage filling orders and making circuit boards. He has his eye on a nice piece of property just outside of town and is already planning another model program that will handle up to 100 employees and several product lines. There is just no stopping free enterprise! ■

```
100 REM ** BUSMOD.BAS* FOR CODEWORKS MAGAZINE **
110 REM ** 3838 S. WARNER ST. TACOMA, WA 98409 (206) 475-2219
120 CLEAR 1000 : 'USE ONLY IF YOUR MACHINE NEEDS TO CLEAR SPACE
130 DIM E(50),A$(50),P(50),P1(50),P$(50)
140 CLS : 'CLEAR SCREEN COMMAND, CHANGE TO SUIT YOUR MACHINE
150 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
160 PRINT "                    B U S I N E S S  M O D E L "
170 PRINT "                     Magarac's Widget Factory "
180 PRINT STRING$(60,"-")
190 PRINT
200 PRINT "  1) Run the model
210 PRINT "  2) Run sensitivity test
220 PRINT "  3) To quit
230 PRINT
240 INPUT"Your choice";Z
250 ON Z GOTO 280,650,270
260 GOTO 140
```

```
270 CLS:END
280 CLS
290 RESTORE
300 FOR I=1 TO 50
310   READ A$(I),E(I)
320   IF E(I)=-1 THEN GOTO 340
330 NEXT I
340 LN=I-1
350 IF Z=2 THEN RETURN
360 FOR I=1 TO LN STEP 2
370   L=I+1
380   PRINT I;"- ";A$(I)+" ";E(I);TAB(33);L;"- ";A$(L)+" ";E(L)
390 NEXT I
400 PRINT
410 INPUT"WHICH ITEM # DO YOU WANT TO VARY";X
420 IF X<1 OR X>LN THEN GOTO 410
430 INPUT"FROM WHAT VALUE -- ";S
440 INPUT"TO WHAT VALUE ---- ";ED
450 PRINT
460 PRINT"VARYING --> ";A$(X)
470 PRINT"STEP";TAB(8);"PRICE";TAB(14);"#SOLD";TAB(20);"#DIRMAIL";TAB(
30);"FIXCOSTS";TAB(40);"VARCOSTS";TAB(50);" GROSS";TAB(60);"EXPENSES";
TAB(70);"$ NET"
480 REM * LINES 510 THROUGH 580 CONTAIN THE EXPANDED FORMULA *
490 FOR I=S TO ED STEP (ED-S)/10
500   E(X)=I
510   US=E(3)*E(7)                    : ' NUMBER OF UNITS SOLD
520   US=US-(E(10)^2/3000)+2:US=INT(US):IF US<1 THEN US=1 : ' CURVE
530   R=US*E(10)                      : ' TOTAL INCOME FROM UNITS SOLD
540   E1=US*(E(1)+E(2)+E(8))          : ' MATERIAL,LABOR,MISC COSTS/UNIT
550   E2=E(3)*(E(4)+E(5)+E(6))        : ' DIRECT MAIL COSTS
560   E3=E1+E2+E(9)                   : ' TOTAL EXPENSES
570   VC=(US*(E(1)+E(2)+E(8)))+E2     : ' VARIABLE EXPENSES
580   P=R-E3                          : ' PROFIT = INCOME LESS EXPENSES
590   IF Z=2 THEN RETURN
600   PRINT I;TAB(9);E(10);TAB(15);US;TAB(21);E(3);TAB(30);INT(E(9));T
AB(40);INT(VC);TAB(50);INT(R);TAB(60);INT(E3);TAB(70);INT(P)
610 NEXT I
620 PRINT"PRESS RETURN FOR MENU";:INPUT X1:GOTO 140
630 END
640 REM ** SENSIVITY OF DATA ROUTINE **
650 CLS
660 PRINT"SENSIVITITY OF DATA. Only those terms causing"
670 PRINT"more than 1% change in bottom line are shown."
680 PRINT
690 PRINT"          $ NET      WITH A 10%"
700 PRINT"        CHANGES      INCREASE IN"
710 PRINT
720 GOSUB 290
730 FOR X=1 TO LN
740   GOSUB 510
750   P(X)=P
```

```
760    E(X)=E(X)+(E(X)*.1)
770    GOSUB 510
780    P1(X)=P
790 NEXT X
800 FOR X=1 TO LN
810    PP=INT((P(X)-P1(X))/P(X)*100):PR=ABS(PP)
820    IF P(X)<P1(X) THEN P$(X)="+"
830    IF P(X)>P1(X) THEN P$(X)="-"
840    IF PR=<1 THEN GOTO 870
850    PRINT TAB(10);P$(X);PR;TAB(16);"%";TAB(22);A$(X)
860    PR=0
870 NEXT X
880 PRINT
890 PRINT"PRESS RETURN FOR MENU";:INPUT X1:GOTO 140
900 END
910 REM ** DATA STATEMENTS FOLLOW **
920 DATA Material Cost/Unit,58
930 DATA Labor Cost/Unit,10
940 DATA # Dir Mail sent,10000
950 DATA Dir Mail Postage,.125
960 DATA Dir Mail Printing,.42
970 DATA Dir Mail $/Label,.075
980 DATA % Return Dir Mail,.01
990 DATA MISC Costs/Unit,5.87
1000 DATA Fixed O'Head/Mo,2275
1010 DATA Unit Sales price,200
1020 DATA ,-1
```

Programming Notes

Recently, while re-working a TRS-80 Model I program, we found an interesting idea. The program printed the value of a variable, then asked for input of that same variable, like this:

```
PRINT A;:INPUT"What is A";A
PRINT B;:INPUT"What is B";B
```

The value in A or B would then show, along with the input prompt, and you could step through the lines with the Return/Enter key without disturbing the variables already there. If you wanted to change one, you simply entered the correct value and kept on stepping. Microsoft BASIC after version 5.1, however, will not let you do that. It changes the variables to null when you step through without changing them.

Here is a password scheme that should appeal to touch-typists. Most passwords can be deciphered easily by anyone knowing anything about you. Too many of us use our name spelled backward, or something equally easy to figure out. This scheme lets you use some password you cannot forget, like your name, or a pet phrase. Simply move your home key position on the keyboard up one line of keys. Now type your favorite password and see how it looks. For example, CodeWorks would look like this: d9e3294iw. You don't need to remember that jumble of characters and numbers, just that the password is CodeWorks and you moved your home keys up one line. You can take it from there and figure out the other variations.

ShareWare

An alternative software source

Al Mashburn, Technical Adviser. If you can think of it, it has probably already been done somewhere by someone. The trick is finding out who and where. Al uses his modem often and offers us an interesting alternative way to acquire software.

I'm like you. I want good software for a reasonable price. If it doesn't work, I want to be able to bring it back. Ok, when you get done laughing, let me tell you a little secret. You can get all those things and more with a new concept that is really catching on. But allow me to digress a little first.

I like to program. In fact, there are times when I go out of my way to find a problem just so I can write a program to solve it. I avoid buying software mostly because I can do it myself. But let's be realistic. If I needed a full-featured communications program I would need to spend months to write it (assuming I could write it in the first place.) I use this example because that is what got me started on this whole thing.

I wanted to use my modem with my new PC, so I wrote a terminal program. It worked, but I couldn't download or upload programs. Rather than try to write in all those features, I went to buy a program instead. Well, I am not that old, but I nearly had heart failure when I saw the price of PC programs. Programs for my old computer were in the $19 to $59 range, but these guys must be real proud of their stuff - $159 for a terminal program that did what I wanted! I only paid $89 for my disk drive - there must be a better way.

About this time a friend came over and after hearing my problem handed me a diskette and told me it contained the program I wanted. I told him I don't pirate programs anymore, and he told me it was no problem because the author *wanted* people to copy his program and hand it out as much as possible. Sure, I thought, and maybe the author wants us to take his new car out for a spin too. He said to simply boot the program and look. I did, and there on the opening screen was this message: "If you find this program to be of value, please send $25 to so and so...", etc. At the bottom of the screen it said, "Please copy this program and give it to your friends."

You could have knocked me over with a feather. I used the program and liked it very much. I sent the $25 to the author and got a nice note back advising me to call a bulletin board in my area and download the latest revision. While I was doing that, I found literally hundreds more programs distributed the same way, all the authors depending on the integrity of the people using their programs.

The concept has many names: Shareware, Freeware, Tryware, to name just three. One thing that should be noted is that this is not public domain software. It is copyrighted software that is for sale. What is different is that instead of spending a bundle on advertising, the programs are distributed, for the most part, on public bulletin board systems or passed out at user group meetings.

So how do you find these programs, and when you do, how do you know which ones are worth downloading? The answer to the first part of this question is: On bulletin boards. Of course one of the largest is Compuserve. In the Special Interest Groups (SIGs) there are hundreds of programs to download. The same is true for many private bulletin board systems around the country. There is one in the Seattle area which has over 100 *megabytes* of programs to download - and more are being added every day.

The second part of the question is much more difficult to answer. If you are calling long distance, you certainly don't want to spend 10 minutes downloading a program only to find out that you not only don't want to pay for it, but you don't even want it to darken your diskette. As of this time, I

have yet to find a way of knowing in advance what I am getting. What is needed is a sort of clearing house for these programs that will at least catalog what they do and, hopefully, rate them in some way. The answer may be a user supported media where people can get their needs together.

Finally, if you have wondered why I haven't told you what kind of programs are available, it is because if you can name any advertised program there is probably a Tryware program just like it. I have mentioned a communications program. I have also downloaded a Side-Kick-like windows program with note pad and ASCII table on line at all times. The largest category of programs available seems to be utility programs. There, you can find everything from re-naming sub-directories to un-erasing files. At least 50 different programs were on one system I recently used.

So there you have it. There is a gold mine of programs out there if you take the time to find them. Just remember that the people writing these programs are counting on your honesty to make their living, so if you really use a Tryware program, please take the time to send the author the small donation he requests. It will keep a good flow of high quality, low price software coming. ■

Programming Notes

Many times in this magazine, we refer to the MERGE command. It is a very powerful device that allows considerable freedom in building data sets or program lines. Here is how to use it.

Assuming you have a program loaded in memory, MERGE will allow you to load a second program from disk and merge its lines with the one already in memory. The program coming from disk must be in ASCII, which means that it had to have been saved with the ASCII identifier, as in SAVE"filename",A. If you do not use the ASCII identifier, the program will have been saved in a compressed format, and cannot be merged.

Two things can happen when you merge two programs. If a line number in the program coming from disk is the same as one in the memory-resident program, the memory-resident program line will be overwritten. If the line number coming from disk does not have a corresponding line number in the memory-resident program, the line coming from disk will be added to the memory-resident program.

If line numbering is a problem, you can call up the lines you wish to merge and renumber them and save them back. The memory-resident program need not necessarily have been saved in ASCII, only the program from disk that will be merged with it needs to be. You can continue to merge programs with the memory-resident program as long as you have programs to merge and memory holds out.

The procedure used to merge is simple: LOAD (don't run) the program you will be merging into, as in:

LOAD"PROG1.BAS"

MERGE"PROG2.BAS"

and that's it. PROG2 will now be merged into PROG1. PROG2 may have been a set of data statements, or it may have been a set of odd-numbered remark lines that you wanted to insert at the proper places in PROG1.

Let's take an example from this issue, BUSMOD.BAS (You will find it on page 6 of this issue.)

The main structure of the program is there. There is no need to invent that wheel again. But the data may not fit you at all, it's for Magarac's Widget Factory. Here is how to make it do what you want.

Write a new program with only the following lines. Line 170 to change the name to what you want. Lines 470 and 600 to make headings appropriate to your application. Lines 510 through 580 to produce the calculations for your problem, and the data lines at the end of the program. You can have more or less data lines; the program will account for it as long as the last data item is DATA ,-1.

Save this program as an ASCII file. Now load BUSMOD.BAS and then merge the program you just created. You now have a completely modified BUSMOD. You could actually have several different merge files that could overlay BUSMOD. If you have more data statements than the original BUSMOD, don't worry about it; they will all be overlaid. If you have less, you will need to delete those left over before you run.

Having done this a couple of times, you will certainly begin to appreciate the power of MERGE.

Beginning Basic

Money is money, but to some people it means dollars and cents, to others marks and pfennigs and yet others see it as rubles and kopecks. It's still money, and how you view it depends upon who and where you are. When you get right down to it, money is either some grade of paper or metal. Computer bytes have some of the same characteristics.

If money is basically paper, computer bytes are basically voltage levels. Eight little switches inside a computer chip can either be on or off. If a switch is off, its voltage level is at or near zero. If the switch is on, the voltage level is at or near 5 volts. There are several terms that describe "on" and "off". "On" can be called "high", "one", "true"; while "off" is usually "low", "zero" or "false". Even these terms are not always absolute. In some machines a low is an enabling level and the high is disabling.

Since there are eight switches in one byte, the number of unique combinations they can take is 256 (including the cases when all are off and all are on.) Let's take the following case:

0 1 0 0 0 0 0 1

The "1's" represent switches that are on, the zero's those that are off. Reading from right to left and using the binary number system, we come up with a one and a 64, the total being equal to 65. The place values from right to left are: 1, 2, 4, 8, 16, 32, 64 and 128. All switches off would equal zero, all on will equal 255, making the 256 possible combinations we mentioned earlier. Now, if you have your machine on and are in BASIC, ask it to print CHR$(65). It should print the capital letter A. If you now ask it to print ASC("A"), you should see it print the number 65. The command PRINT ASC("A") is telling the computer to print the ASCII (American Standard Code for Information Interchange) value of the letter A. ASCII is nothing more than looking at a binary number (or eight little switches) and reading the *decimal* value of them. As you can see, it takes eight bits (called a byte) to define any character, number or symbol in your computer. Asking your computer to PRINT CHR$(65) tells it to print the character that the ASCII (decimal) number 65 stands for, in this case, A.

There are other ways to look at the eight switches than in binary or decimal. One way is called hexadecimal. Now, we are going to look at the binary number above as two groups of four binary digits. Each group of four can only count to sixteen. But how would we name the numbers above 9 if there are sixteen? This is where hexadecimal comes in. It counts 10 as A, 11 as B and so on through 15, which is the letter F. In our sample number above, we don't run into anything above nine, but let's look at it anyway. The right most group of four is equal to one. The left group is equal to four. So the hexadecimal value of A is 41, even though the decimal and ASCII values of it are 65. It's still the same number and represents the letter A.

Now let's take a different binary number:

0 1 1 1 1 0 1 0

The decimal value of this binary number is 122. So is the ASCII value of it. In hexadecimal though, it turns out to be a value of 7A. In your computer, it prints the lower case letter "z". Why is 122 equal to z? The ASCII people did that a long time ago, when teletypes were in vogue, before computers were even a glint in anyone's eye.

There is still another way to look at binary numbers. It seems popular to look at them in different ways because they are so hard for us humans to read. We can read them in "Octal". To do this, we group the binary digits into three's.

0 1 1 1 1 0 1 0

Now, reading from right to left, we get a two, a seven and a one. So 172 is the octal value of the binary number that represents lower case z. Octal isn't used that much anymore.

Is this "need to know" information? Yes, it helps to know if you get into the logic AND, OR, XOR and others, which can be used in BASIC to do some interesting things. In the next issue, we will present a little program that will allow you to look around inside your computer's memory and see what things look like - in ASCII. ■

Puzzler # 4

Our puzzler in the last issue was to fill an array with the numbers of playing cards using the least number of BASIC statements. The DIM statement was given.

The problem can be solved with three statements: a For.. statement, a statement to stuff the array, and the Next statement. All three, of course, could be put into one multiple statement line of code.

We were rather surprised at the variation (and ingenuity) of the answers we received. The solution we had in mind was like that in Figure 3, using integer division. The solution we like the best, and our "hands down" (no pun intended) winner, is Robert E. Brown of Schenectady, New York. His solution appears in Figure 1. He also showed how he arrived at the solution, and it goes like this:

```
B(I+1)=(I+1)+1+100+(100*INT(I/13))—13*INT(I/13)
                    or
B(I+1)=I+102+(100*INT(I/13))—13*INT(I/13)
                    or
B(I+1)=I+102+87*INT(I/13)
```

Other solutions are shown in Figures 2 through 5. The solution in Figure 2 uses the Mod function, that in Figure 3 uses integer division. Larry Abbott of Wyomissing, Pennsylvania, sent in the solution in Figure 5. So did Eldon Clark of St. Petersburg, Florida. They both used Boolean concepts in their solutions. In spite of the length of the statement, it still works and qualifies because it is just one statement. Those who used only three statements are:

Jim Offenbacher, Huntsville, AL
Mark Gardner, N. Hollywood, CA
R. B. Hodges, Amherst, NH
D. E. Harlow, Altadena, CA
Sara Pinkert, Sturgeon Bay, WI
Edward Engberg, Santa Barbara, CA
Robert E. Brown, Schenectady, NY
E. L. Clark, St. Petersburg, FL
Larry Abbott, Wyomissing, PA
Robert T. McCay, Neshanic Station, NJ

Jerry Bails of St. Clair Shores, MI, says the DIM statement is unnecessary, and submitted this defined function:

```
DEFFNB%(I%)=87*(INT((I%-1)/13))+I%+101
```

David Lovelace of Springfield, MO (as well as several others) asked why we start at 102 and go through 414, rather than call the ace 1 and the king 13. Taken by itself, it seems more logical to do it that way. However, in both Bridge and most poker games, the ace is counted high. The exception in straight draw or stud poker is the 5-high straight, where the ace counts low. This is the only exception you need make then in evaluating hands, since the ace at 14 would always be higher than any other card. It is easier this way than counting the ace low and making exceptions for all the other cases.

Thank you all for such interesting and innovative answers. Now on to puzzler 4.

Puzzler #4

Given these two input lines:

```
10 INPUT"VALUE FOR A";A
20 INPUT"VALUE FOR B";B
```

what is the least possible amount of code following these lines that will print the larger of the two values on the screen as well as print 0 if the two values are equal? Start your answer with line 30.

Does this sound too easy? Hint: It can be done with just one BASIC statement, and it starts with the command PRINT. What is it? ■

```
10 DIM B(52)
20 FOR I=0 TO 51
30   B(I+1)=I+102+87*INT(I/13)
40 NEXT I
```

Figure 1. The most concise answer to the puzzler.

```
10 DIM B(52)
20 FOR I=0 TO 51
30   B(I+1)=100*INT(I/13+1)+I MOD 13+2
40 NEXT I
```

Figure 2. Similar to the one above, but longer.

```
10 DIM B(52)
20 FOR I=1 TO 52
30   B(I)=(((I-1)\13)+1)*100+I-(((I-1)\13)*13)+1
40 NEXT I
```

Figure 3. Using integer division.

```
10 DIM B(52)
20 FOR I=1 TO 52
30   B(I)=((INT((I-1)/13)+1)*100)+(I-(INT((I-1)/13)*13)+1)
40 NEXT I
```

Figure 4. Similar to Figure 3, but without integer division.

```
10 DIM B(52)
20 FOR I=1 TO 52
30   B(I)=((I<14)*-1)*(101+1)+((I>13 AND I<27)*-1)*(188+I)+((I>26
AND I<40)*-1)*(275+I)+((I>39 AND I<53)*-1)*(362+I)
40 NEXT I
```

Figure 5. Using Boolean operators.

Merge/Sort

Sorting files too big to fit in memory

T. R. Dettmann, Associate Editor. There is an interesting concept at work here, more so than just sorting files too large for memory.

Sorting records into order is a common operation on most computer systems. It is one of the things the computer does best since the operation is repetitive. Most books and magazine articles deal with the problem of sorting many records in memory, but very seldom do you find anything on how to sort files that are so large that the entire list of items cannot fit into memory. This article deals with just such problems.

A few years back, a friend was maintaining a mailing list of over 25,000 names on 5 inch floppy diskettes. Sorting this file was a real problem for mailings. However, there are techniques used for sorting particularly large files. Professional programmers have been using them for years in sorting tape files on large computers. The basic technique is called the "sort merge" operation.

The idea behind the sort merge operation is to accomplish the whole job by breaking it into small steps and doing the job little by little. For large jobs, you may have to let it run for some time, but you could safely restructure an entire database this way. Let's look at how a sort merge is done and see how it is put together.

We can look at the basic operation as a series of simple steps:

1. Open the input file.
2. Split the input file into smaller files by reading as many items as possible, sorting them into order and then writing the items out to a separate, smaller, file.
3. Merge the smaller files together by opening them together and reading the smallest item from the files (the smallest in each file will be the first because they are sorted) and writing that item to the new master file.

The basic operation sounds rather easy. It is, actually, but there are some interesting complications that you have to work with if it is to work right.

Program Notes

Let's go over the sample program and see what makes it work. The lines through 260 serve as an initialization for the program. The important code is between lines 150 and 240. In line 150, we set up the parameters for the program which make it very flexible. We have set some arbitrarily low parameters here to be able to do sample runs with small files in a short amount of time. These parameters are:

NL - The number of lines to read into memory for sorting. This parameter is determined by the maximum capacity of program memory for string lines. This will vary from computer to computer, though even a small system can typically hold 300 to 400 lines. The number is set here to 10 so that we can see the program work quickly.

NFL - The number of files that will be declared upon entering BASIC to run the program. This is set to four so that three files could be merged into one. Depending on the computer, you can typically handle up to 16 files maximum. On the Kaypro, where this program was created, we entered BASIC by typing "MBASIC /F:4" in order to get the program to work.

NX - The maximum number of temporary files to generate. Since a list of these files is kept in memory (not really necessary, just convenient), this will serve as the dimension of the array of names.

Line 190 sets up the line array (LN$), the temporary file name list (FF$), and the merge file list (FC$). The root name for all temporary files is "TEMP" and the number of the current temporary files is set to zero. On some systems, it should be noted that you cannot set the array dimensions with variables as has been done here. The basis for doing this here is to make sure that if we change the parameters, the array dimensions will change appropriately. One of the many things which can go wrong with a program is changing a parameter and not changing it everywhere. This helps to assure it won't happen with this program.

Next we get the names of the desired input and

output files. Then lines 330-440 break the input file into a series of smaller temporary files. Each file will have no more than NL lines in it, all sorted in alphabetical order. Subroutine 690 first reads NL or fewer lines. The subroutine at 800 sorts them using a standard memory sort called the "Shell Sort." This subroutine could be replaced with a call to a fast machine language sort if you have one available. Finally, subroutine 950 writes the sorted lines into a temporary file.

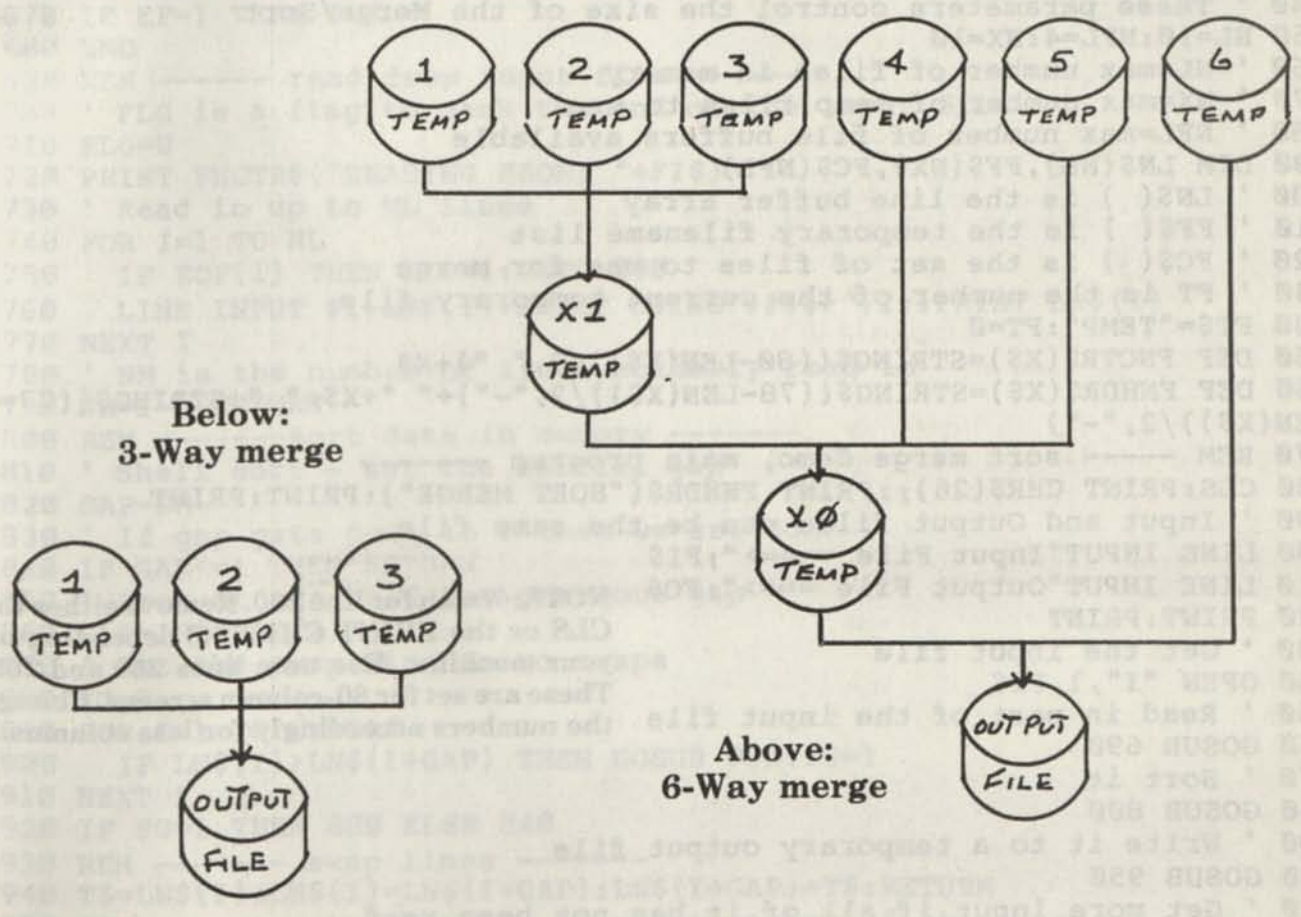
When the input file is completely broken up, it is necessary to recombine the sorted files into a single file. This is the merge operation and takes place in lines 450 through 680.

Subroutine 1060 chooses an output filename for each merge and selects which files to merge. If we have enough file buffers available to merge all of the output files, then we will just merge them all directly into the final output file.

If there are not enough file buffers available we merge as many as we can into a series of intermediate files, TEMPX0 and TEMPX1. Why two files? We do this so that we can merge them with other files in the next step. If there are still too many files, then we again merge to intermediate files in the next step until we can finally get them all together. The figures show some typical file combinations for sort merge operations.

Once the files are selected for the merge and the output filename is assigned, we read the first line from all of the input files in lines 480 through 540. With a line entered from each file, we look to see which line is smallest using subroutine 1340. When the line is selected, subroutine 1340 also reads in a new line from the file the line was selected from. The selected line is written to the output file in line 610. If there was no line (we are at the end of file on all input files) we check in subroutine 1460 to see if we are all done.

You can visualize how the program works by



imagining a deck of playing cards. In order to sort them, we could break them up into small stacks and sort each stack into order. With each stack in order, we then take the smallest card from the top of the stacks that are visible and place it on a final stack. If we just keep repeating this step over and over again, we will eventually have the entire card deck sorted into order.

The computer makes this technique simple and automatic. It can operate without anyone worrying about it and can handle quite large files. With some modification, the program could be made to deal with files that go across a series of disks. It does require considerable disk space for temporary files, but if you can't solve the problem any other way, it certainly beats doing nothing.

The technique used here could be used to do some other interesting things:

1. You could take several files, group them, sort them together into a common file during the merge operation.
2. You could break a file into a set of standard size files, all of which are sorted in order.
3. You could set up a standardized grouping for files and put the lines into them using the sort merge operation.
4. For large files that cover many disks, you could have the program prompt to load and unload disks as needed for the operation and thereby deal with very large numbers of records indeed.

Can you modify the program to do any of these things? Try and see, it should be an enjoyable learning experience. ■

```
100 REM * MGSORT.BAS * MERGE/SORT FOR ISSUE 5 CODEWORKS MAGAZINE
110 REM * 3838 S. WARNER ST. TACOMA, WA 98409 (206) 475-2219
120 REM * WRITTEN BY TERRY R. DETTMANN
130 DEFINT A-Z: ' Clear 10000 also if your machine needs it.
140 ' These parameters control the size of the Merge/Sort
150 NL=10:NFL=4:NX=10
160 ' NL=max number of files in memory
170 ' NX=max number of temp files to use
180 ' NFL=max number of file buffers available
190 DIM LN$(NL),FF$(NX),FC$(NFL)
200 ' LN$( ) is the line buffer array
210 ' FF$( ) is the temporary filename list
220 ' FC$( ) is the set of files to use for merge
230 ' FT is the number of the current temporary file
240 FT$="TEMP":FT=0
250 DEF FNCTR$(X$)=STRING$((80-LEN(X$))/2," ")+X$
260 DEF FNHDR$(X$)=STRING$((78-LEN(X$))/2,"-")+" "+X$+" "+STRING$((77-
LEN(X$))/2,"-")
270 REM ----- sort merge demo, main program -------
280 CLS:PRINT CHR$(26);:PRINT FNHDR$("SORT MERGE"):PRINT:PRINT
290 ' Input and Output files can be the same file
300 LINE INPUT"Input File ====>";FI$
310 LINE INPUT"Output File ===>";FO$
320 PRINT:PRINT
330 ' Get the input file
340 OPEN "I",1,FI$
350 ' Read in part of the input file
360 GOSUB 690
370 ' Sort it
380 GOSUB 800
390 ' Write it to a temporary output file
400 GOSUB 950
410 ' Get more input if all of it has not been read
```

NOTE: Watch for line 280. Remove either the CLS or the PRINT CHR$(26) depending on your machine. Also note lines 250 and 260. These are set for 80-column screens. Change the numbers accordingly for less columns.

```
420 IF FLG<>1 THEN 360
430 ' Close all files
440 CLOSE
450 ' Start the merge opration
460 GOSUB 1060
470 ' Open the files
480 OPEN "O",1,FC$(1)
490 PRINT FNCTR$("OUTPUT FILE: "+FC$(1))
500 FOR I=2 TO NZ
510   OPEN "I",I,FC$(I)
520   PRINT FNCTR$("INPUT FILE: "+FC$(I))
530   IF EOF(I) THEN LN$(I)="":GOTO 540 ELSE LINE INPUT #I, LN$(I)
540 NEXT I
550 ' Choose the smallest line
560 GOSUB 1340
570 ' If there is no smallest line then we are done with this
580 '  pass through the files.
590 IF LN$="" THEN 630
600 ' Put the smallest line into the output file
610 PRINT #1,LN$:K=K+1:PRINT USING"####>";K;:PRINT LN$
620 GOTO 560
630 CLOSE
640 ' Check to see if we are all done
650 GOSUB 1460
660 ' If not done, go get the rest
670 IF EF=1 THEN 460
680 END
690 REM ------ read from input file ------
700 ' FLG is a flag to mark the end of the input file
710 FLG=0
720 PRINT FNCTR$("READING FROM: "+FI$)
730 ' Read in up to NL lines
740 FOR I=1 TO NL
750   IF EOF(1) THEN FLG=1:GOTO 790
760   LINE INPUT #1,LN$(I):PRINT USING"####>";I;:PRINT LN$(I)
770 NEXT I
780 ' NM is the number of lines actually read in
790 NM=I-1:RETURN
800 REM -------sort data in memory -------
810 ' Shell sort - set the initial gap
820 GAP=NM
830 ' If gap gets down to 1 then we are done
840 IF GAP<=1 THEN RETURN
850 ' Look at one-half the previous gap
860 GAP=INT(GAP/2)
870 ' Set the swap flag to no swaps
880 FG=0
890 FOR I=1 TO NM-GAP
900   IF LN$(I)>LN$(I+GAP) THEN GOSUB 930:FG=1
910 NEXT I
920 IF FG=1 THEN 880 ELSE 840
930 REM ------- swap lines -------
940 T$=LN$(I):LN$(I)=LN$(I+GAP):LN$(I+GAP)=T$:RETURN
950 REM -------write data to temporary output file -------
```

```
960 ' FT is the current temporary file number
970 ' Make NF$, the current temporary filename and save it
980 ' If we have used NX files, then cut off the input
990 FT=FT+1:NF$=FT$+MID$(STR$(FT),2):FF$(FT)=NF$:IF FT>=NX THEN FLG=1
1000 ' Write the lines to the temporary file
1010 OPEN "O",2,NF$
1020 PRINT FNCTR$("TEMPORARY FILE: "+NF$)
1030 FOR I=1 TO NM:PRINT #2,LN$(I):PRINT USING"####>";I;:PRINT LN$(I):
NEXT I
1040 CLOSE 2
1050 RETURN
1060 REM ------ pick output file and input offset -------
1070 ' F1 is the first file to read
1080 ' F2 is the last file to read
1090 ' FF is the number of the intermediate merge file
1100 ' If your BASIC doesn't have MOD use: FF=FF-INT(FF/2)*2
1110 F1=F2+1:F2=F2+NFL-1:FF=(FF+1)MOD 2
1120 ' Make the intermediate merge file name
1130 FF$=FT$+"X"+MID$(STR$(FF),2)
1140 ' Two cases: FC$(1)="" = first time through
1150 '            FC$(2)<>"" = already been through
1160 IF FC$(1)="" THEN 1210
1170 ' We need to merge the last intermediate file and one
1180 '   less temporary file.
1190 FC$(2)=FC$(1):F2=F2-1:J=2
1200 GOTO 1230
1210 J=1
1220 ' Check for last file
1230 IF F2>FT THEN F2=FT
1240 ' If this is the last pass, then write to desired output file,
1250 '  otherwise, we will use an intermediate file.
1260 IF F2=FT THEN FC$(1)=FO$ ELSE FC$(1)=FF$
1270 ' Get the temporary file names
1280 FOR I=F1 TO F2:J=J+1
1290   FC$(J)=FF$(I)
1300 NEXT I
1310 ' NZ is the number of files for this go around.
1320 NZ=J:K=0
1330 RETURN
1340 REM ------- select smallest input line -------
1350 FOR I=2 TO NZ:IF LN$(I)<>"" THEN 1390
1360 NEXT I
1370 LN$=""
1380 RETURN
1390 FC=I:LN$=LN$(I):J=I+1
1400 IF J>NZ THEN 1440
1410 FOR I=J TO NZ:IF LN$(I)="" THEN 1430
1420   IF LN$>LN$(I) THEN LN$=LN$(I):FC=I
1430 NEXT I
1440 IF NOT EOF(FC) THEN LINE INPUT #FC, LN$(FC) ELSE LN$(FC)=""
1450 RETURN
1460 REM ------- check for end of processing -------
1470 IF FC$(1)<>FO$ THEN EF=1 ELSE EF=0
1480 RETURN
```

VXREF.BAS

A line number, variable reference program

J. Melvin Jones 71 West St. Warwick, NY 10990 A utility program that gives a sorted line number and variable cross-reference. It also gives you a paged listing of your program.

It all started about three months ago. While I was rummaging around in the attic, I came across an old paper tape that was the only existing copy of a game that I wrote back when I was in college. At the time I wrote it, personal computers (and I don't think they even called them that quite yet) had about 1K (average) of memory, a hexadecimal display and keyboard, and if you were really lucky, could save programs using a cassette recorder. Of course, my game was written on the school mini-computer (in BASIC, fortunately), so when I left that school I was suddenly without a system capable of running it.

Ergo, it was packed up and relegated to the place in the attic where I keep all the things that are "too good to throw away but not really good for anything in particular."

Envision the hands spinning on a clock and pages flying away from a calendar (oooh! how Hollywood)! Now I have my own computer room with a micro-computer that has all the bells and whistles of that mini- computer I used in college, and lo and behold, I came across this program. Fate works in mysterious ways.

After loading the program onto diskette, I settled back to do the necessary conversion to Microsoft BASIC (I don't remember the name of the original dialect, but it sure didn't look *anything* like the ones I use now.) After several frustrating hours, I decided that I had a serious problem.

You see, when I wrote that game, I was but a novice when it came to programming. I had yet to learn the value of writing comments to remind me of how the program worked. What I have now to work with is about 1000 lines of the notorious "spaghetti code" that everyone warns you about when you begin writing programs in BASIC. The flow of the program, apparently, started somewhere near the end of the code and, jumping quite often to the beginning, wound its way toward the middle where, as far as I could tell, it entered the Twilight Zone.

I had to find some method of documenting the flow of the program so that I could unravel the mysteries that lay therein. The result, after a little back and forth with the editors of CodeWorks, is the program that is presented here.

The Concept

VXREF is a program, written entirely in BASIC, which analyzes a BASIC program that is stored as an ASCII file, and generates a complete cross reference listing for that program. The report includes a complete source code listing of the program (in neat, paged format), a list of all the line numbers referenced by GOTO, GOSUB, IF...THEN, RESUME statements, etc., and an alphabetical listing of each variable used in the program, specifying the line numbers in which they are used. If all that seems a bit confusing, let me try to explain a little more clearly.

Let's take a typical example. Suppose that you have written a BASIC program and that program, for some unknown reason, suddenly executes line 2000 at what appears to be random intervals. No matter how you try, you can't seem to find any reason for this erratic behavior, and you are preparing to commit suicide because the program simply has to be ready by tomorrow morning. This is the perfect time to use VXREF.

First, save the errant program in a sequential ASCII file (SAVE"filename",A). Once this is accomplished, you run VXREF. When VXREF asks you for the source file, feed it the name of the file containing the program; make sure that the line printer is on line. Then go get a cup of hot cocoa to soothe your nerves.*(Ed. note: Try two cups, especially on long programs. After that, we compiled the program and the difference in operation speed was no less than astounding. If you have access to a good BASIC compiler, by all means, this is a program that will benefit from it.)*

When you return to the computer, and VXREF has completed its task, you should have a report that will supply you with a lot of information that

can help you with your problem. The report (which you can tear off from the line printer) is in three parts: the Program Source Code Listing, the Line Number Reference Listing and the Variable Reference Listing.

The program source code listing is just that. It lists your program, broken down into nice neat pages, so that you can refer to it when trying to find your problem. A lot of programmers try to debug their program entirely interactively right on the video display, and, while this saves paper, a video display gives too much of a tunnel vision perspective to let one really see the program that was written.

The line number reference listing is the next section of the report, and is particularly relevant to our hypothetical problem. The line number reference listing shows you the line numbers which are used by the various branches within your program and where those branches occur. In our example, you would look down the list of line numbers until you find 2000 and under "Referenced in," you will find a list of all the line numbers containing instructions which could branch to line 2000. To solve your problem, you need only to examine each of these instructions and determine why, and when, they branch to 2000. If they are all correct, you have discovered that the program is simply "falling through" to line 2000 from above.

The variable reference listing performs a similar function, except that it tells you the line numbers of the instructions which use each variable. For example, if you have determined that the value of "A" is not what it should be at some point in the program, you can use the variable reference listing to examine each line in which the variable "A" appears and, by checking every statement that can alter "A", find out where the program is going wrong. When I was first testing VXREF, I had a problem with X$. Fortunately, VXREF showed me that X$ is used in several places and before entering the routine that was giving me problems, contained some residual information. From that realization, I fixed the problem immediately by setting X$ to null.

One additional feature of the variable reference list is the discrimination between simple and subscripted variables. If the variable name is suffixed with a "+", the name refers to a subscripted variable. Also, explicit type declarators (such as "$" to denote string variables, and "%" to denote integer variables) are preserved, thereby allowing easy determination of questions relating to variable types.

VXREF need not be limited to debugging. A copy of the cross reference listing should be a standard part of your program documentation, along with your programming notes, so that you will be able to remember how the program works ten years from now when you find it in your attic.

The Program

The code for VXREF is fairly self-documenting. Operation is straightforward and the program flow should be quite easy to determine from the listing.

The program reads one line at a time from the ASCII file and then echoes it to the line printer. After the line has been printed, it eliminates lexical constants (string literals like -Programmers do it with their fingers-) and any REMark statements. These removals are handled by the subroutines beginning at lines 11610 and 11500, respectively. Note that the dialect of BASIC that I've used allows the substitution of an apostrophe for REM, and that unmatched quotations are assumed to be closed at the end of each line.

Once the line has been prepared, and miscellaneous delimiters have been converted to spaces (lines 11720-11820), it is scanned from left to right by the "Token Parser" subroutine (lines 11830-12220). This routine finds the first character of each token in the line, and then determines if it is a keyword, constant, line number or variable name. If it is a variable name or line number, it is added to the appropriate list. Otherwise, it is ignored.

When a line number of variable name is added to a list, the list is kept in alphanumeric order (ASCII collating sequence.) This allows the list builder subroutines (lines 12230-12540 for line number references and lines 12550-12860 for variable references) to employ a quick Binary Search algorithm to determine if the variable has been referenced before. All references are packed into strings to reduce memory overhead and allow efficient dynamic allocation.

After the file has been evaluated, the tables (which are already in alphanumeric order) are printed, and the program is finished.

Conversion to other dialects

VXREF does not transcend the capabilities of most versions of Microsoft BASIC. The major alterations that will need to be made are to the BASIC keyword database located in lines 13060 through 13240. These keywords are the basis for the token parser's decision as to what is a variable

name and what is not.

The BASIC keyword database *must* be in alphanumeric order, because a Binary Search algorithm is used by the token parser. The manual for your BASIC probably has an appendix which lists all of the reserved words in alphabetical order. Furthermore, if delimiters are required (for example, my version of BASIC requires that each keyword be followed, or delimited, by a space), the required delimiter must be included in the database. This will guarantee that the keyword will be recognized as such.

Finally, if the keyword can typically be followed by a line number reference (GOTO, GOSUB, THEN, ELSE, RESUME, etc.), the data entry must be suffixed with "#" so that the token parser will know that if a number is encountered immediately after the keyword, it is a line number and not a constant.

Conclusion

The program listing shown is written for the TRS-80 Model 4/4P running under TRSDOS 6. It has been tested on several quite complex programs (including itself) and has never failed to provide accurate and enlightening results. Good luck and happy computing. ●

```
10000 '-----------------------------------------------------------
10010 '          VXREF (Version 1.01.00)  Written by J. Melvin Jones.
10020 '          Written for CodeWorks Magazine, December 20, 1985.
10030 '-----------------------------------------------------------
10040 '          MAIN PROGRAM SECTION:
10050 '
10060 ' This program processes a BASIC program, stored in an ASCII
10070 '  text file, and generates a report in three parts.
10080 '
10090 '          Section One:  The program listing (in paged format).
10100 '          Section Two:  A Line number cross reference listing.
10110 '          Section Three: A variable cross reference listing.
10120 '-----------------------------------------------------------
10130 '
10140 ' First we must initialize some important information...
10150 '
10160 FALSE=1>2:TRUE=1<2
10170 DIM LR$(500),LV$(2,500): 'These arrays hold cross references.
10180 '
10190 ' Now, clear the screen, print the title header, and prompt the
10200 ' user for the name of the source file.
10210 '
10220 CLS:PRINT "VXREF (Version 1.01.00) BASIC Program Cross Reference ";
10230 PRINT"Utility.":PRINT"For CodeWorks Magazine by J. Melvin ";
10240 PRINT"Jones.":PRINT
10250 PRINT :PRINT "SOURCE FILE: ";:LINE INPUT F$
10260 PRINT
10270 '
10280 ' Now, we can begin to process the file, after we have loaded
10290 ' our keyword table and opened the input file.
10300 '
10310 GOSUB 12870
10320 FB=1:GOSUB 11090
10330 PRINT"Processing file called ";F$;"."
10340 '
10350 ' Process the file, one line at a time, and list the source
10360 ' file to the line printer (in a paged format).
10370 '
```

```
10380 H$="PROGRAM SOURCE CODE LISTING FOR "+F$+".                    Page ####
   "
10390 PG=1
10400 LPRINT STRING$(8,13);:LPRINT USING H$;PG:LPRINT
10410 IF EOF(FB) THEN GOTO 10500
10420   GOSUB 10990: ' Get one line from the source file.
10430   LPRINT LEFT$(A$,LEN(A$)-1)
10440   LL=LL+1
10450   GOSUB 11610: ' Eliminate Lexical Constants.
10460   GOSUB 11500: ' Eliminate REMark statements.
10470   GOSUB 11720: ' Eliminate Delimiters.
10480   GOSUB 11830: ' Parse string.
10490 IF LL>=48 THEN LPRINT CHR$(12);:PG=PG+1:LL=0:GOTO 10400:ELSE GOTO 1
0410
10500 GOSUB 11160
10510 X$=""
10520 PRINT"Source file analysis complete."
10530 PRINT"Now printing Cross Reference Lists."
10540 '
10550 ' Now we can output the contents of the lists...
10560 '
10570 H$="CROSS REFERENCE LISTING FOR "+F$+".                        Page ####
   "
10580 GOSUB 11380
10590 LPRINT"LINE NUMBER REFERENCES: "
10600 LPRINT"========================= "
10610 LPRINT
10620 LPRINT"LINE NUMB                                          REFERENCED IN"
10630 LPRINT"---------                                          ------------"
10640 LPRINT
10650 LL=LL+6
10660 FOR I=1 TO NL
10670   T$=STR$(CVI(LEFT$(LR$(I),2))+32768!)
10680   FOR I1=3 TO LEN(LR$(I))-1 STEP 2
10690     X1$=STR$(CVI(MID$(LR$(I),I1,2))+32768!)
10700     X1$=SPACE$(9-LEN(X1$))+X1$
10710     IF I1=LEN(LR$(I))-1 THEN X1$=X1$+" ":ELSE X1$=X1$+","
10720     X$=X$+X1$
10730     IF LEN(X$)>60 THEN GOSUB 11230
10740   NEXT I1
10750   IF X$<>"" THEN GOSUB 11230
10760 NEXT I
10770 IF LL>40 THEN GOSUB 11380
10780 LPRINT
10790 LPRINT"VARIABLE REFERENCES: "
10800 LPRINT"====================== "
10810 LPRINT
10820 LPRINT"VARIABLE                                           REFERENCED IN"
10830 LPRINT"--------                                           ------------"
10840 LL=LL+7
10850 FOR I=1 TO NV
10860   T$=LV$(1,I)
10870   FOR I1=1 TO LEN(LV$(2,I))-1 STEP 2
10880     X1$=STR$(CVI(MID$(LV$(2,I),I1,2))+32768!)
```

```
10890      X1$=SPACE$(9-LEN(X1$))+X1$
10900      IF I1=LEN(LV$(2,I))-1 THEN X1$=X1$+" ":ELSE X1$=X1$+","
10910      X$=X$+X1$
10920      IF LEN(X$)>60 THEN GOSUB 11230
10930    NEXT I1
10940    IF X$<>""THEN GOSUB 11230
10950 NEXT I
10960 LPRINT CHR$(12);
10970 PRINT"JOB COMPLETE."
10980 END
10990 '-------------------------------------------------------
11000 '          SEQUENTIAL ASCII FILE READER SUBROUTINE
11010 ' This subroutine gets lines of text from a sequential ASCII
11020 ' file and returns the string, appended with a space, in A$.The
11030 ' buffer # of the file being accessed must be specified by FB.
11040 '-------------------------------------------------------
11050 LINE INPUT #FB,A$
11060 A$=A$+" "
11070 X=FRE(0)
11080 RETURN
11090 '-------------------------------------------------------
11100 '          SEQUENTIAL ASCII INPUT FILE INITIALIZER ROUTINE
11110 ' This subroutine opens an ASCII file as type "INPUT". The file
11120 ' to be opened must be specified in F$, the buffer number in FB
11130 '-------------------------------------------------------
11140 OPEN "I",FB,F$
11150 RETURN
11160 '-------------------------------------------------------
11170 '          FILE CLOSER SUBROUTINE
11180 ' This subroutine closes the file whose buffer number is
11190 ' specified by the value of FB.
11200 '-------------------------------------------------------
11210 CLOSE FB
11220 RETURN
11230 '-------------------------------------------------------
11240 '          LINE PRINTER DRIVER/FORMATTER SUBROUTINE
11250 ' This subroutine controls the report generated at the line printer
11260 ' by keeping track of the number of lines output, and paging
11270 ' accordingly. All output lines must be passed to this subroutine i
n
11280 ' X$ for output, with title item in T$, and page header in H$.
11290 '-------------------------------------------------------
11300 IF T$<>LT$ THEN LPRINT:LL=LL+1
11310 IF LL=>50 THEN GOSUB 11380
11320 IF LT$<>T$ THEN LPRINT T$;TAB(11);:LT$=T$:ELSE LPRINT TAB(11);
11330 LPRINT X$
11340 LL=LL+1
11350 IF LL=>50 THEN GOSUB 11380
11360 X$=""
11370 RETURN
11380 '-------------------------------------------------------
11390 '          PAGE/HEADER PRINTER INITIALIZER
11400 ' This subroutine is executed at the beginning of each new page of
11410 ' output and formats the top margin and header lines.
11420 '-------------------------------------------------------
11430 LPRINT CHR$(12);
```

```
11440 PG=PG+1
11450 LPRINT STRING$(8,13);
11460 LPRINT USING H$;PG
11470 LPRINT:LPRINT
11480 LL=3:LT$=""
11490 RETURN
11500 '-------------------------------------------------------
11510 '          REMARK ELIMINATOR
11520 ' This subroutine removes all text from A$ which is not part of
11530 ' the executable code of a BASIC program (i.e., Remarks) and
11540 ' replaces it with spaces.
11550 '-------------------------------------------------------
11560 A1=INSTR(A$," REM ")
11570 IF A1<1 THEN GOTO 11590
11580 MID$(A$,A1,LEN(A$)-A1+1)=SPACE$(LEN(A$)-A1+1)
11590 A1=INSTR(A$," '"):IF A1>0 THEN GOTO 11580
11600 RETURN
11610 '-------------------------------------------------------
11620 '        LEXICAL CONSTANT ELIMINATER SUBROUTINE
11630 ' This subroutine scans the text in A$ for lexical constants
11640 ' (string literals) and replaces them with blanks.
11650 '-------------------------------------------------------
11660 A1=INSTR(A$,CHR$(34))
11670 IF A1<1 THEN RETURN
11680   A2=INSTR(A1+1,A$,CHR$(34))
11690   IF A2<=0 THEN A2=LEN(A$): ' NOTE: UNMATCHED QUOTATIONS!!
11700   MID$(A$,A1,A2-A1+1)=SPACE$(A2-A1+1)
11710 GOTO 11660
11720 '-------------------------------------------------------
11730 '        IRRELEVANT DELIMITER ELIMINATER
11740 ' This subroutine scans the text in A$ for delimiters having
11750 ' no meaning to the token parser and replaces them with blanks.
11760 '-------------------------------------------------------
11770 ZZ$="&')*:;=-+,/?<>^"
11780 FOR I1=1 TO LEN(ZZ$)
11790   A1=INSTR(A$,MID$(ZZ$,I1,1))
11800   IF A1>0 THEN MID$(A$,A1,1)=" ":GOTO 11790
11810 NEXT I1
11820 RETURN
11830 '-------------------------------------------------------
11840 '          TOKEN PARSER
11850  ' This subroutine is the heart of the program. It scans the
11860  ' contents of A$ and determines the nature of each remaining
11870  ' item in the string. If the item is a basic keyword or a
11880  ' numeric constant, it is eliminated. Otherwise, it is passed
11890  ' to the appropriate logging subroutine. NOTE: L9 contains
11900  ' the line number of the line being evaluated.
11910 '-------------------------------------------------------
11920 SKIP=FALSE:S1=FALSE
11930 L9=VAL(A$):IF L9=0 THEN RETURN
11940 A9=INSTR(A$," "):IF A9<1 THEN RETURN
11950 FOR I=A9+1 TO LEN(A$)
11960   IF MID$(A$,I,1)=" " THEN SKIP=FALSE:GOTO 12210
11970   IF SKIP=TRUE THEN GOTO 12210
11980   IF VAL(MID$(A$,I))>0 AND S1=FALSE THEN SKIP=TRUE:GOTO 12210
```

```
11990    IF VAL(MID$(A$,I))=0 THEN GOTO 12020
12000    GOSUB 12230
12010    SKIP=TRUE:GOTO 12210
12020    IF MID$(A$,I,1)="#" THEN GOTO 12210
12030    IF MID$(A$,I,1)="0" THEN GOTO 12210
12040    '
12050    'If control passes here, the token is alphanumeric, so we
12060    'must determine if it is a BASIC keyword.
12070    '
12080    H=NK:L=1
12090    P=INT(H+L)/2
12100    BK$=BK$(P):IF RIGHT$(BK$,1)="#"THEN BK$=LEFT$(BK$,LEN(BK$)-1)
12110    IF BK$>MID$(A$,I,LEN(BK$)) THEN H=P:GOTO 12150
12120    IF BK$<MID$(A$,I,LEN(BK$)) THEN L=P:GOTO 12150
12130    SKIP=TRUE:IF RIGHT$(BK$(P),1)="#" THEN S1=TRUE:ELSE S1=FALSE
12140    GOTO 12210
12150    IF INT(H+L)/2<>P THEN GOTO 12090
12160    '
12170    'If control passes here token is a variable name, so pass it.
12180    '
12190    SKIP=TRUE:S1=FALSE
12200    GOSUB 12550
12210 NEXT I
12220 RETURN
12230 '-------------------------------------------------------------
12240 '              LINE NUMBER REFERENCE LIST BUILDER
12250 ' This subroutine records the line number references in the
12260 ' array LR$. The values are compressed into strings to allow
12270 ' more  efficient dynamic memory usage.
12280 '-------------------------------------------------------------
12290 H=NL:L=1:P=INT((H+L)/2)
12292 '
12294 ' Fix a problem in some BASIC interpreters ...
12296 '
12297 CW=I+1
12298 IF INSTR("0123456789",MID$(A$,CW,1))>0 THEN CW=CW+1:GOTO 12298
12300 K$=MKI$(VAL(MID$(A$,I,CW-I))-32768!)
12310 X$=MKI$(L9-32768!)
12320 IF NL=0 THEN NL=1:LR$(1)=K$+X$:GOTO 12530
12330 IF CVI(K$)<CVI(LEFT$(LR$(1),2)) THEN P=1:GOTO 12420
12340 IF CVI(K$)>CVI(LEFT$(LR$(NL),2)) THEN P=NL+1:GOTO 12510
12350 P=INT((H+L)/2)
12360 IF CVI(K$)=CVI(LEFT$(LR$(P),2)) THEN GOTO 12450
12370 IF CVI(LEFT$(LR$(P),2))>CVI(K$) THEN H=P:ELSE L=P
12380 IF P<>INT((H+L)/2) THEN GOTO 12350
12390 IF P-1=0 THEN GOTO 12410
12400 IF CVI(K$)<CVI(LEFT$(LR$(P-1),2)) THEN P=P-1:GOTO 12400
12410 IF CVI(K$)>CVI(LEFT$(LR$(P),2)) THEN P=P+1:GOTO 12410
12420 '
12430 ' Add the reference to the list.
12440 '
12450 IF CVI(K$)<>CVI(LEFT$(LR$(P),2)) THEN GOTO 12480
12460   IF RIGHT$(LR$(P),2)<>X$ THEN LR$(P)=LR$(P)+X$
12470   GOTO 12530
12480 FOR I0=NL TO P STEP -1
```

```
12490   LR$(I0+1)=LR$(I0)
12500 NEXT I0
12510 NL=NL+1
12520 LR$(P)=K$+X$
12530 X=FRE(0)
12540 RETURN
12550 '-------------------------------------------------------------
12560 '              VARIABLE REFERENCE LIST BUILDER
12570 ' This subroutine adds references to the variable reference
12580 ' list which is stored in array LV$. Again, string compression
12590 ' is used to save space.
12600 '-------------------------------------------------------------
12610 K$=MID$(A$,I,(INSTR(I+1,A$," ")-I))
12620 X$=MKI$(L9-32768!)
12630 H=NV:L=1
12640 IF NV=0 THEN NV=1:LV$(1,1)=K$:LV$(2,1)=X$:GOTO 12850
12650 IF K$<LV$(1,1) THEN P=1:GOTO 12730
12660 IF K$>LV$(1,NV) THEN P=NV+1:GOTO 12830
12670 P=INT((H+L)/2)
12680 IF K$=LV$(1,P) THEN GOTO 12760
12690 IF K$<LV$(1,P) THEN H=P:ELSE L=P
12700 IF INT((H+L)/2)<>P THEN GOTO 12670
12710 IF K$<LV$(1,P-1) THEN P=P-1:GOTO 12710
12720 IF K$>LV$(1,P) THEN P=P+1:GOTO 12720
12730 '
12740 ' Add the reference to the list.
12750 '
12760 IF K$<>LV$(1,P) THEN GOTO 12790
12770   IF RIGHT$(LV$(2,P),2)<>X$ THEN LV$(2,P)=LV$(2,P)+X$
12780   GOTO 12850
12790 FOR I0=NV TO P STEP -1
12800   LV$(1,I0+1)=LV$(1,I0)
12810   LV$(2,I0+1)=LV$(2,I0)
12820 NEXT I0
12830 NV=NV+1
12840 LV$(1,P)=K$:LV$(2,P)=X$
12850 X=FRE(0)
12860 RETURN
12870 '-------------------------------------------------------------
12880 '            BASIC KEYWORD DATABASE INITIALIZER
12890  ' This subroutine loads the list of basic keywords into the
12900  ' array BK$. The 1st element in data specifies the number of
12910  ' keywords listed. NOTE: Keywords MUST be in alphabetical
12920  ' order because a binary search is employed to optimize
12930  ' program execution speed in the token evaluation routine.
12940 '-------------------------------------------------------------
12950 READ NK
12960 DIM BK$(NK)
12970 FOR I1=1 TO NK
12980   READ BK$(I1)
12990 NEXT I1
13000 RETURN
13010 '
13020 '  BASIC Keyword Database (for TRSDOS 6.1.2, BASIC 1.01.01)
13030 '  NOTE: Keywords which may be followed by a line number or
```

```
13040 '  list of line numbers must be postfixed with the '#' char.
13050 '
13060 DATA 133,"ABS(","AND ","ASC(","ATN(","AUTO ","CALL ","CDBL("
13070 DATA "CHAIN ","CHR$(","CINT(","CLEAR ","CLOSE ","CLS ","COMMON "
13080 DATA "CONT ","COS(","CSNG(","CVD(","CVI(","CVS(","DATA ","DATE$ "
13090 DATA "DEF ","DEFDBL ","DEFINT ","DEFSNG ","DEFSTR ","DELETE #"
13100 DATA "DIM ","EDIT #","ELSE #","END ","EOF(","EQV ","ERASE "
13110 DATA "ERL ","ERR ","ERROR ","ERRS$ ","EXP(","FIELD ","FIX(","FN "
13120 DATA "FOR ","FRE(","GET ","GOSUB #","GOTO #","HEX$(","IF ","IMP "
13130 DATA "INKEY$(","INP(","INPUT ","INSTR(","INT(","KILL ","LEFT$("
13140 DATA "LEN(","LET ","LINE ","LIST #","LLIST #","LOAD ","LOC("
13150 DATA "LOF(","LOG(","LPOS(","LPRINT ","LSET ","MEM ","MERGE "
13160 DATA "MID$(","MKD$(","MKI$(","MKS$(","MOD ","NAME ","NEW "
13170 DATA "NEXT ","NOT ","OCT$(","ON ","OPEN ","OPTION ","OR ","OUT "
13180 DATA "PEEK(","POKE ","POS(","PRINT ","PUT ","RANDOM ","READ "
13190 DATA "RENUM #","RESTORE ","RESUME #","RETURN ","RIGHT$(","RND("
13200 DATA "ROW(","RSET ","RUN #","SAVE ","SGN(","SIN(","SOUND "
13210 DATA "SPACE$(","SPC(","SQR(","STEP ","STOP ","STR$(","STRING$("
13220 DATA "SWAP ","SYSTEM ","TAB(","TAN(","THEN #","TIME$ ","TO "
13230 DATA "TROFF ","TRON ","USING ","USR ","VAL(","VARPTR(","WAIT "
13240 DATA "WEND ","WHILE ","WIDTH ","WRITE ","XOR "
```

Programming Notes

Robert Hood of Bremerton, WA sent in a couple of interesting notes. One has to do with the INT function, which he says has an error which he found a way to correct. Try the following code and see how your computer handles it:

```
10 DATA 2.34,13.34,14.34,15.34
20 FOR J=1 TO 4
30  READ K
40  PRINT INT(K*100),
50 NEXT J
```

He got these answers:

2.33 13.33 14.33 15.33

To correct for this error, he changed the function to INT(K*100+.001).

Along those same lines, we found that BASIC running with MS-DOS has a problem with the value .09. Try this in command mode: A=.09:PRINT A. We get a number that goes into exponential notation. But A=.091 or .0899 will return the .091 or .0899. What's so special with .09? Also try adding 2.2 plus 2.2 plus 2.2. You may get something other than the 6.6 you expected.

Here is the other of Mr. Hood's ideas. Have you ever wanted to use INKEY$ to get more than one character from the keyboard? Here is a sample code that will allow you to get two digits. Note that if you wait too long before entering the second digit, only the first will be accepted. The wait time can be adjusted by changing the delay loop. Yes, INPUT$(X) could be used to input more than one digit, but in that case X is a fixed value. This method allows the choice of entering either one or two digits.

```
100 PRINT"INPUT A TWO DIGIT
NUMBER"
110 GOSUB 1000
120 PRINT W
130 END
1000 W$=INKEY$:IF W$= THEN 1000
1010 FOR J=1 TO 200
1020   W$=W$+INKEY$
1030 NEXT J
1040 W=VAL(W$):RETURN
```

We have been asked several times why we use LPRINT" " instead of simply LPRINT. It is because several printers require it, in particular some of the Centronics models. We happen to have an old work-horse Centronics 703 which does require the quote marks or it won't print the line feed. If your printer does not need them, simply leave them out.

Convert.Bas

Pick, choose and convert data formats

Staff Project. If we were always that smart we would do it right the first time. But we aren't — so here is a program that lets you change your mind about the format your data will have.

Convert.Bas is a program which allows you to change data statements into sequential files or convert sequential file information into data statements. A serendipitous by-product of these exchanges is that you can re-order, as well as selectively eliminate, data during the transfer. The program is actually two independent programs under one heading with menu selection.

But why would you want to do something like this in the first place? The reason this program came into being is as good an example as any. Let's see how it evolved.

We recently started work on a football prediction program. The obvious place to start was to get all the statistics for an entire season. We used Maker.Bas (from our very first issue) to enter the stats for 28 teams for the first eight weeks of the 1985-86 season. It made 224 data lines, each containing 24 data items. We had to include as much data as possible since we were going to check it to try and find which statistics had significance.

This large block of data was then merged with several different correlation programs and various tests were run. Those tests showed us that several items were insignificant and could be ignored. We found, for example, that time of possession had no measurable influence on the outcome of the game, even though the sportscasters make a big deal of it during games on television. Several other items could also be eliminated. We also found that it would be more convenient to have the remaining data in a different order. Someone suggested that it may even be easier to handle the data if it were in a sequential file. Convert.Bas was then born, and we could try it both ways without the drudgery of re-typing and checking all that data.

The program is not a simple type-in-and-run affair. It is not difficult to use, but there are two lines that need to be edited prior to running the program. Another caveat is that the number of items in each data line (or in each line of your sequential file) must be the same for all lines. The program easily handles strings, integers or both mixed. Even if you do not wish to have your data in statements (or in a sequential file), going through the process allows you to select data and order it as you wish.

Program Notes

The program is in two main sections. The first will convert data statements into a sequential file. It resides in lines 270 through 510. Let's say you have a different program which has data statements you wish to convert. First you load the other program and delete all the program lines except the data statements. Then renumber the data statement lines starting at 1000. Then save the data statements using the ASCII identifier (SAVE"filename.DAT",A). Then load this program and merge the data statements into it (MERGE"filename.DAT").

Now you need to decide, in line 460, how your data statements (and which ones) will appear in the sequential file. We only show line 460 with four items in ascending order. You can add or delete items in that line, as well as change the order of the items if you wish. It stands to reason that you cannot put more data items into your sequential file than exist in the data statement.

The prompts for this section of code ask for the number of data lines to be converted and how many items there are in each line. Enter the number of items actually in each line now, not the number you want to have after conversion. Lines 380 through 430 read the specified number of lines into array A$(I,J). Lines 440 through 480 open the file you named in line 360 and print the specified items to the file. You may want to check the article in Issue 4 if you are hazy on sequential files.

And back again

The other half of the program, from lines 530 through 840, will take data from a sequential file and make data statements from it. This section is a little trickier than the first. It needs to create actual line numbers and the word DATA for your data

lines. The prompts ask which file to read from, what line number your data statements will start with, how many data lines to make and how many items in each line. Based on this information, an array is set in line 600, and lines 610 through 680 read the file into array A$(I,J). Line 690 defines D$ as the word DATA with a space before and after it. Lines 700 through 730 build each data line. Each data line is held in array Z$(I), the first in Z$(1), the second in Z$(2) and so on. Z$(I) is made up first with the string value of LL (which you had already input in line 570) and is the beginning line number for the data statements. Next, we add D$ (the word DATA with the spaces around it) to Z$(I). Then we add the data items from the A$(I,X) array, along with explicit commas to separate the data items.

Line 740 simply prints the data lines on the screen for you to see before you give them a file name and save them. Before we leave here, we should add that line 720 increments your starting line number by two. After you have saved the data statements on diskette, you can always load them like any BASIC program, renumber them and save them back again. If you intend to merge them with another BASIC program later, be sure to save them with the ASCII identifier.

As in the first section of the program, line 710 can be extended or shortened to fit your needs. And, as before, you can change the order in which the data items appear in your data line.

General notes

You may well ask why lines 510 and 840 both say to RUN 100 rather than GOTO 140. It is because each option of the program stands alone and each must have dimensions for arrays set differently. After running each section of the program what needs to be done is done. If you go directly from option 1 to 2 you would get a "Duplicate Definition" error because of the dimension statements. The "RUN 100" in those two lines clears all dimensions and starts you from square one and avoids the error message.

Notice that the choice input in lines 240 through 260 avoids the use of the usual code to trap for inputs larger than, or smaller than, those allowed. In this case, it was more simply done by using line 260. Now, if X is not 1, 2 or 3, line 260 simply gets you back to re-display the heading and another try.

The data lines starting at line 1000 are for sample testing only. If you entered this program correctly, you should get what the sample figures with this article show.

You will find that making a sequential file from data statements happens rather quickly. On the other hand, creating data statements from a sequential file may take some time, especially when the file is medium to large. We created 225 data statements from a sequential file and it took about 20 minutes.

Now you can play "manipulation mania" as you wish. It may sound trivial, but it sure beats keying in data that has already been entered and checked. Add this one to your library of utilities. The odds are, sooner or later, that you will make good use of it. ■

```
100 REM * CONV.BAS * CONVERT DATA STATEMENTS TO SEQ FILE AND BACK *
110 REM * CODEWORKS MAGAZINE * 3838 S. WARNER ST. TACOMA,WA 98409
120 REM * SEE ISSUE 5 FOR FURTHER ENLIGHTENMENT
130 'CLEAR 10000: ' USE ONLY IF YOU NEED TO CLEAR STRING SPACE
140 CLS: ' CHANGE THIS CLEAR SCREEN COMMAND TO SUIT YOUR MACHINE
150 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
160 PRINT "          D A T A  C O N V E R T E R  P R O G R A M"
170 PRINT "     Convert Data statements to Sequential files and back"
180 PRINT STRING$(60,"-")
190 PRINT
200 PRINT TAB(5);"1) Convert Data statements to a Sequential file."
210 PRINT TAB(5);"2) Convert a Sequential file to Data statements."
220 PRINT TAB(5);"3) To quit."
230 PRINT
240 INPUT"Your choice";X
250 ON X GOTO 270,520,850
260 GOTO 140
270 CLS
```

```
280 PRINT"Convert Data statements to a sequential file."
290 PRINT
300 PRINT"Your Data statements must be merged with this program"
310 PRINT"starting at or above line 1000."
320 PRINT
330 PRINT"Before you run, adjust line 460 to fit your format."
340 INPUT"How many Data lines are to be converted";DL
350 INPUT"How many items does each line contain";DI
360 INPUT"What will you name your sequential file";F$
370 PRINT
380 DIM A$(DL,DI)
390 FOR I=1 TO DL
400   FOR J=1 TO DI
410     READ A$(I,J)
420   NEXT J
430 NEXT I
440 OPEN "O",1,F$
450 FOR I=1 TO DL
460   PRINT #1,A$(I,1);",";A$(I,2);",";A$(I,3);",";A$(I,4)
470 NEXT I
480 CLOSE 1
490 PRINT"Your data statements are now a sequential file called ";F$
500 INPUT"Press RETURN for menu";XX
510 RUN 100
520 CLS
530 PRINT"Convert a Sequential file to data statements."
540 PRINT
550 PRINT"Adjust line 710 to conform to your data line format."
560 INPUT"What is the name of your sequential file ";F$
570 INPUT"What line number should your data lines start with";LL
580 INPUT"How many data lines do you wish to make";DL
590 INPUT"How many data items in each line";DI
600 DIM A$(DL,DI),Z$(DL)
610 OPEN "I",1,F$
620 FOR I=1 TO DL
630  IF EOF(1) THEN GOTO 680
640   FOR J=1 TO DI
650     INPUT #1,A$(I,J)
660   NEXT J
670 NEXT I
680 CLOSE 1
690 D$=" DATA "
700 FOR I=1 TO DL
710   Z$(I)=STR$(LL)+D$+A$(I,1)+","+A$(I,2)+","+A$(I,3)+","+A$(I,4)
720   LL=LL+2
730 NEXT I
740 FOR I=1 TO DL:PRINT Z$(I):NEXT I
750 INPUT"What file name shall your data statements have ";F1$
760 OPEN "O",1,F1$
770 FOR I=1 TO DL
780   PRINT #1,Z$(I)
790 NEXT I
800 CLOSE 1
```

```
810 PRINT"Your data statements in file ";F1$;" may now be merged"
820 PRINT"into any other BASIC program."
830 INPUT"Press RETURN for menu";XX
840 RUN 100
850 CLS:PRINT"DONE":END
1000 DATA JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC
1002 DATA 1,2,3,4,5,6,7,8,9,10,11,12
1004 DATA ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE,TEN,ELEVEN,TWEL
VE
1006 DATA JOHN,PAUL,PETER,MARY,ELLEN,JO,OSCAR,BOB,ALBERT,EDWARD,HENRY,
STEVE
```

```
Convert Data statements to a sequential file.

Your Data statements must be merged with this program
starting at or above line 1000.

Before you run, adjust line 460 to fit your format.
How many Data lines are to be converted? 4
How many items does each line contain? 12
What will you name your sequential file? SAMPLE.TXT

Your data statements are now a sequential file called SAMPLE.TXT
Press RETURN for menu?
Break in 500
Ok

B>TYPE SAMPLE.TXT
JAN,FEB,MAR,APR
1,2,3,4
ONE,TWO,THREE,FOUR
JOHN,PAUL,PETER,MARY
```

Figure 1. This sample shows how a portion of the data statements can be changed to a sequential file.

```
Convert a Sequential file to data statements.

Adjust line 710 to conform to your data line format.
What is the name of your sequential file ? SAMPLE.TXT
What line number should your data lines start with? 1000
How many data lines do you wish to make? 4
How many data items in each line? 4
 1000 DATA JAN,FEB,MAR,APR
 1002 DATA 1,2,3,4
 1004 DATA ONE,TWO,THREE,FOUR
 1006 DATA JOHN,PAUL,PETER,MARY
What file name shall your data statements have ?
```

Figure 2. This segment shows data statements which were created from the sequential file shown in Figure 1.

```
710   Z$(I)=STR$(LL)+D$+A$(I,4)+","+A$(I,3)+","+A$(I,2)+","+A$(I,1)

Convert a Sequential file to data statements.

Adjust line 710 to conform to your data line format.
What is the name of your sequential file ? SAMPLE.TXT
What line number should your data lines start with? 1000
How many data lines do you wish to make? 4
How many data items in each line? 4
 1000 DATA APR,MAR,FEB,JAN
 1002 DATA 4,3,2,1
 1004 DATA FOUR,THREE,TWO,ONE
 1006 DATA MARY,PETER,PAUL,JOHN
What file name shall your data statements have ?
```

Figure 3. This does the same as in Figure 2, but before running the program, line 710 was changed as shown. The data in the statements is now in reverse order of those in Figure 2.

Download

What's Happening on the Download

There are two sections to the CodeWorks download. The Demo section and the Subscriber section.

The Demo Section

This section is intended to show non-subscribers what the download system is all about. When you call the system and it answers, it will ask for a User Name. The Demo section will accept almost anything as a user name. There is no password required here.

After entering a name the system will present you with a Message of the Day. After you have read the message you will be presented with a menu of choices. Choosing HELP will give you more information about how to use the system.

There are currently five programs available on the Demo section available for download. They are intended to give an example of the type of programming that can be found in the magazine. These programs will be replaced from time to time, some may be removed, new programs may be added.

The non-subscriber may request a sample issue by using the Send Me More option of the menu. In this case, he leaves his name and address on the system and a sample copy of the magazine will be sent to him.

The Signup option gives a place for anyone to leave his name, address and credit card number to enter a subscription to CodeWorks.

The Comments section is provided so that notes about the system, or other comments may be left. We check the system in the morning of each working day and fill requests and note the comments.

The comments are very helpful to us. It gives us a window through which to view the system from the other side. For example, one comment saying that it was difficult to do some particular thing would simply be noted. Several such comments, however, could easily point out a problem with the system. Through the comments we have already noted some difficulty in getting on the system when the telephone signal is routed through several exchanges. Every now and then the satellites carrying telephone signals do funny things too. This was noted by subscribers in Hawaii and Alaska.

The Terminal Setup option is used to tell our system something about your system so that the information we send will be formatted properly for your screen, etc. We cannot permanently record your terminal setup from the Demo section; however, subscribers can make their terminal characteristics permanent on our end.

The Demo section is designed, quite frankly, to encourage people to subscribe to CodeWorks. Giving the download telephone number to your computing friends is encouraged. You might also tell them what the protocol is (300,n,8,1).

The Subscriber Section

Logging on as a subscriber is not difficult, although it is a little more involved than that of the Demo section. When asked for a login name, a subscriber should enter his last name followed by the numbers on his address label on the magazine. The number is found near the upper right of the label. The name may be entered in lower case, upper case or mixed. The number should follow the name immediately, with or without a space between it and the name. Our system has your name and number stored in it. When you enter your name and number, the system quickly checks the list it has to look for a match. If no match is found it assumes you are not a subscriber and shifts you to the Demo section.

If a match is found, the system then welcomes you by name to the system and asks you to enter your password. If this is the first time you have logged on to the system, you will be asked to enter a password of your choice (6 or more characters.) This password is then encoded and attached to your name and number in our system. The next time you log on then, the system will recognize your name, number and password, and allow you access to all the programs from all issues to date.

You may also set your terminal characteristics and make them permanent so that the next time you use the system, it will remember who you are and set your terminal properly. If you should forget your password we cannot tell you what it was because we don't know. Should this happen, give us your name and subscriber number and we

Continued on back cover —

Where is Issue 1?

We have been receiving requests for Issue 1 at an increasing rate. Here is what happened. Last September we issued a promotional issue with the September/October 1985 date. It had no number. We then numbered the following issues 2, 3, etc.

We were still promoting various lists in February and March 1986 when we ran out of Issue 1. So, we made some minor adjustments to it and re-printed it as the Special Sampler Issue in March 1986.

Those who received this Special Sampler Issue apparently think they have missed Issue 1. That is not the case; the Special Sampler Issue *is* number 1. It is a free, additional issue, that come in addition to the regular subscription.

There were two programs in the original Issue 1 that are not in the Special Sampler Issue. One of them, PLOT, received so little attention that we didn't feel we should reprint it again. The other program, WRITER, was unique to 80-column screens with 24 lines and we left it out because of its non-applicability to many readers. Both these programs are still available on the download under the Issue 1 menu.

For those of you who do not use the download we are considering supplying all CodeWorks programs on diskette at the end of the subscription year. There would be a charge for this diskette. If you are interested in the diskette idea, drop us a card and let us know what format you need for your computer. We can make formats for IBM-PC and compatibles, TRS-80 Models I/III/IV, Kaypro (CP/M) and if necessary, some 8 inch diskette formats for CP/M, and Tandy Models II/12/16. By the end of the subscription year (in Sep/Oct 86) we expect there will be between 25 and 30 programs available for the diskette. (The diskette method may even be cheaper than the long distance charges for the download.)

Subscription ORDER FORM

586

Computer type:

Do you have a modem?
If so, what baud rate?

Comments:

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

☐ **Check or MO enclosed.**
☐ **Charge to my VISA/MasterCard #_____ Exp date_____**

Please Print clearly:

Name _____

Address _____

City _____ **State** _____ **Zip** _____

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

Charge card orders may be called in (206) 475-2219 between 9 AM and 4 PM weekdays, Pacific time. Sorry, no "bill me" orders.

— continued from page 38

can reset you in the system. What that does is clear your old password and terminal characteristics, and will treat you as a first-time user next time you log on.

Don't forget to prepare your system to accept the information it will download to you. Your terminal program must be able to save whatever arrives in your buffer to your diskette. Sometimes, it may be necessary for you to use a text editor to clean up some of the extraneous characters at the beginning or end of a downloaded program before you can run it. You can compare the downloaded program to the listing in the magazine if you are uncertain about what may be extra characters.

Once on the system as a subscriber you can select any issue from the menu and download any or all the programs from that issue. Make sure you have enough disk space to receive them. You may also need to reset your buffer between successive downloads to clear out anything left over from the last transmission. The reason we say that is because we have seen the same person download the same program as many as four or five times. From that, we can only assume that there was noise in the transmission or that the computer at the other end was not prepared to receive the program.

(206) 475-2356

300 baud, No parity

8 bits, 1 stop bit

full duplex

We are working on an automatic 300/1200 baud operation. When this is fully implemented, we will leave a message in the Message of the Day. When it works, the system will automatically answer you in the baud rate you sign on with. This should help cut costs for long distance charges, especially from the East coast and Midwest. In response to many requests, we are also trying to implement an abort feature so that if you want to terminate transmission for any reason in the middle of a download you can, without the need to disconnect and re-dial.

The system has been running continuously since January 1986, when we had a lightning storm that took out our hard drive. The system security has yet to be breached, although we have seen many attempts to do so.

These comments have all been presented because of many requests on the system itself. One last item. Please do not turn echo on at your end. It causes an endless loop wherein the system receives back its own query as a response to your input. We hope this helps you get more efficient and cheaper use of the CodeWorks download. ■

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

Address correction requested

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

CodeWorks

Issue 6 July/August 1986

CONTENTS

CodeWorks

Issue 6 **July/August 1986**

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

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

300 Baud, 8 bits, no parity, 1 stop bit

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

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

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

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

Editor's Notes

A couple of issues ago a request was made for a way to list programs in double space. Terry Dettmann got to work on it and the results of it are in this issue. Not only does it allow single or double spacing, but it formats the page and prints a heading with page numbers as well. One of the other refinements of the program seemed very worthwhile for programs listed in *CodeWorks* and so we are listing our programs with it.

That refinement was the ability to indent wrap-around lines, especially when the original line was already indented, as inside a For...Next loop. This makes for a much more readable page of code. Of course, when you type the programs, you need not allow for the extra spaces in the wrap-around.

Since it will be mid-summer when this issue arrives we have left the heavy explanations off of our major program. It's called NETWORK, and is an adaptation of a public domain program called Startraders. We have, however, included easy instructions so that you can make the program over to your own liking. Although I have never been too hung up on games, this one and a couple of others have been favorites for a long time. In fact, this is the first game we have published so far in the magazine, and we hope you agree that it is a fun game to play.

Based on many requests, both on the download and by mail and telephone, we are starting a series on random file techniques in this issue. File handling has always been one of the more important parts of computing, but somehow, random files have been tougher to master than other types. We intend to make the series both informative and useful, and our goal is to work towards a random file version of Card.Bas. Judging from your response, Card.Bas was one of the more popular programs we have published to date. There have been many requests to make it handle larger files, even though it was originally designed to replace a simple 3 x 5 card index file. By the time we get through with the random file series, it should be a functional database manager in its own right. Then, if all works well, we will create a report generator for it. Terry suggested we call it "Report.Card," and why not?

Since early on, we have said that the programs we publish are in public domain and that you can do with them what you want. We recently saw a version of one of our programs in another magazine. It had a subtle name change and was gussied up a bit. We applaud the enterprising author who sold that version to the other magazine. But he (or the other magazine) failed to acknowledge the origin of the program. For shame!

Can you believe it? It's almost football season again. In our next issue we plan to have an NFL Football Oracle program. We have been working with previous year's data and have it predicting the entire season at about 68%. As you might expect, we have found that it is virtually impossible to predict a true upset. There is simply nothing in the statistics that serves as a leading indicator of one. For the 1985-86 season, we had some weeks where it picked 12 out of 14 games. The worst was week seven (in 1985) which was a week of terrible upsets and we only picked seven of 14. In addition to picking the winner and a probable point spread, it shows relative indicators on the strength of both the offense and defense for all the teams, and other interesting statistics as well. In case our issue for Sep/Oct does not arrive until the first week in September, keep the stats printed in your local newspaper for games played before we arrive. We are anxious to see what the program will do with this year's schedule.

All in all, we hope you enjoy our mid-summer madness. We certainly had a good time putting it together.

Irv

Forum

An Open Forum for Questions & Comments

First I want to tell you that you are doing one fine job. There is a real need for a publication devoted to BASIC. It can't be beat for non-professional programmers. Cross your heart and hope to die that you won't ever publish a Pascal program; that can lead to Forth and Fivth and straight to eternal damnation. Also I think the Download is great and a breeze to use. Never mind the fancy error correcting systems; I never get past the place where you have to select between half a dozen different protocols named in a secret code.

I did have a problem with the MCI long distance circuit which had such a weak signal that your carrier wasn't readable. Has anyone else reported this? *(Yes, in a few isolated cases. Ed.)*

I have a conversion of the WOOD program (issue 3) to operate on fabric (anything you buy by the yard.) I don't really expect that my wife (a mild compuphobe) will use it for sewing materials but it is interesting...

...Meanwhile, I was intrigued by the comment in *CodeWorks* that nothing disastrous can really happen if you jump out of a FOR...NEXT loop with a GOTO. That had been my impression from experience also. I did a little experiment which may interest you. The supposed problem is that there is a stack where addresses and other goodies for FOR...NEXT and GOSUB are stored. One could have a real problem if things were repetitively pushed on that stack and never popped off. This should show up as a progressive disappearance of available memory. Now refer to the program.

In line 20 after setting the only variables, PRINT MEM tells the starting memory available before any FOR...NEXT loop is set up. Then for 100 times, execution jumps out of the loop to line 20. The first time you see a loss of 17 bytes of memory to set up the FOR...NEXT, as expected. However, there is no additional loss of memory as you repetitively jump out and re-enter the loop. Apparently, Microsoft has idiot-proofed this version of BASIC somehow. Perhaps other BASIC's are different; the program in lines 10-80 should check it out for users of other systems (mine is a TRS-80 Model III.)

```
10 ' TEST JUMP OUT WITH GOTO
15 J=0:A=0
20 PRINT MEM;
30 FOR J=1 TO 2
40 A=A+1
50 IF A>100 THEN 60
55 GOTO 20
60 NEXT J
70 PRINT MEM
80 END
```

Also of interest is the situation with programs that set up a GOSUB but jump out of the subroutine without hitting the RETURN. This is a different story as can be seen by running program lines 100-180. Referring to the program and the screen printout should make the situation obvious. Note that only five bytes are required to set up the GOSUB, but **WOW**, it eats up memory like a ravenous crocodile. Every repeat eats another five bytes. Needless to say, you don't get your memory back if you finally hit the RETURN one time.

I feel that this program enabled me to have confidence in what BASIC will do. Splurge 17 bytes on a jump out of a FOR...NEXT if you feel like it. But watch out for those GOSUB's that really should have been GOTO's.

```
110 'TEST OF GOSUB WITH RETURN
115 A=0
120 PRINT MEM;
130 GOSUB 150:PRINT MEM;:END
140 'OTHER LINES COULD BE HERE
150 A=A+1
160 IF A>100 THEN 180
170 GOTO 120
180 RETURN
```

R. R. Keegan
Fayetteville, AR

Thank you for your interesting observations, but Fivth?

Thanks for sending me the (sample copy.) I think you have a great idea for those who are interested in writing their own programs, and quite a relief from (other magazines) which are mostly ads, and a lot of the material assumes that

we know everything about everything!

I had a lot of fun putting your Calendar program into my computer. As you might expect I had to run down quite a few syntax errors in my typing, plus making the proper choices to make it run on my machine and printer.

Originally I typed lines 680-690 exactly as your sample and got the results that you did. If you will examine your printout, you will note that each of the months from January through September start with their first letter over the "N" in MON. However, October, November and December start over the space following "MON"! I tried editing line 690 and matters got worse when I got down to the line feed. Finally when I got to the MAY ", I hacked the rest of the line, then typed the following months, allowing nine spaces for each month starting with the first letter of each month. It all seemed to work ok!

Edson B. Snow, Col AUS-Ret.
Pompano Beach, FL

Your printout looks great. We missed that spacing the very first time we printed it. Then, wouldn't you know it, it was copied intact into the sampler issue.

I am using a C.Itoh 1550 printer with my Columbia PC and have problems with graphic screen dumps. I have several programs to print graphic screens and they all give the same result. The problem is a series of white lines across the print of the graphic screen dump. It would appear to me that one of the wires on the printhead is not printing. At first I thought there was a printer problem, but I can dismiss this, since the printer is printing the true descenders on all letters. Also with a dot graphics command I can get all of the wires to print.

I was told by the author of one graphics dump program that there is a problem with the BIOS of the Columbia that causes this. I have written to the Columbia Data Products people - but no answer. I would like to find someone with a solution.

Donald M. Dealy
South Attleboro, MA

We don't have an answer for you. But here are some things to consider: Does it happen on every line or once or twice per page? How wide is the little white line? Some of these new graphics printers have a paper pull-down of as little as 1/144 of an inch. If the line is really narrow and happens at an interval that corresponds to the circumference of your platen, your platen could have a flat spot along its length. Are there setup switches on your printer and if so, are they set correctly? It would have helped to see some sample output. I would worry more about the pull-down and the system driving the printer than the printhead. As you said, all the wires in the printhead are apparently printing.

...I'm just getting around to keying in CAL.Bas from your sample issue. The history of calendars is most interesting, and while your comments on the limitations of year 3999 are apologetic, I seriously doubt that problem will be a major issue with most of us.

I did have a small problem with the heading display of the year, so I simply re-wrote that section of the program. My version prints the year double wide...Anyway, rather than try to trace down the flow of your program, I just re-wrote it and, of course, I like my version better. Keep up the good work!

```
770 FOR L=1 TO 7
780 READ ST$
790 FOR J=1 TO 4
800 ND$(J)=MID$(ST$,DG(J)*6+2,5)
810 ND$=ND$+ND$(J)+" "
820 NEXT J
830 '
840 LE=LEN(ND$):LPRINT TAB(18);:
    FOR J=1 TO LE:IF MID$(ND$,J,
    1)>" "THEN LPRINT"##";ELSE
    LPRINT"  ";
850 NEXT J:LPRINT" "
860 ND$="":NEXT L
870 RETURN
```

Dexter Walker
Birmingham, AL

We like yours better too.

Since you mentioned that A=B=C is a legal construct, I have run across it in a program called "Racetrack" in a book by Stiegler and Hansen, called *Programming Languages: featuring IBM PC and compatibles.* (#5 in the Pournelle Users Guide series.) I've entered the program but am stuck by an error message "Illegal function call" on one of those lines: 28530 IF ONMAP THEN HITFENSE=(MID$(MAP$(YNOW%),XNOW%,1) = FENSE$:FINISHED= (MID$(YNOW%), XNOW%,1)=FINISHLINE$ Can you give me any other construction to replace them?

Allan W. Wardell, O.D.
Providence, RI

IF CODE$ = THAT BAD$ THEN I$ = (GIVEUP$)*2 *Seriously though, the first thing we*

would jump on is that "IF ONMAP" because it looks like a reserved word, even though that would give a syntax error and not an illegal function call Without seeing the rest of the code (and somehow we really don't want to) it's very difficult to determine what is supposed to be going on here.

...I am wondering how the programmer decides which letters and/or combination of letters to use when constructing programs. Is there a dictionary of these key letters such as there is for keywords?...

James L. Lopez
Pasadena, TX

No, there is no dictionary of key letters. You can assign them starting at A and ending at Z or if that isn't enough, start with AA and end with ZZ. Some computers allow more than two characters for a variable name. That helps make the variable names more descriptive of what they stand for (see the above letter, for example.) As a holdover from FORTRAN days, most programmers use the letters I, J, K and L for loop counters. You can also use a letter and a number, as in A9, but ***not*** *a number then a letter since variable names must always start with a letter, otherwise the computer wouldn't know if it were dealing with a variable or an integer. When you plan a program it is best to jot down groups of variables you will be using. Make all those that have a similar function, for example, start with the same letter. That way you can easily track through the program flow knowing what each is supposed to do. If your computer can use longer variable names you can be more descriptive, but watch out for embedded keywords within your variable names.*

...You asked about the game Star Traders. I have one by that name we enjoy playing very much. It came on Disk 45 of the Public Domain Software. I am enclosing a printout of the first few lines of the program which does give the author's name...

R. Oberdorfer
Newport,WA

Thanks Richard, but it says "Modified for ALTAIR BASIC 4.0 by S.J.Singer", and we still don't know if he is the author or not. Anyway, the program is in this issue. We have modified it further and call it NETWORK. Incidentally, did you know that MITS ALTAIR BASIC was written by Bill Gates and Paul Allen (who later formed Microsoft)?

I entered your Card.Bas program into my Kaypro CP/M computer and it works well. I have made revisions to make it more adaptable to my needs. I wrote a BASIC word processor program which I am using to write this, and have the word processor and card file on the same disk and can run either program from the other. The card file is just what I have been looking for.

Bill Heffley
Lantana, FL

We are glad you like it. It seems that more Kaypro users (both CP/M and MS-DOS) are finding our material useful.

In the past I have subscribed to several of the computer (magazines) available. Some I have found to be informative. When I first received your free sample issue I placed it in a drawer for 4 to 5 weeks without looking at it, believing it to be just another computer book. When I finally read some of the articles and put some of the programs on disk and ran them, I was quite pleased. Your article on SEARCH has been used by me on many occasions to search for strings, etc. I also enjoy running Wood.Bas, Card File and Payroll.Bas.

In running some of the programs, and certainly there are many others out there doing the same, I look for ways to embellish on a program, such as Allie Peed from New York did on Card.Bas.

In running Payroll.Bas, you note that there are no trailing zeros. In playing with the program I used the LPRINT USING statement and was able to eliminate most of the program lines that would call for them.

One last comment. Some computer magazines have several programs each month. However, many are written for (computers using PEEK and POKE.) I have a Tandy Model 12. Some of these programs appear to be interesting until I see a PEEK or a POKE statement. Do you know of any possible way to overcome this, by a patch or whatever?

Howard M. Cruff, Jr
Attleboro, MA

You are doing exactly what we expect most of our readers do, which is to take our code and make it do what you want it to do. As for putting our magazine in the drawer for four to five weeks: We would probably have a tough time explaining to the direct mail experts that we are still getting orders (I am writing this on the 1st of June) with a direct mail piece sent out last August! And some wag once said you were supposed to sell the sizzle, not the steak. Perhaps you are only supposed to do that when your product isn't as good as you say it is? Whatever. It's working, thank you. Now about your Model 12: The Tandy II, 12 and 16 BASICs do not have PEEK or POKE. We once did minor

surgery on a Model II which took out the commands HEX$ and OCT$ and put in the PEEK and POKE. But then you have a new problem. The PEEKs and POKEs printed for other computers are specific to those machines. The same PEEK (number) or POKE (number) will not necessarily work on yours. You need to know ***what*** *the PEEK or POKE is supposed to do on the other machine, and then find out what locations these same functions take place at on yours. Fortunately, some of those programs using these commands only use them to lock out the BREAK key or to test to see if the printer is ready or some such. If you can tell from the rest of the code that this is the case you can simply ignore those commands and get away clean. If that is not the case you would need to know* ***both*** *machines intimately at machine level to find out what to do. Not much help. If you really want to adapt the program to your machine, write the author of the program and ask him/her what they are trying to do. Knowing that, you can sometimes find a way around the PEEKs and POKEs. We try to avoid the use of these commands even though they are on most of the machines we have. In this issue though, there is a short demo program in Beginning BASIC that uses PEEK. In this case, we couldn't think of any possible way to get around using it.*

...thank you for your help in getting the Card.Bas/Check.Bas programs running for me...I called you and after some discussion we found the problem. I now have what I consider a "Great" program that I use regularly, and the best part is that it's FREE...

A. Vincent DiVirgilio
North Tonawanda, NY

You are welcome, but the program isn't free. When you subscribe to the magazine you get somewhere between 25 and 30 such programs, so the cost for that one would be just under one dollar. Such a deal!

I enjoy *CodeWorks* very much. The programs are great. I especially like the payroll program in Issue 4... I hope we can get an inventory program for a small business soon. Keep 'em flying!

R. Barden
Wood River Jct., RI

Hang in there, it's still in the works.

I think *CodeWorks* is a great magazine. It's too bad I cannot reap the benefits since I own a TI 99/4A. I think for all concerned, that it would be advantageous to compile and print a command or instruction list of all the different BASICs for cross reference, in an issue of your magazine.

In Issue 5, I agree with Mr. Overton (see Forum, Issue 5). I hope others show interest so you will change your format to possibly add an addendum to each listing for the "not-BASIC, BASIC" machines. Again, I like the magazine but I can't spend all my time converting a program not knowing how...

James Bowe
Canton, OH

We do not anticipate a format change in ***CodeWorks****. Unfortunately, Texas Instruments decided to march off to a totally different drummer. Their BASIC, even though written by Microsoft (to TI specifications), is so different that we don't even pretend to cover it in this magazine. As for a cross reference, try Dr. David Lien's book* ***The BASIC Handbook****, 2nd Edition, ISBN 0-932760-05-8, Compusoft Publishing, San Diego, CA. It should be available at most bookstores which deal in computer-related books (B. Dalton, Waldenbooks, etc.) for about $19.95*

We are out of space again, and there is still a stack of good letters. Well, more next time. Thanks again for the input. - Irv

"Somebody stole our security program."

Random Files

The 1st of a series on how and why

Terry R. Dettmann, Associate Editor. In this series on random files we will explore the why's and how's from the ground up. The demo program accompanying this article should be a good first experience for those who have so far avoided random access because of the complexity. Our goal is to work towards a mini-database program and a report generator to go along with it.

Many people who play with programming on the side shy away from random files because of the difficulty of making them work correctly. Sequential files are much easier since we can simply read them into a program and process them on a line by line basis.

Random files can be used like that too, but working with them directly gives you the power to do things with your programs you might consider fantastic. How about a file that spans more than one floppy disk? How about nearly instantaneous data retrieval? These are only samples of what you can do with random files.

An example of where random files can be used is the typical business inventory system. Generally, the businessman will want to look at the status of any item in the inventory at any time. If he wants to look at item 4965, he does not want to wait for the system to read in the first 4964 items from the disk first.

With random files, you can go directly to item 4965 in the same amount of time it would take to get to item 1. Even on older computers this seems instantaneous. Random access files (sometimes called *Direct Access*) work differently than sequential files. We will need to learn some new programming techniques to handle them.

This article will go into the basic details of random files and present a sample program to show their operation. This should give you some insight into how you can apply them to your own programming projects.

Working with Random Files

We will be introducing seven commands (five of them new) when working with random files. They are:

OPEN - opens a buffer between memory and your diskette.
CLOSE - closes the buffer between memory and your diskette.
FIELD - formats the file buffer (tells what goes where in the record.)
GET - reads information from the diskette into the buffer.
PUT - writes the information in the buffer to the diskette.
LSET - left- justifies the information in a *field* within the buffer record.
RSET - right-justifies the information in a *field* within the buffer record.

We also have to become acquainted with some special functions in BASIC that are used to convert numbers so that they can be used with random files (all information in random files must be string form.) These are:

To the disk via the buffer - MKI$, MKS$ and MKD$, and from the disk via the buffer - CVI, CVS and CVD.

Before we get into these commands, let's look at how a random file actually works.

Disk space is set up as a series of "cubbyholes", each representing one physical record on the diskette. That is, the size that the system will handle in one "read" from the disk. This size is fixed at some number of bytes. CP/M single density systems would handle 128 bytes per read. Sizes such as 256 bytes, 512 bytes and 1024 bytes are common as are many other sizes. On MS- DOS systems, the default disk access size is set at 128 bytes, (but can be specified in the range from 1 byte to 32768 bytes.)

Once we know the size of the record on diskette, we can get any record there by its address. Let's take the example we mentioned earlier, an inventory system. Assume we want the inventory stored by part number with part numbers between 1 and 300. By using the inventory part number as the address of the item, we can get to any item directly with random access techniques.

Random Access Commands

In order to understand the random access inventory, we need to understand that each inventory item as a "record". A record is simply a group of logically related information. Imagine the record is a file folder labeled with that item's inventory number and stored in a file cabinet in order of inventory number. We can pick out any item by its number quickly by looking at the folder. The file cabinet corresponds to the random access file in which the records are stored.

We will store all the information in the record as string variables. We will want to know the inventory number (its address in storage), its name, price, cost, amount on hand, who the supplier is and the reorder level. We could store a lot more but this wil do for the moment. Table 1 shows the expected size of string fields needed to store this information.

Table 1

Inventory Fields

Item	Size	Field Name
Name	20	NM$
Price	8	PR$
Cost	8	CS$
On hand	4	OH$
Supplier	20	SP$
Reorder level	4	RL$
Total	64	
Comment	190	
Record Size	254	

Why these fields and these sizes? They are a best guess for a simple inventory based on experience. Every project is a little different so the numbers could vary depending on what you are trying to do.

Notice that we have included a rather large comment field that can hold any information of interest. For the moment, this will serve to illustrate how such a file is set up. We can make this whole scheme more efficient depending on the computer and operating system we are using.

Now that we know what we want to store in the file, how do we deal with it? For this, we will need to understand how BASIC deals with files.

OPEN/CLOSE

The OPEN statement tells the BASIC interpreter that it is to get ready to access a particular file. It specifies the mode of access and assigns a file buffer for the system to use in transferring information to and from the diskette. The form of the command is:

```
OPEN "file mode","buffer number","file name","record size"
```

The arguments to the command start with the file mode. For random access files, we use the file mode "R" (on some systems "D") to indicate that this file is to be used in "Random" or "Direct" access mode. In this mode, we are allowed to read from, or write to, the file at any time.

The buffer number is a memory area associated with the file. BASIC allows the use of buffers numbered from 1 through 16, however, not all of them are always available. When we start BASIC from the operating system level, we usually have three buffers available unless we specify more or less.

The filename parameter tells the operating system what file to use with the file buffer. If the file does not exist, it will be created on the diskette as a random access file.

The record size is the number of bytes per record in the file. In our example, record size is 254. If we would eliminate the comment field, the record size would be 64.

FIELD

The FIELD statement lays out the exact distribution of the items in each record. We need to specify *where* each item will go and *how many spaces* it will take up. For our file, we can write the FIELD statement like this:

```
FIELD #1,20 AS NM$,8 AS PR$,8 AS CS$,4 AS OH$,20 AS SP$,4 AS RL$,190 AS CM$
```

This assigns variable names to fields in Input/Output Buffer #1 for each of the items in our inventory according to the field name assignments in Table 1. The statement first identifies which file buffer it is talking about (#1)

and then assigns fields to each variable: 20 bytes for NM$ for the item name, eight bytes to PR$ for the item prices, etc.

After a file is opened for random access, we FIELD it with the field statement for the file.

Putting Information into the File

Once the file buffer is fielded, we have to put information into the buffer. This is done with LSET and RSET instructions. They assign information to the field variables that we assigned with the FIELD statement. LSET puts the information into the buffer left-justified and RSET puts it in right-justified. It is important to remember that if the string to be put into the field is too long, then LSET and RSET will drop extra characters from the *right* of the string. Once the information is all in the buffer, we write it to diskette. This is done with the PUT instruction.

PUT/GET

PUT is one of two commands we use to actually move things to and from the diskette file. PUT takes anything in the file buffer and places it in the designated record on the diskette. For example, PUT 1,1 will put the current file buffer into record number 1. PUT 1,IN will put the file buffer into the record specified by the variable IN. The disk operating system (DOS) can go directly to the record on diskette.

To retrieve the record for use, we use the GET statement the same way we used PUT. We specify the number of the file and the number of the record to read from the diskette and put into the buffer. GET 1,1 will get record number 1 from file number 1. GET 1,N will get the record specified by the value of the variable N into buffer number 1.

When we get a record, we don't change a thing on the diskette. We just make a copy of it in the memory buffer that we can use to fill our program variables with.

After we GET a record, we can PRINT the values of the field variables, and even use them to write assignment statements. We cannot use them on the left side of an equal sign however. With our previous FIELD statement, if we wrote: NM$ = "Terry Dettmann" without LSET or RSET, then NM$ would become a normal string variable and no longer a FIELD variable. But we could use it like this: NAME$(I) = NM$. In this case we are assigning the value of the variable NAME$(I) to be whatever is in the field NM$. In this way can transfer information out of the buffer for use.

The listing with this article is a short random access file program to work with. It allows you to enter 10 items in a file, and then read them back one by one and compare them to what you had input to start with. Try it.

What about End of File?

In a random access file, the system takes care of the end of file automatically. If you PUT to a record number higher than the current end of file, the system assigns enough space to handle all the record numbers up through the record number you asked for. Those "in between" records are created with nothing in them (except what may have been on the disk before, which will probably be meaningless.) Once those records are created the system will not release them unless you delete the entire file.

To find the highest record number in a random access file, the function LOF(N) is provided in BASIC. It returns the highest record number of the file associated with buffer number N. It really doesn't matter whether the file is sequential or random, LOF works just the same, but the number is only meaningful with random files.

The function of EOF(N) does not work at all with random access files. It gives you no indication of whether the end of file has been reached with random access files. If you try to GET a record past the end of the file, all that will happen is that your program will stop with an error. Try it with the sample program by trying to GET record 100.

Numbers

What do we do about numbers in a random file? We have already stored them as strings. However, this can be wasteful of space. If we want to store a number like 30000 as a string we need five bytes, but as an integer it only takes two bytes in memory. Can we save it in only two bytes in the file?

The answer, of course, is yes. But we have to be tricky about it. The reason is that everything in the file has to be stored as if it were ASCII coded. This is fine for strings, but not for numbers. Unless we want to convert our numbers to strings for storage, we need to do something else.

BASIC gives us three functions to convert numbers to equivalent length strings and three more to convert them back again. They are listed here (see Table 2.) Notice that in going from

number to string, they always start with "M" and we have a "$" to indicate that they give string values. To go the other way, they always start with "C".

Table 2

Format Conversion Functions

Function	Var.Type	Field Size
	Number to String	
MKI$	Integer	2
MKS$	Single Precision	4
MKD$	Double Precision	8
	String to Number	
CVI	Integer	2
CVS	Single Precision	4
CVD	Double Precision	8

Now, the result of the number-to-string functions does not give us a printable number. For integers, what we have is a two-byte, binary number that represents our integer. In fact, you couldn't tell it from the number stored in memory. What has happened is that the system has been fooled into thinking this is a string.

Where is it?

Random access files give us a lot of power, but they require us to do a lot more work than with sequential files. With a random file we can get to any piece of information in the file without delay and without searching. All we need to know is the record number for the information, that is, what was the number we used when we put the information into the disk file?

The unfortunate situation in many applications is that the information we want to file has no natural ordering number. For example, let us say we are storing a list of our musical cassette collection. We could assign a number to each cassette and store the information about it by the cassette number. That's good, but it has some problems: (1) the numbers have no natural relation to individual diskette records, the numbering is *imposed*; (2) the information we want about the cassettes has no relation to the numbers unless we also impose a numbering system; and (3) we still have to add some kind of system to handle things like alphabetical and category searches.

Imagine the problem of trying to find something in your cassette collection. You know that you want something by Mozart, or Duran Duran. How are you going to find it? By number? It would be more convenient to simply tell the computer what we want and let it worry about finding it. Since the numbering system doesn't relate to the kinds of questions we want the data base to answer, we are back to brute-force searching to find things by name.

We could go on, but the primary problem should be clear. For many situations, the record numbers do not bear a direct relation to the problem unless we impose a relation. This is not necessarily bad, but is usually over-used.

Problems do exist where such an externally imposed relation is used. For example, in an inventory with part numbers from one to 1000, we might simply assign space on a data diskette so that each part number corresponds to the record number exactly. This is a very simple solution to a very tricky problem.

The advantages to this simple kind of file model are: (1) it is very simple to program; (2) it does not involve much special programming to make it work; (3) there is no translation to make between part numbers and record numbers and (4) it is very fast.

The disadvantages are: (1) part numbers not being used are just empty space on the diskette; (2) part numbers must be *forced* into correspondence with the record numbers; (3) the largest part number is set by the maximum size of the disk file; and (4) the information on a single part might not correspond to the size of one record.

If the problem is best solved without special tricks or programming techniques, then by all means, try the simplest way. If some of the disadvantages are important, it may be necessary to improve your programming skills to accommodate the requirements of your application. ■

In the program on the facing page note that we had to use a record length when we opened the file. This may or may not be the case for you depending on your computer. Check your BASIC manual for the syntax and your DOS manual to find the default record length of your machine.

```
100 REM ******************************************************
110 REM *                                                    *
120 REM *            RANDOM ACCESS FILE DEMO                 *
130 REM *            FILENAME: RANDEMO.BAS                   *
140 REM *                                                    *
150 REM ******************************************************
160 ' Clear some string space if your machine needs it.
170 ' CLEAR 10000
180 ' A$ will hold the inputs in memory, N$ holds the field names.
190 DIM A$(10,7),N$(7)
200 ' Open the file for random access
210 OPEN"R",1,"TEST.DAT",256
220 ' This is the FIELD statement from the article. Everything is
230 ' stored in string form. Try re-doing this for the conversion
240 ' functions for numbers.
250 FIELD #1, 20 AS NM$, 8 AS PR$, 8 AS CS$, 4 AS OH$, 20 AS SP$, 4
    AS RL$, 190 AS CM$
260 ' Read in the field names from the data statement.
270 FOR I=1 TO 7
280   READ N$(I)
290 NEXT I
300 DATA NAME,PRICE,COST,ON HAND,SUPPLIER,REORDER LEVEL,COMMENT
310 ' Loop to enter data into the file in records 1 to 10.
320 FOR I=1 TO 10
330   CLS:PRINT:PRINT:PRINT TAB(10)"RANDOM FILE TEST":PRINT
340   PRINT TAB(10)"ITEM NUMBER: ";I
350 ' Prompt for item by name, then input it.
360   FOR J=1 TO 7
370     PRINT TAB(10) N$(J);
380     INPUT A$(I,J)
390   NEXT J
400 ' Put the data into the buffer fields with LSET. Also try
410 ' running this program using RSET instead to see what happens.
420 ' Remember to use the "M" conversion functions if you redo
430 ' this with numbers instead of strings.
440   LSET NM$=A$(I,1):LSET PR$=A$(I,2):LSET CS$=A$(I,3):LSET
    OH$=A$(I,4)
450   LSET SP$=A$(I,5):LSET RL$=A$(I,6):LSET CM$=A$(I,7)
460 ' PUT the buffer onto the disk at record location "I".
470   PUT 1,I
480 NEXT I
490 ' Look at records on request and compare them with what
500 ' we actually input from the keyboard.
510 CLS:PRINT TAB(10)"RANDOM FILE TEST":PRINT
520 PRINT TAB(10)"ENTER A RECORD NUMBER (ENTER 0 TO END)";
530 ' Provide for the end of the routine with the 0 check.
540 INPUT N:IF N=0 THEN 770
550 ' GET the requested record. Note: If the record number is
560 ' >10 the program will error.
570 GET 1,N
580 ' Display the FIELD names and what was put there next to
590 ' what is on the disk.
```

```
600 PRINT,"ENTERED","IN FILE"
610 ' If you are experimenting with the type conversion functions,
620 ' use the "C" conversions on the field variables in order
630 ' to get printable numbers.
640 PRINT N$(1),A$(N,1),NM$
650 PRINT N$(2),A$(N,2),PR$
660 PRINT N$(3),A$(N,3),CS$
670 PRINT N$(4),A$(N,4),OH$
680 PRINT N$(5),A$(N,5),SP$
690 PRINT N$(6),A$(N,6),RL$
700 PRINT N$(7),A$(N,7),CM$
710 ' Wait now until you have a chance to see that it's right.
720 PRINT"PRESS ANY KEY TO CONTINUE";
730 ' If any key is pressed, INKEY$ will have a value.
740 ' In that case, go get another record number.
750 IF INKEY$<>"" THEN 510 ELSE 750
760 ' End of program. Always close the file before leaving.
770 CLOSE
780 END
```

Programming Notes

Have you ever wondered how a deleted or killed disk file can be restored? There are commercial programs available that will un-kill a file. They don't always work though, and here is why. When you kill or delete a file from disk the program or file is not actually touched. The kill or delete command usually only removes the filename from the active directory of the disk. In effect, it releases the space the file is using for some future data. Commercial programs that un-kill can be used at this point to restore the directory entry. If the released file space was used for another file or program in the meantime, the original is lost and cannot be recovered.

In most of our programs until now, we have used statements like this: IF A<>1 THEN GOTO 100. The "GOTO" is, of course, unnecessary. It is implied and our example may just have well been written: IF A<>1 THEN 100. We like to include it for clarity, but it is not really required, and if you are short on memory, you can save a few bytes.

Here's a neat little program to play with. It was sent in by Alva A. Shipman, of Kankakee, Illinois. You type in your name in all capital letters and the program will make lower case of all but the first letter in each word. He said it was "just a little learning routine" and he is right.

```
10 ' CHGCASE.BAS ALVA A SHIPMAN
15 DIM AB$(100)
20 CLS
25 LINE INPUT"INPUT YOUR NAME ";
   A$
30 PRINT
35 K=LEN(A$)
40 AB$(1)=CHR$(ASC(MID$(A$,1,1)))
45 FOR I=2 TO K
50   AB$(I)=CHR$(ASC(MID$(A$,I,
     1))+32):IF AB$(I)=CHR$(64)
     THEN AB$(I)=CHR$(32) ELSE IF
     AB$(I-1)=CHR$(32) THEN AB$(I)
     =CHR$(ASC(MID$(A$,I,1)))
55 NEXT I
60 FOR J=1 TO K
65   AD$=AD$+AB$(J)
70 NEXT J
75 PRINT
80 PRINT AD$
```

Puzzler # 5 and # 6

and the answer to puzzler # 4

Puzzler 4 from the last issue asked for the least amount of code needed to print the larger of two values that were input. It was also supposed to print zero if the two values were equal.

It turns out that the number of characters required to do the job is 21 (including the command PRINT, but not the line number.) See figure 1. We (and apparently, you too) couldn't find a way to do it in less. Using IF...THEN to find the larger value would work, but would not use the least code to do it. Using Boolean values shortens the code and allows the decision to be made directly. Somehow, understanding what happens is easier than explaining it. If we take (A>B) we get -1 if A is greater than B. We get zero if it is not. If we multiply the resulting number by A we will get -A, which is -1 times A or zero when A<B. Otherwise, if we do (B>A)*B, we get -B when B>A and zero when B<A. If we subtract the two parts of the formula (A>B)*A-(B>A)*B and multiply by -1 we have the formula which works.

Figure 2 shows how this formula can be included in a defined function. You can use it to test for equality of A and B, as in: IF FNBIG(A,B)=0 then.... Or, you can test to find out which *variable* is the larger of the two: IF A=FNBIG(A,B) THEN..(A is the larger) ELSE (B is).

It is interesting to note that this formula will return the more positive of two negative numbers and not the absolute value of the number. Which is to say that -3.5 and -3.8 will show -3.5 as the larger number.

Of the many answers we received to this puzzler, only four of you did it in 21 bytes. Most of the others worked correctly, but used more code to get to it. One of them, A*-(A>B) OR B*-(B>A), worked well on any numbers except decimal numbers. Given 3.4 and 3.5, for example, it returned 4 as the larger of the two. Those who did it in 21 were:

E. L. Clark, St. Petersburg, Florida
Jerry Bails, St. Clair Shores, Michigan
Richard Oberdorfer, Newport, Washington
Mark Gardner, N. Hollywood, California

Someone commented that we would have to come up with tougher puzzlers than this one.

John Anderson, of Arlington, Massachusetts, made an interesting comment:
..."I really enjoy the Puzzler. However, I feel that you should point out that the objective of good code is *not* to be a "Puzzler" but to be clear and simple, for then it has the best chance of being correct..."

We agree, but the puzzlers are fun, and sometimes even useful.

Thank you all again for responding.

Puzzler 5

While working on a text formatting program (for a future issue) we ran across a particularly interesting case involving leader lines. On typesetting machines, leaders are lines that can be drawn across the available line length space. As an example, there is one earlier in this column, just before we started with Puzzler 5. It was a font 3 leader (bold underline), which covers the entire line length (in this case, 20 picas, or about three and one-quarter inches.) Leaders automatically adjust to remaining line length, and can come before some word or words, or after them. They can also be used to fill between two groups of words, assuming there is enough space.

We write text for CodeWorks on a computer, then send our files (written with a word-processor) via RS-232 to the typesetter. This means, of course, that there must be some convention in using control codes to tell the typesetter what to do and when. On our particular typesetting machine (an A-M Varityper 3510) the control convention is to use a dollar sign followed immediately by two capital letters. This signals the typesetter that the two capital letters, and the characters following, up to the next space, are some kind of control and to *act* on it but not to *print* it. The control code to use a leader is a $UL.

Although this is not particularly relevant to our

puzzler, it does provide some background clues. During the development of our text processing program it became apparent that making leaders which easily adjust to the line width would be very nice to have. Using the typesetting code and modus operandi seemed natural. It also seemed like a perfect "black box" problem for our puzzler, which follows:

Given a line length of 40 characters (variable LL) and A$ as the input string, the black box produces output as in figure 3. The input lines look like this:

```
100 * REM PUZ5.BAS
110 LL=40
120 PRINT"For a leader line, insert $UL at the
place"
130 PRINT"where you want it to appear in the
line."
140 PRINT
150 LINE INPUT"Input A$ ";A$
```

That is what goes into the box. Some examples of what comes out of the box are shown in figure 3. The question is: What is the code in the black box?

We have it down to less than a dozen BASIC statements. It goes without saying that they are mostly involved with string manipulation. They are all common BASIC functions found in most Microsoft BASICs (with the exception of some of the lap-portables.)

On this one we are not necessarily looking for the shortest code. Rather, we are looking for an approach and code that solves the problem in the most straightforward and efficient manner. That leads to a subjective evaluation on our part, but we won't be the only judge. We will print the best of your answers and let you in on the judging. To keep things simple, do not use more than one leader command per input line. Figure 3 provides several clues.

Puzzler 6

Now, just to get our puzzler numbers and our issue numbers synchronized, here is "throw in" puzzler 6:

In the above discussion we noted that the code to make the typesetter do a leader line was $UL. There is no key or switch on our typesetter to prevent those control codes from acting. So how did we get them typeset on this page?

Have fun with both puzzlers. We are now in sync.

Figure 1

```
10 INPUT"WHAT IS VALUE OF A";A
20 INPUT"WHAT IS VALUE OF B";B
30 PRINT-(A>B)*A-(B>A)*B
40 '234567890123456789012345
```

Figure 2

```
10 DEF FNBIG(A,B)=-(A>B)*A-(B>A)*B
20 INPUT"WHAT IS VALUE OF A";A
30 INPUT"WHAT IS VALUE OF B";B
40 PRINT FNBIG(A,B)
```

Figure 3

```
Input A$: $UL leader first
_____ leader first

Input A$: leader last $UL
leader last _____

Input A$: leader in $UL the middle
leader in _____ the middle

Input A$: Coke(per case) $UL $6.95
Coke(per case) _____ $6.95

Input A$: This $20 device is UL approved$UL
This $20 device is UL approved_____
```

Plist.Bas

Give some form to your program listings

Terry R. Dettmann, Associate Editor. This useful utility is our "tutorial" program for this issue. It observes line indents on wrap-lines and pages to your choice, along with headers and page numbers. Terry wrote it in a conversational style with his friend "Harvey" and tried to answer the most asked questions about such a program. It also allows single or double spacing, a much asked-for feature by many of you.

We got together that night to do a little training in BASIC programming. Harvey and I had known each other for awhile, but Harvey had only recently become interested in programming and wanted to soak me for information. I didn't mind, it's what I do best.

"I'm fresh out of ideas for what to do," Harvey said. "I want to learn, but if you get too complicated, I'll be lost before we're started."

"Let's start with a simple project then, one that we can use when we go on to more difficult projects later. How about a program that formats a BASIC program onto a printed page with nice headers and so forth. Shall we start?"

"All right, but don't go too fast or I'll be lost for sure."

"OK. To start with, let's figure out what we want this program to do. It's got to read a program from the disk and output it to the printer nicely formatted. Further, we'll want to have the listing tell us the name of the program file we're formatting and number the pages. To save ourselves a lot of trouble, we'll insist that the program on disk has to be saved with the 'A' option on the save command."

"What's that?"

"The 'A' option tells the computer to write the program to the disk in readable form. Some people call this the ASCII option since it writes out the whole program in readable ASCII form, just like you typed it into the computer. To show you what to do, I'll type in a little two line program and save it normally and in ASCII form like this:"

```
10 REM THIS IS A TEST PROGRAM
20 PRINT "TESTING 1 2 3":GOTO 10
SAVE "TEST.BAS"
SAVE "TEST.BAS",A
```

Figure 1 - Two line program and both save commands.

"On the disk, the normal save has made a file in a special compacted form which matches what the interpreter for BASIC stores in memory. This saves time when you want to start a BASIC program from disk. With the 'A' attached, the program is stored in readable form so we can write a program to read it like any other text file. Let's see what we can do with it.

"First, let's write a program to simply read the file and write it to the screen. Since the file is a sequential file, we'll have to OPEN it and then read in the lines one by one and deal with each one while it's in memory. We can do that like this:"

```
200 OPEN "I",1,"TEST.BAS"
210 IF EOF(1) THEN 300
220 LINE INPUT #1, LN$
230 PRINT LN$
240 GOTO 210
300 CLOSE 1
999 END
```

Figure 2 - short program to read lines from the file and write to the screen.

"If we run the program as is, it will read the program TEST.BAS from the disk to the screen. Simple huh?"

"Oh sure" was Harvey's reply. "I understand how to open files. The first string in quotes tells the

system we want to input information from the file, the number after it tells it to use file buffer 1 in memory, and the last is the filename to use. But what does the EOF in the next line mean?"

"EOF stands for 'End of File'. It's a test that tells us whether or not the file buffer given in the number has reached the end of the file. You can imagine it returning a TRUE or a FALSE value. If EOF(1) is TRUE, that means that we have already reached the end of the file. If we try to read again, we'll get an error from the BASIC interpreter to tell us we tried to read beyond the end of the file. That will cause our program to crash. We'd like to avoid crashes, they tell us we didn't write the program right.

"After that it's simple, read a line, print it, and then go back to the test to see if the last read took us to the end of the file.

"We simply keep that up until we're done. Now does it make sense?"

"Sure, but what good does this do us for formatting the printed text?"

"Well, I'll show you. Will you agree that the real problem could be solved if we would just have a formatted way to print the line of text instead of our simple print statement?"

"Sure, but I don't see how we can do it."

"We can solve the problem easily by replacing the PRINT statement with a subroutine that prints the line the way we want it. After all, a statement in BASIC such as PRINT is really just a subroutine call anyway, it's just that the subroutine is in the interpreter and not in our program."

"That actually makes some sense for once. Subroutines and GOSUB's didn't really make much sense but when you think of them as equivalent to new instructions for the interpreter to execute, it seems simpler."

"In some programming languages, the similarity is so great that you actually ARE creating new statements. We call these 'extensible' languages since we can 'extend' them as we want. Back to our subroutine though. We could write the subroutine like this and make it do the same thing as the original program:"

```
200 OPEN "I",1,"TEST.BAS"
210 IF EOF(1) THEN 300
220 LINE INPUT #1, LN$
230 GOSUB 1000
240 GOTO 210
300 CLOSE 1
999 END
1000 REM --- PRINT A LINE
1010 PRINT LN$
1020 RETURN
```

Figure 3 - same program but with a subroutine containing the PRINT statement.

"When we run the program, it does the same as before, no change. Now, let's make the subroutine more complicated so it breaks the line apart the way we want. Let's start by looking at the problem of breaking the line apart as a problem in itself. You agree that the rest of the program will print the program line by line don't you?"

"Sure"

"Then if we solve the problem of breaking apart a line, we've got the whole program ready to go. Let's start with breaking the line on spaces only. That would be OK if we were just trying to do word wrapping for a print formatter, but there is going to be more before we're done. For the moment though, let's deal with spaces only.

"Let's say we want the line to be 60 characters long. If the line is less than 60 characters long to start with, we'll just print the line as is. If the line is greater than 60 characters long, then we go at it like this:

1) Start looking at the line starting at the 60th character. If it's a blank then we breat the line at that point and look at the remainder of the line as a separate problem.

2) If the 60th character is not a blank, then we step back one character at a time until we find one and break the line there.

3) If we don't find a blank by some point, we just say Aw *$!% and break the line on the 60th character anyway.

"Now the subroutine at line 1000 could be written like this:

```
1010 REM --- PRINT A LINE
1020 GOSUB 1100
1030 PRINT STRING$(LI," ");PL$
1035 IF LN$="" THEN RETURN
     ELSE 1020
1040 GOTO 1010
```

Where subroutine 1100 will implement a routine to copy the first 60 characters or less of IN$ into JN$ and then put the rest of the characters in IN$. This way the program sets up a loop which extracts the portion of the line to print in the subroutine at line 1100, then prints it, and finally checks to see if there's anything left to print. If not,

it's done."

"But this still doesn't get the line printed really does it?"

"No, the real heart of the program is now concentrated in the subroutine at line 1100. As I said before, the subroutine will break apart the line and deal it out for printing as we need it.

"Now let's get the rest of the program written. We start off at line 1100 like this:

```
1100 REM -- BREAK A LINE
1110 IF LEN(LN$)<60 THEN PL$=LN$:
     LN$="":RETURN
1120 FOR J=60 TO 30 STEP -1
1130 IF MID$(IN$,J,1)=" " THEN
     1150
1140 NEXT J
1150 J=60
1160 JN$=MID$(IN$,1,60):IN$=
     STRING$(SL," ")+
     MID$(IN$,J+1)
1170 RETURN
```

"This makes it pretty simple. First we see if the line is already less than 60 characters in length. If it is, then just return it for printing. If it's not short enough, we start at the 60th character and scan backwards through the line until we find a blank or get down to the 30th character. If we ever get that far, we simply break the line at the 60th character and decide that we can't break the line normally."

"That's all there is to it?"

"That's all. At this point, the program is complete. Everything that has to be done is done. It will successfully print the program with lines formatted to the correct length on the screen."

"I can see how to get the program to the screen now. I can even see how I can get it to the printer by changing the PRINT statement to an LPRINT statement. I get a pretty nice listing that way, but how do I get it to break the listing into numbered pages with headings?"

"In order to do page breaks, we have to keep track of how many lines we're printing. We might as well also change all PRINT statements to LPRINT right away so we can print to the printer.

"In paging our listing, we have to create a routine (another subroutine again) which will print the headers we want. Here's a simple one that should work:

```
1050 REM -- PRINT HEADER
1060 FOR I=1 TO MX:LPRINT" ":
     NEXT I:PG=PG+1
1070 LPRINT STRING$(LI," ");:
     LPRINT"FILENAME: ";FF$;
     TAB(LW)"PAGE";PG
1080 LPRINT STRING$(LI," ");:
     LPRINT STRING$(LI,"-")
1090 MX=6:LC=5:RETURN
```

"What this does is get us from page to page with some blank lines at the bottom and the top. The variable MX is set to count down several lines (3 on the top and 3 on the bottom) so we don't write over the page break. Then we increase the page number (PG) for the next page and print a two line header with the file name and page number.

"As we come out of the header, we set the number of lines to get across the page break at 6 since we will print 60 lines per 66 line page."

"Wait a minute! What's this about a 66 line page and printing 60 lines? And where did 'MX' and 'LC' and 'LW' come from?", Harvey asked.

"Sorry. First let's look at the numbers 60 and 66. If you use standard 8 1/2 X 11 inch paper, your printer can print 66 lines on a single sheet and 80 characters across the sheet. This is pretty standard for the kinds of printers we deal with. Some printers allow changing these sizes, but we won't worry about those for this.

"Since we don't want to print on top of the page break, we'll limit ourselves to printing only 60 lines on each page including the header. That's also pretty standard. Now, on the very first sheet, we put the printer aligned at the top of the page, so the first time we print a page, we'll want to print only three lines. We can handle that by setting MX to 3 when the program starts. On any page after the first, we have to also space down three lines at the bottom of the page so we set MX to 6 at the end of the subroutine.

"The variable LC is our line counter. We'll bump it every time we print a line so that we can check to see when we've printed all the lines we need on a page. We'll force it to print the first header by setting the line counter to 99 when the program starts. 'LW' is simply the line width which we set to 60 characters.

"In order to make this work, we need to add some other lines to call the header routine and set the line values we need. You can add these and then try the program again:

```
10 LP=60:PL=66:LI=10:LW=60
205 PG=0:MX=3:LC=99
1025 IF LC>LP THEN GOSUB 1050
1035 LC=LC+1
```

"Pretty impressive, isn't it?"

"Can I find a use for that! I'll make nice listings of all my programs with this so I can keep them in a notebook for reference."

"That's a good idea. Use the notebook to document your programs along with notes on the design. It will help you make changes later. I guarantee that six months after you wrote the program, you'll have to learn it all over again.

"Before you just run off though, I have a listing here of the same program from the CodeWorks Download which goes a little further still. I'll explain how it works and then you can go off and get a copy yourself from the Download System or type this one in directly. It's been renumbered to make it easier to type in using the AUTO numbering function in BASIC.

"To start with, the program is built in the same way we've been building our page lister, but it's more flexible and sophisticated. The main part of the program runs from lines 100 through 550 and is broken up into three major pieces.

"Lines 100-170 are an initialization section. You'll recognize the variable setting in line 150, but there's something new as well. DS is a double spacing parameter. We set it to '1' for single spacing and '2' for double spacing. Line 160 defines variables TRUE and FALSE for use in logical decisions. I find it easier to understand a program which has IF statements using variables that are TRUE or FALSE. It's just a convenience but I find it nice. It also matches common practice in programming languages like 'C'.

"Line 170 adds a whole new dimension to the program. It sets the characters we'll allow lines to be broken on. In a program we can safely break on more than just spaces. I'll show you how it works in a minute.

"Lines 180 through 480 do some setup by interacting with the operator. Beyond pretty formatting for the screen, we've set it up to act on a menu of options. You can leave everything at default just by pressing the RETURN or ENTER key. If you type a number though, we'll enter a new value for that parameter. Notice how lines 320 through 350 make this decision for us on single key strokes.

"After we've gotten the parameters set the way you want them, we input the file name to print rather than requiring you to retype the program each time you want to change the program name. Then comes the main loop in lines 490 through 530 just as before."

"Nice," Harvey said, "How did you know to check for all those things? I would have never thought of setting up a menu to enter things. In my programs you have to type all the parameters each time."

"To be honest with you, I cheated. See I've done this kind of program before, and each time I've done one I've learned a little more. Every program I do makes the next just a little better.

"The rest of the program is pretty much the same. Our print lines subroutine has some new variables as you'll see. SL is defined to locate the first non-blank character in the line after the line number for indenting.

"First we set it to the first blank in the line which is always after the line number, then in line 590 we keep advancing it until the next character is not a blank.

"In line 630, we take care of double spacing if it's needed. The page header routine hasn't changed at all, but the line selection routine has gotten more complex.

"We set the string flag to FALSE and then check to see if the line is short enough to print directly."

"Whoa, Whoa! You brushed over that string flag too quickly. Why bother keeping track if we're in a string? Who cares?"

"Well I do for one, and when you see why, you'll care too. If you don't check whether you're in a string and find a breakable character in it, you'll break the string right in the middle and make it harder to understand. Since I generally keep my strings pretty short and don't write big long ones, I decided to keep them together as much as possible, so when I'm inside a string, I don't break the string if I can help it. Line 750 does this by skipping the break step if I'm inside a string.

"Lines 730 through 770 are our loop to break the line. Lines 740 and 750 are in there for the string check I've already mentioned. Line 760 has changed some though. Now I look at each character and check to see if it's in the string I defined above with the characters to break the line on. If I find one, I break the line as before."

"Move over. I want to get this from the download system right away. What's the number again?"

"206-475-2356. Remember it's 300 baud, 8 bits, ignore parity, 1 stop bit."

```
100 REM * PLIST.BAS * PROGRAM LIST UTILITY * CODEWORKS MAGAZINE
110 REM * 3838 S. WARNER ST. TACOMA, WA 98409 (206) 475-2219
120 REM * WRITTEN BY TERRY R. DETTMANN
130 DEFINT A-Z
140 'CLEAR 10000: ' USE ONLY IF YOUR MACHINE NEEDS TO CLEAR SPACE
150 LI=10:LW=60:LP=60:PL=66:DS=1
160 TRUE=-1:FALSE=0
170 WP$=" :;+-,"
180 CLS: ' CHANGE THIS CLEAR SCREEN COMMAND TO SUIT YOUR MACHINE
190 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
200 PRINT "        P R I N T E R   P A G E   F O R M A T T E R"
210 PRINT "            a page formatter by Terry R. Dettmann"
220 PRINT STRING$(60,"-")
230 PRINT
240 PRINT"Current settings: "
250 PRINT TAB(10);"1 - Left Margin is now set at ---";LI
260 PRINT TAB(10);"2 - Line length is now set at ---";LW
270 PRINT TAB(10);"3 - Page length is now set at ---";PL
280 PRINT TAB(10);"4 - Lines to print now set at ---";LP
290 PRINT TAB(10);"5 - Line spacing is now set at --";DS
300 PRINT
310 PRINT"Press Return for current settings, 1 to 5 for changes."
320 Q$=INKEY$:IF Q$="" THEN 320
330 IF Q$=CHR$(13) THEN 470
340 IF Q$<"1" OR Q$>"5" THEN 310
350 ON VAL(Q$) GOTO 360,380,400,420,440
360 INPUT"How many spaces for left margin --------- ";LI
370 GOTO 180
380 INPUT"How many characters per line ------------ ";LW
390 GOTO 180
400 INPUT"How long is your printed page ----------- ";PL
410 GOTO 180
420 INPUT"Print how many lines on the page -------- ";LP
430 GOTO 180
440 INPUT"Enter 1 for single, 2 for double spacing- ";DS
450 IF DS<1 OR DS>2 THEN GOTO 440
460 GOTO 180
470 PRINT
480 LINE INPUT"Name of file to print (must be an ASCII file) ";FF$
490 OPEN "I",1,FF$:LC=99:PG=0:MX=(PL-LP)/2
500   IF EOF(1) THEN 540
510   LINE INPUT #1,LN$
520   GOSUB 570
530   GOTO 500
540 CLOSE
550 END
560 REM ------- PRINT A LINE ---------------------------------
570 SL=INSTR(LN$," ")
580 IF MID$(LN$,SL+1,1)=" " THEN SL=SL+1:GOTO 580
590 IF LC>LP THEN GOSUB 650
600 GOSUB 700
610 LPRINT STRING$(LI," ");PL$:LC=LC+1
```

```
620 IF DS=2 AND LC<=LP THEN LPRINT" ":LC=LC+1
630 IF LN$="" THEN RETURN ELSE 590
640 REM ------- PRINT PAGE HEADER ----------------------------
650 FOR I=1 TO MX:LPRINT" ":NEXT I:PG=PG+1
660 LPRINT STRING$(LI," ");:LPRINT "FILENAME: ";FF$;TAB(LW)"PAGE";PG
670 LPRINT STRING$(LI," ");:LPRINT STRING$(LW,"-")
680 MX=PL-LP:LC=3:RETURN
690 REM ------- SELECT THE LINE TO PRINT ---------------------
700 SF=FALSE
710 IF LEN(LN$)<LW THEN PL$=LN$:LN$="":RETURN
720 FOR J=LW TO 30 STEP -1
730   IF MID$(LN$,J,1)=CHR$(34) THEN SF=NOT SF
740   IF SF THEN 760
750   IF INSTR(WP$,MID$(LN$,J,1))>0 THEN 780
760 NEXT J
770 J=LW
780 PL$=MID$(LN$,1,J):LN$=STRING$(SL," ")+MID$(LN$,J+1)
790 RETURN
```

We have used this program to list itself. You can do the same to make sure you have entered it correctly.

"But I can't ignore parity."

"Well start with Odd parity then, if it doesn't work then try Even. Some computers seem to work one way, some another. I've been successful using 7 bits, no parity, but some people have trouble."

"When you get the program, there is an improvement you can think about for it. Imagine what would happen if the 60th character in the line were already inside a string when we reach that point. The program wouldn't deal with the strings correctly. In order to check this possibility, we have to scan the line to the 60th character and set the string flag (SF) when we start the line break routine. Do you think you can do that?"

"Sure, let me at it."

"There are lots of other things you could do to the program. For example, you could put the date and time of the listing there. You could modify the program to understand compacted BASIC files and translate them back to the full ASCII form and then do the listing. You could also adapt the program to make nicely paragraphed listings of text files. Try some of these and see if you can do them." ■

"His *keywords* are *shut up* and *get out of here.*"

Dstat.Bas

Describing your statistical samples

Philip E. Clark, Martinez, GA. There are statistics and then there are statistics. This program provides you with descriptive statistics on any set or sets of data. The solution fits on one screen and may be printed using your screen-print function, if desired.

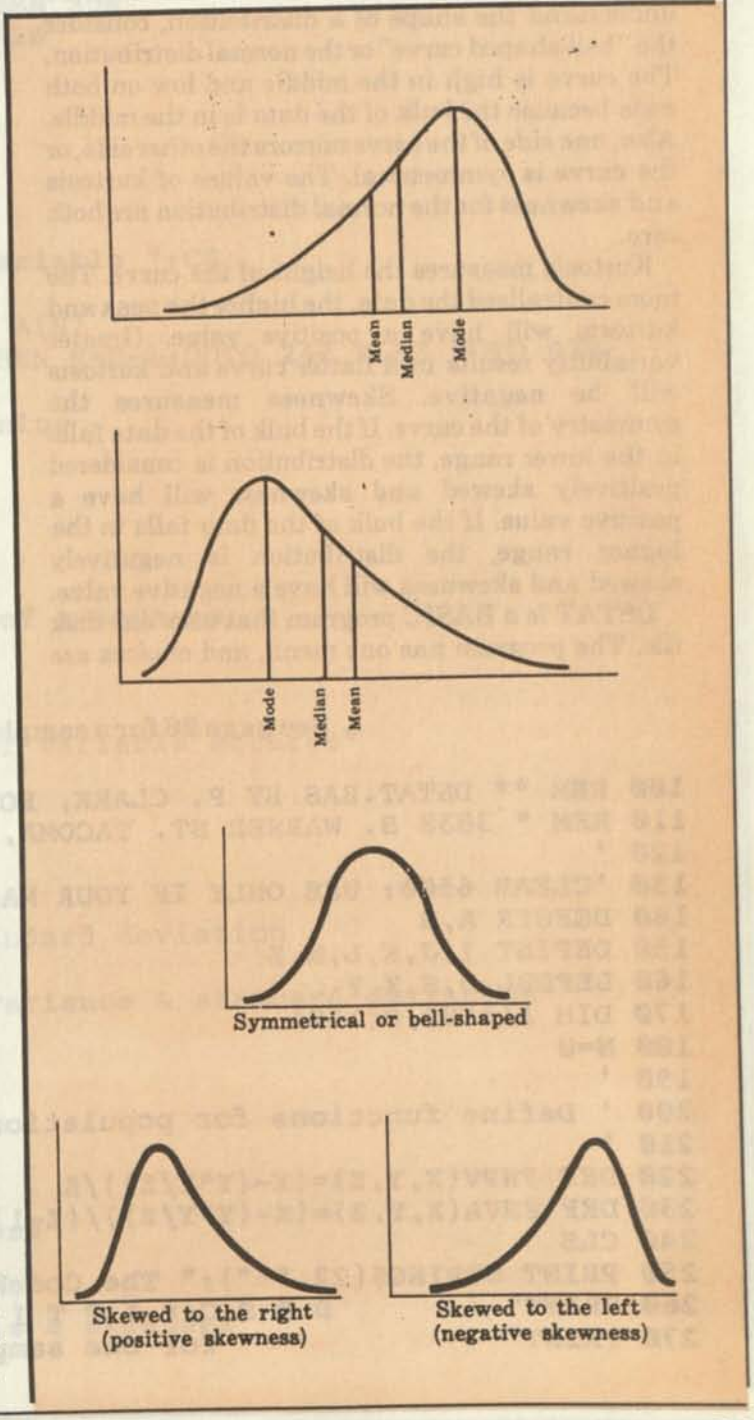
DSTAT.Bas is a descriptive statistics program that calculates descriptive measures sufficient to portray a data set. However, to correctly interpret the computed values, the user must have an understanding of each statistic. The purpose of this article is to present an overview of the program's descriptive measures as well as describe the program.

Statistical tests are often divided into two groups: inferential and descriptive. Inferential statistics, such as t-test, analysis of variance, and chi-square, help determine if an hypothesis is true or false. Descriptive statistics describe or portray a set of data. Any set of data can be summarized by measuring three characteristics: central tendency, variability and the shape or curve of the distribution. Descriptive statistics involve a set of mathematical formulae that measure these qualities.

Measures of central tendency provide an estimate of where on a continuum most of the data falls or clusters. The common tests are Mean, Median and Mode. The Mean is simply the arithmetic average of the data. The Median is the midpoint of a set of data. Half the scores will be above and half below the median. The Mode is the value of values that occur most often. A set of data may have more than one mode.

The Mean is the most common estimate of central tendency. However, extremely high or low scores pull the mean in their direction. When extreme scores occur, the median or mode may be the better measure of central tendency. When describing group performance, such as class performance on an exam, the mean is generally appropriate. However, when attempting to determine the typical salary in the United States, the mean will provide an inflated estimate because of the extremely high salaries of a few citizens.

Variability refers to the dispersion of the data and is measured by Range, Variance, and Standard Deviation (SD). Range is the difference between the highest and lowest scores. Obviously,



the greater the range, the greater the data's variability. Of Variance and standard deviation, the latter is the most useful. The unit of measurement of standard deviation is the same as the original measure. For example, if the data is in inches, the standard deviation will be in inches. As a general rule, approximately 68% of a sample will fall within one standard deviation above and below the mean and 95% within two standard deviations of the mean.

The shape of the distribution of scores is measured by tests for *kurtosis* and *skewness*. To understand the shape of a distribution, consider the "bell-shaped curve" or the normal distribution. The curve is high in the middle and low on both ends because the bulk of the data is in the middle. Also, one side of the curve mirrors the other side, or the curve is symmetrical. The values of kurtosis and skewness for the normal distribution are both zero.

Kurtosis measures the height of the curve. The more centralized the data, the higher the peak and kurtosis will have a positive value. Greater variability results in a flatter curve and kurtosis will be negative. Skewness measures the symmetry of the curve. If the bulk of the data falls in the lower range, the distribution is considered positively skewed and skewness will have a positive value. If the bulk of the data falls in the higher range, the distribution is negatively skewed and skewness will have a negative value.

DSTAT is a BASIC program that uses one disk file. The program has one menu, and choices are made by pressing the appropriate key. The program utilizes a string array and a single precision array of equal length. Data is entered and stored as string values. This allows positive, negative and zero values to be used as data. String values are converted to single precision prior to processing by the VAL function.

The following variables are double precision:

SU: the total of all the data, also called the sum of x's.
SV: the total of each datum squared, also called the sum of the x-squares.
Q3: the total of each deviation score (difference between each datum and mean) raised to the third power.
Q4: same as Q3, but deviation scores are to the fourth power.

The values of these variables can be quite large and double precision was necessary despite the slowing of program execution.

Data may be entered from the keyboard or disk drive. After entering data from the keyboard, type END to return to the menu. You may load data from the diskette and enter more values manually. When processing the data, you are informed what statistics are being calculated. You may recalculate statistics at any time since intermediate values are zeroed each time through.

The program could be improved in a few ways. An additional statistic, Confidence Interval, could be added. ■

See page 26 for a sample run of this program.

```
100 REM ** DSTAT.BAS BY P. CLARK, FOR CODEWORKS MAGAZINE
110 REM * 3838 S. WARNER ST. TACOMA, WA 98409 (206) 475-2219
120 '
130 'CLEAR 6500: USE ONLY IF YOUR MACHINE NEEDS TO CLEAR SPACE
140 DEFSTR A,B
150 DEFINT I,J,K,L,N,Z
160 DEFDBL D,S,X,Y
170 DIM A(150),C(150)
180 N=0
190 '
200 ' Define functions for population variance and sample variance.
210 '
220 DEF FNPV(X,Y,Z)=(X-(Y*Y/Z))/Z
230 DEF FNVA(X,Y,Z)=(X-(Y*Y/Z))/(Z-1)
240 CLS
250 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
260 PRINT"          D E S C R I P T I V E   S T A T I S T I C S"
270 PRINT"                  for one sample - by Philip Clark"
```

```
280 PRINT STRING$(60,"-")
290 PRINT
300 PRINT TAB(10)"I)nput or add data"
310 PRINT TAB(10)"R)ead data from disk"
320 PRINT TAB(10)"W)rite data to disk"
330 PRINT TAB(10)"S)how results
340 PRINT TAB(10)"Q)uit the program"
350 PRINT:PRINT"Selection ...";
360 B=INKEY$:IF B="" THEN 360
370 IF ASC(B)>90 THEN B=CHR$(ASC(B)-32)
380 I=INSTR(1,"IRWSQ",B):IF I=0 THEN 360
390 ON I GOTO 430,1570,1720,530,1820
400 '
410 ' Input data from keyboard
420 '
430 CLS
440 PRINT"Type END to quit":
450 PRINT
460 IF C$="" THEN INPUT"Name of variable ";C$
470 N=N+1
480 PRINT"Datum #";N;"...";:INPUT A(N)
490 IF A(N)="END" OR A(N)="end" THEN N=N-1:GOTO 240 ELSE GOTO 470
500 '
510 ' Convert string to numeric value
520 '
530 FOR I=1 TO N
540   C(I)=VAL(A(I))
550 NEXT I
560 '
570 ' Compute sum of x's and sum of x-squares
580 '
590 SU=0:SV=0
600 CLS
610 PRINT"Computing sum and sum of variable squares"
620 FOR I=1 TO N
630   SU=SU+C(I)
640   SV=SV+C(I)*C(I)
650 NEXT I
660 '
670 ' Compute mean, variance, standard deviation
680 '
690 PRINT:PRINT"Computing mean, variance & standard deviation
700 ME=SU/N
710 VR=FNPV(SV,SU,N)
720 ES=SQR(VR)
730 EV=FNVA(SV,SU,N)
740 ED=SQR(EV)
750 '
760 ' Compute kurtosis and skewness
770 '
780 Q3=0:Q4=0
790 PRINT:PRINT"Computing kurtosis & skewness"
```

```
800 FOR I=1 TO N
810   DE=C(I)-ME
820   Q3=Q3+DE^3
830   Q4=Q4+DE^4
840 NEXT I
850 EK=((Q4/N)/VR^2)-3
860 OS=(Q3/N)/ES^3
870 PRINT:PRINT"Arranging data in ascending order"
880 FOR I=1 TO N-1
890   FOR L=I+1 TO N
900     IF C(L)<C(I) THEN T=C(I):C(I)=C(L):C(L)=T
910   NEXT L
920 NEXT I
930 '
940 ' Compute range and median
950 '
960 PRINT:PRINT"Computing range and median"
970 MI=C(1):MA=C(N):RA=MA-MI
980 '
990 ' Median
1000 '
1010 I=N/2
1020 '
1030 ' Integer division results in an integer quotient
1040 ' if I*2=N then sample size is even otherwise it is odd
1050 '
1060 IF I*2=N THEN MD=(C(I)+C(I+1))/2 ELSE I=(N+1)/2:MD=C(I)
1070 '
1080 ' Determine mode
1090 ' First, zero the mode array
1100 '
1110 FOR I=1 TO 5:MO(I)=0:NEXT I
1120 PRINT:PRINT"Checking for existence of mode(s)"
1130 K1=1:     ' Counter for mode array
1140 MK=3:     ' Currtent number of values in mode
1150 TS=C(I): ' Variable used to compare data
1160 MR=1
1170 '
1180 ' MR is a variable to count data items & compare with MK. To
1190 ' create a mode, MR must equal or exceed MK. There must be
1200 ' at least 3 occurrences of a datum to create the first mode.
1210 '
1220 FOR I=2 TO N
1230   IF TS=C(I) THEN MR=MR+1:IF I<>N THEN 1390
1240 '
1250 ' MK & MR are now compared
1260 '
1270   IF MK>MR THEN 1330
1280   K1=1:MO(K1)=TS:MK=MR:K1=K1+1
1290   FOR L=K1 TO 5
1300     MO(L)=0
1310   NEXT L
1320   GOTO 1370
```

```
1330    IF MK=MR THEN MO(K1)=TS:K1=K1+1
1340  '
1350  ' MK is greater than MR
1360  '
1370    TS=C(I)
1380    MR=1
1390  NEXT I
1400  '
1410  ' Display results
1420  '
1430  CLS:
1440  PRINT"Variable  ";C$;TAB(32)"Sample size  ";N
1450  PRINT:PRINT"Mean    ";ME;TAB(32)"Median  ";MD
1460  IF MO(1)=0 THEN PRINT:PRINT"There is no mode.":GOTO 1480 ELSE
      PRINT:PRINT"Mode  ";:FOR I=1 TO 5:IF MO(I)<>0 THEN PRINT MO(I);"
       ";:NEXT I
1470  PRINT" Frequency  ";MK
1480  PRINT:PRINT"Population variance  ";VR;TAB(38)"Std. deviation ";
      ES
1490  PRINT"Sample variance      ";EV;TAB(38)"Std. deviation ";ED
1500  PRINT:PRINT"Range  ";RA;TAB(20)"Max  ";MA;TAB(32)"Min  ";MI
1510  PRINT:PRINT"Kurtosis  ";EK;TAB(32)"Skewness  ";OS
1520  PRINT:PRINT"Press any key to continue";
1530  B=INKEY$:IF B="" THEN 1530 ELSE 240
1540  '
1550  ' Reads data from sequential disk file
1560  '
1570  CLS
1580  LINE INPUT"Name of disk file to read: ";F$
1590  OPEN "I",1,F$
1600  INPUT #1,C$
1610  PRINT C$
1620  IF EOF(1) THEN 1670
1630  N=N+1
1640  INPUT #1,A(N)
1650  PRINT A(N);" ";
1660  GOTO 1620
1670  CLOSE 1
1680  GOTO 310
1690  '
1700  ' Sequential output of data to disk file
1710  '
1720  IF N=0 THEN 290
1730  CLS
1740  LINE INPUT"Name of file to write to: ";F$
1750  OPEN "O",1,F$
1760  PRINT #1,C$
1770  FOR I=1 TO N
1780   PRINT #1,A(I)
1790  NEXT I
1800  CLOSE 1
1810  GOTO 310
1820  END
```

```
Name of disk file to read: DSTAT.DAT
DSTAT.DAT
2 4 6 8 9 10 11 12 13 13 13 20 13 10 7 5 3
        R)ead data from disk
        W)rite data to disk
        S)how results
        Q)uit the program

Selection ...
```

If you input the data as shown above, you should get the results shown below.

```
Variable  DSTAT.DAT              Sample size   17

Mean    9.352941                 Median   10

Mode   13    Frequency   4

Population variance    19.87543        Std. deviation  4.458187
Sample variance        21.11765        Std. deviation  4.595394

Range   18          Max   20    Min   2

Kurtosis  -9.518648E-02          Skewness   .3196447

Press any key to continue
```

Programming Notes

Some operating systems will automatically go to any other active drive to find a program or file if it is not on the logged-on drive. This is especially true of TRSDOS systems. MS-DOS does not, but you can do other interesting things by using the PATH provided in MS-DOS. For example, you can keep all your system utilities, including BASIC, on drive A. Then set the path to A and change your logged-on drive to B. Now, when you call for BASIC or any other system utility on A, the PATH command will route your request to A to find it. PATH can be included in your AUTOEXEC.BAT file, so that when you boot it will be set automatically. Using PATH without arguments will show you the current path setup. Check your DOS manual for the many uses of this command.

Word processors are very handy programming tools. You can actually write a BASIC program using a word processor. They can create ASCII files that can be loaded and run under BASIC. Another use of a word processor is to make global changes in a BASIC program. First, save the program as an ASCII file. Then load it with your word processor. Now you can search for all occurrences of a specific variable or command and automatically change it to something else. As a case in point, you could search for all PRINT statements and change them to LPRINT, or the other way around if you like.

Many times when programs are downloaded via modem, they pick up extra characters. Since BASIC expects to see line numbers when it loads, these extra characters will give a "Direct statement in file" error. Use your word processor to load the file first and eliminate the extra characters, which sometimes may be nothing more than a few spaces before the first line of code.

Beginning Basic

Look at memory with this ASCII dump program

As promised in the last issue, this installment contains a program which allows you to see, in ASCII, what your computer memory holds up to address 32767.

The real working part of the program is from lines 250 through 370. The object of the program is to read memory from a given address to another, larger, address. Then to print the address that starts each eight-byte group at the left, the ASCII value of each of the eight bytes, then space and print the value represented by each of the eight bytes. If the value is 32 (space) or less, then print a period to show that. Also if the value is greater than 126, then also print a period. In other words, we want to print characters, numbers and symbols, but not control codes.

The starting and ending addresses of memory are entered in lines 220 and 230. They become variables A and B, respectively. Line 250 says that if the starting address (which we will increment as we go along) is equal to or larger than the ending address then we are done, so return and ask for a new start and end address.

Line 260 is going to position our address properly and right-justify it, followed by the colon. In it, we are taking the RIGHT$ of four blank spaces plus the string value of the address and adding the colon. Since the string value of the address can be anywhere from one to five characters long, it will occupy at least one space (for single digits) and the spaces to the left will remain as the spaces we allowed for. Note that we end line 260 with a semi- colon to keep the cursor positioned there for what is to follow.

Next there follows a little loop that starts at the current memory address and looks at it plus the next seven addresses. In line 280 we first print three spaces to separate the data. Then we use the same right-justification idea as earlier. This time (in line 280) we allow for a maximum of three spaces (for the ASCII number), make a string value of what we found at PEEK(I), print it and leave the cursor where it is when we are done. This loop prints the value for eight bytes across the page and leaves the cursor at the end of the last one printed.

Line 300 prints four spaces again to separate the groups of data, and again leaves the cursor at the

```
---------------------- The CodeWorks ----------------------
            A S C I I  D U M P  P R O G R A M
      displays the contents of your computer's memory
-----------------------------------------------------------

Memory range is from Ø to 32767.

Show memory starting at decimal address? 4Ø4Ø
      Ending at what decimal address? 4136

 4Ø4Ø:    Ø    Ø    Ø    Ø    Ø    Ø    Ø    Ø  ........
 4Ø48:    Ø    Ø    Ø    Ø    Ø   1Ø   16  1ØØ  .......d
 4Ø56:    Ø  143   32   42   32   65   68   85  ...*.ADU
 4Ø64:   77   8Ø   46   66   65   83   32   42  MP.BAS.*
 4Ø72:   32   65   83   67   73   73   32   68  .ASCII.D
 4Ø8Ø:   85   77   8Ø   32   8Ø   82   79   71  UMP.PROG
 4Ø88:   82   65   77   32   42   32   67   79  RAM.*.CO
 4Ø96:   68   69   87   79   82   75   83   32  DEWORKS.
 41Ø4:   42    Ø   67   16  11Ø    Ø  143   32  *.C.n...
 4112:   42   32   51   56   51   56   32   83  *.3838.S
 412Ø:   46   32   87   65   82   78   69   82  ..WARNER
 4128:   32   83   84   46   32   84   65   67  .ST..TAC
```

```
100 REM * ADUMP.BAS * ASCII DUMP PROGRAM * CODEWORKS *
110 REM * 3838 S. WARNER ST. TACOMA,WA 98409 (206)475-2219
120 ' CLEAR 1000: ' USE ONLY IF YOUR MACHINE NEEDS TO CLEAR SPACE
130 CLS: ' CHANGE TO YOUR PARTICULAR CLEAR SCREEN COMMAND
140 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
150 PRINT "              A S C I I  D U M P  P R O G R A M  "
160 PRINT "     displays the contents of your computer's memory"
170 PRINT STRING$(60,"-")
180 PRINT
190 PRINT"Memory range is from 0 to 32767."
200 PRINT
210 DEFINT A,B,C,I
220 INPUT"Show memory starting at decimal address";A
230 INPUT"          Ending at what decimal address";B
240 PRINT
250 IF A=>B THEN GOTO 380
260   PRINT RIGHT$("    "+STR$(A),5);":";
270   FOR I=A TO A+7
280     PRINT"  ";RIGHT$("  "+STR$(PEEK(I)),3);
290   NEXT I
300   PRINT"    ";
310   FOR I=A TO A+7
320     C=PEEK(I)
330     IF C=<32 OR C>126 THEN PRINT".";ELSE PRINT CHR$(C);
340   NEXT I
350   PRINT
360   A=A+8
370 GOTO 250
380 GOTO 220
```

end of those four spaces. The loop starting in line 310 is going to examine the same eight bytes we just printed using the PEEK command and if the ASCII value of any of the bytes is less than or equal to 32 (ASCII space) or larger than 126 we will print a period. Otherwise, we will print the character represented by that ASCII value. Again, the semi-colon at the end of line 330 will keep the cursor positioned after whatever it had just printed. When we have printed all eight values we have completed one entire line on the screen, so the print statement in line 350 will throw the carriage and give the return for the next line. Before we begin to print the next line we need to increment the address number at which to start. This is done in line 360, where we simply add 8 to the current address number. We then go back to line 250 to print the next line.

This program was designed to fit both 64-column and 80-column screens without modification. The sample dump with this article shows a dump of memory locations 4040 through 4135. On the computer that did it, we can see portions of the program that produced the output. What you see, and where you see it, will vary from machine to machine and also on the number of file buffers which were allocated upon entry into BASIC. It is interesting to note that on most machines, if you have previously loaded and run several different programs and then issued the NEW command and run this program; you will find parts of all of those previous programs still floating around in memory. Mostly, you will see literal string values, some commands and a few subscripted variable names. Many of the parts of a program (like the line numbers) are compressed and are no longer recognizable in ASCII. One other note: trying to see memory locations above 32767 will result in an overflow error.

Since most machines are different we cannot make a unique sample run for you to follow. Your output, though, should resemble that in the sample run to a large degree. In any case, you can now rummage around in memory and see what's there, and in the meantime, you have learned quite a bit about ASCII. ■

Network.Bas

Build your own broadcasting empire

Staff adaptation. This is an old favorite, redone in a new setting. It was originally called "Startraders" and is in public domain. We are not sure who the author is, but think it may be S. J. Singer. Whoever it was, they did a nice job. The program works on most machines using Microsoft BASIC.

Network is a game for two to four players in which the object is to amass the greatest wealth in 48 turns of play. You do it by establishing broadcasting networks in a rectangular country which is represented by a 9 x 12 grid.

Upon running the program, the computer picks several grid points at random to become possible headquarters cities. These then remain fixed throughout the current game. Each player receives five grid point selections in his turn. If any of the choices are at right angles to a headquarters city, a new broadcasting network can be formed. If none of the choices are near a city, an independent station may be formed and will show as a plus sign. Getting another space at right angles to an independent station will form a network, but will not produce as much revenue as will one connected to a headquarters city. (See sidebar for the rules of the game.)

This program is a close adaptation of one called "Startraders" which is in public domain and has been around for several years. We first saw it back in 1978, and it was running on a CP/M system. We would very much like to give the original author credit, but so far, have been unable to determine who that is.

This is a rather lengthy program with subroutines which are nested. Since this game is intended purely for your summertime enjoyment we will dispense with the exacting details. In their place, however, we will provide an overview of the program and some tips which should help you change it to whatever you like. The program as listed will run on MS-DOS machines without change. We have not checked it on all versions of MBASIC running under CP/M, but suspect it should perform well there too. In addition, we have checked it on TRS-80 Models III and IV (see box for changes for those two machines.)

Variable P1 is the number of players. Line 710 checks for end of game. The routine at 740 is to select random numbers and is called from 770, 790, 850, 860, 870, 880, 890, 900, 910 and 920.

The routine at line 950 is used to show the selected choices of moves on the screen. Connected with this, at line 1150, is the "I don't understand you" routine.

The routine which comes into play when a new company is formed is at line 1470. The routine which checks for a stock split (when the stock of a company reaches $3000 per share) is at line 1730.

The routine that determines the amount of dividends paid each player is at 1740, and the code starting at line 1800 shows you the amount of cash you have and presents the choices for buying stock.

Probably the most used subroutine in the program is at line 2050. This one re- displays the updated map after each move.

The merger subroutine is located at 2520. The routine at line 3110 prints the stock split information on the screen. At the end of the game, the final standings are handled by the routine at line 3230, while the routine at 3370 prints the special announcement heading when needed. The "Game is over" routine is located at line 3400.

Modifications you can make

In our version, we have picked five fictitious names with familiar sounding initials. You can put your own creativity to work here and make them anything you like. The company names are in lines 150 through 190. Be sure to leave the space after each name so that it will not run into other messages when printed on the screen. If you should happen to change the names and they start with different letters than ours, put the first letter

text continues on page 31

Network Takeover

The rules of the game for Network.Bas

RULES FOR NETWORK TAKEOVER

The object of the game is to amass the greatest amount of money. This is accomplished by establishing large broadcasting networks, and purchasing stock in the companies that control other networks. During the course of the game, stock appreciates in value as the networks become larger. Smaller networks can be merged into larger ones, and the stock in the smaller company is converted into stock in the larger one.

At each playing turn the computer will present the player with five prospective spaces to occupy on a 9 x 12 grid (which represents the entire country.) The player, after examining the map of the country to decide which space he wishes to occupy, responds with the row and column of that space (i.e., 1E, 8A, etc.)

There are four possible moves a player can make:

1. He can establish an independent station (shown as a + sign) if he selects a space that is not adjacent to a Headquarters city (shown as an *) or another independent station.

2. He can add to an existing network if he selects a space adjacent to an existing network. The space he selects will then be added to that network and will be designated with the first letter of that network's name. If there are any Headquarters cities or independent stations also adjacent to the selected space, they too, will be incorporated into the existing network. Each new space adjacent to a Headquarters city adds $500 per share. Each new independent station adds $100 per share to the market value of that company.

3. He may establish a new network. If there are less than five existing networks the player may, given the proper space to play, establish a new network. He may do this by occupying a space that is not adjacent to an existing network, but is adjacent (but only at right angles) to a Headquarters city or an independent station. If he establishes a new network he is automatically issued five shares in the new company. He may then proceed to buy stock in any active company including the one just formed.

4. He may force a merger of two existing networks. If there is only one space separating two networks and that space is chosen, a merger occurs. The larger network will take over the smaller (if both are exactly the same size, the survivor is determined by the alphabetical order of the company names; the earlier one survives.) The stock of the surviving company is increased in value according to the number of spaces and Headquarters cities added to its network. Each player's stock in the defunct company is exchanged for shares in the survivor on a ratio of two for one. Also, each player is paid a cash bonus proportional to the percentage of outstanding stock he held in the defunct network. Note that after a network becomes defunct through the merger process it can reappear elsewhere on the board when, and if, a new company is established.

At each turn, the computer adds stock dividends to each player's cash on hand (5% of the market value of the stock in his possession) and offers him the opportunity to purchase stock in any of the active networks on the board. Stock may not be sold, but the market value of each player's stock is taken into account at the end of the game to determine the winner.

If the market value of a given stock exceeds $3000 at any time during the game, that stock will split two for one, the price is cut in half and the number of shares owned by each player is doubled.

During each player's turn, the number of shares he owns in the various networks is shown to the right of the map. He may, when it is his turn, type "Stock" (or S) and see a complete breakdown of the price and value of his stock. From the stock display he may type "Map" (or M) to get back to the map.

The game ends and a general accounting is made on the 48th turn, regardless of how many players there were.

of each in line 260, where the ACNPT initials are now. You may also need to change the abbreviations for the company names in lines 2190 through 2230. Keep them short so that they will fit on the screen along with the map. The word "network" appears only once, in line 1490. "Broadcasting" appears in lines 1540, 1850, 2640, 3160 and 2060.

When you first run the program, line 430 picks both the number and location of the headquarters cities. If you want more (or less) cities on startup you can change this line. For example, picking one number from a possible fifty numbers would give very few cities, while picking two out of 25 (as it does now) gives an average of about six to eight. We found the game to be most interesting with about 4 or 5 cities.

The number of turns to play is set in line 710, denoted by variable K. The original authors seem to have picked this number about right. With more, the map fills up. With less, you don't have the opportunity to force the most interesting mergers and splits that naturally occur from about the 35th to 48th moves. By the way, it is possible to force two mergers with one space, and it does happen occasionally. When that happens, it is usually followed by a stock split in one or more companies.

The value of stock of a company established next to a headquarters city is $500. This is set in line 1580. The value of stock for companies formed from independent stations is $100, and appears in lines 230, 1390, 1590, 1620, 1650, 1680, 3030 and 3280. The amount of dividends paid each player is currently 5% and is set only once, in line 1750.

After you have made your move, the program presents you with the opportunity to buy stock in the various companies. As the program now stands, it will not show you the companies for which you do not have enough money to buy a share. If you are broke, it will show you no companies at all. If you would like to see them anyway, put a remark at the beginning of line 1790. Incidentally, you do not have to enter zero to buy no shares, pressing RETURN will suffice.

If you would like the stock of a company to split 2 for 1 at a different value than $3000, change it in line 1710 and 2980.

If you do not want to see the Portfolio and the companies along with the map, remove the word "Portfolio" from line 2100 and delete lines 2190 through 2240.

This is a fun program to play with and modify. About the only thing we would like to see added is the ability to buy a specific space (at a very high price, of course). It would be interesting because cash draws no interest, while stock appreciates, so holding your cash to buy a space could either make it big or lose it all for you. In any case, enjoy the game! ●

Changes for TRS-80 Models III/IV

Here are changes to make for both Model III and IV TRS- 80:
Remove the remark in line 120.
Remove (or remark) line 420.
Change the following lines:

```
430 IF RND(25)<>10 AND RND(25)<>5 THEN
M(I,J)=1 ELSE M(I,J)=3
580 I=RND(P1)
740 R(I)=RND(8)+1
750 C(I)=RND(11)+1
```

In addition, for Models III and other computers that use 64 column, 16 line screens:
Remove (or remark) lines 2070 and 2090.
In line 2100, remove the space between the first quote and the capital letter A. And that's it.

Note: Because there are so few changes, the version of this program on the CodeWorks download system will be as in the listing accompanying this article.

```
100 REM ** NETWORK.BAS ** AN ADAPTATION OF A PUBLIC DOMAIN PROGRAM
110 REM ** ORIGINALLY ENTITLED 'STAR-TRADERS'**
120 'CLEAR 1000 : ' USE ONLY IF YOUR MACHINE NEEDS TO CLEAR SPACE
130 DEFINT C,I
140 DIM M(10,13),S(5,4),N$(5),D1(5),S1(5),Q(5),M$(12),C$(25),C1$(25),
    C2$(25)
150 DATA 1,"AMERICUS "
160 DATA 2,"COLUMBUS "
170 DATA 3,"NATIONUS "
180 DATA 4,"PUBLICUS "
190 DATA 5,"TURNICUS "
```

```
200 CLS
210 FOR I=1 TO 5
220   FOR J=1 TO 4
230     S(I,J)=0:D1(T)=0:S1(I)=100:Q(I)=0:B(I)=6000
240   NEXT J
250 NEXT I
260 L$=".+*ACNPT"
270 M$="ABCDEFGHIJKL"
280 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
290 PRINT "              N E T W O R K   T A K E O V E R"
300 PRINT "           or, how to make it big in broadcasting"
310 PRINT STRING$(60,"-")
320 PRINT
330 PRINT"Build your own broadcasting network. Buy stock in any"
340 PRINT"other network. Force mergers and stock splits. When it"
350 PRINT"is your turn you may type S to see your stock. Typing M
360 PRINT"will return you to the map. During your turn, the number"
370 PRINT"of shares you own are indicated at the right of the map."
380 PRINT"There are 48 turns to play. Good luck."
390 PRINT
400 FOR I=1 TO 9
410   FOR J=1 TO 12
420   RR$=MID$(TIME$,4,2)+MID$(TIME$,7,2):RR=VAL(RR$):RANDOMIZE RR
430   IF INT(RND*25+1)<>10 AND INT(RND*25+1)<>5 THEN M(I,J)=1 ELSE
      M(I,J)=3
440   NEXT J
450 NEXT I
460 INPUT"HOW MANY PLAYERS (2-4) ";P1
470 IF P1<2 OR P1>4 THEN 460
480 PRINT
490 FOR I=1 TO P1
500   PRINT"PLAYER ";I;
510   INPUT"WHAT IS YOUR NAME ";P$
520   IF I=1 THEN P1$=P$
530   IF I=2 THEN P2$=P$
540   IF I=3 THEN P3$=P$
550   IF I=4 THEN P4$=P$
560 NEXT I
570 PRINT TAB(10);"... NOW I WILL DECIDE WHO GOES FIRST ":PRINT
580 I=INT(RND*P1+1)
590 GOSUB 610
600 GOTO 670
610 PRINT
620 ON I GOTO 630,640,650,660
630 PRINT P1$;:P5$=P1$:RETURN
640 PRINT P2$;:P5$=P2$:RETURN
650 PRINT P3$;:P5$=P3$:RETURN
660 PRINT P4$;:P5$=P4$:RETURN
670 PRINT" GETS THE FIRST MOVE."
680 FOR W=1 TO 2000:NEXT W
690 K=1
700 P=I:GOTO 730
710 K=K+1:IF K>48 THEN GOTO 3400
```

```
720 P=P+1:IF P=P1+1 THEN P=1
730 FOR I=1 TO 5
740   R(I)=INT(RND*9+1)
750   C(I)=INT(RND*12+1)
760   FOR I1=I-1 TO 0 STEP -1
770     IF R(I)=R(I1) AND C(I)=C(I1) THEN GOTO 740
780   NEXT I1
790   IF M(R(I),C(I))>1 THEN GOTO 740
800   FOR I1=1 TO 5
810     IF Q(I1)=0 THEN GOTO 930
820   NEXT I1
830   IF M(R(I),C(I)+1)>3 OR M(R(I),C(I)-1)>3 OR M(R(I)+1,C(I))>3 OR
      M(R(I)-1,C(I))>3 THEN GOTO 930
840   A1=M(R(I),C(I)+1):A2=M(R(I),C(I)-1):A3=M(R(I)+1,C(I)):A4=M(R(I)-
      1,C(I))
850   IF A1=2 AND A2<4 AND A3<4 AND A4<4 THEN GOTO 740
860   IF A2=2 AND A1<4 AND A3<4 AND A4<4 THEN GOTO 740
870   IF A3=2 AND A1<4 AND A2<4 AND A4<4 THEN GOTO 740
880   IF A4=2 AND A1<4 AND A2<4 AND A3<4 THEN GOTO 740
890   IF A1=3 AND A2<4 AND A3<4 AND A4<4 THEN GOTO 740
900   IF A2=3 AND A1<4 AND A3<4 AND A4<4 THEN GOTO 740
910   IF A3=3 AND A1<4 AND A2<4 AND A4<4 THEN GOTO 740
920   IF A4=3 AND A1<4 AND A2<4 AND A3<4 THEN GOTO 740
930 NEXT I
940 GOSUB 2050
950 I=P
960 GOSUB 610
970 PRINT", Here are your choices for turn";K
980 FOR I=1 TO 5
990   PRINT R(I);MID$(M$,C(I),1);" ";
1000 NEXT I
1010 PRINT
1020 INPUT"WHAT IS YOUR MOVE ";R$
1030 IF LEN(R$)=0 THEN R$="S"
1040 IF LEFT$(R$,1)="M" THEN R$="" ELSE GOTO 1070
1050 GOSUB 2050
1060 GOTO 950
1070 IF LEFT$(R$,1)="S" THEN R$="" ELSE GOTO 1100
1080 GOSUB 3230
1090 GOTO 950
1100 IF LEN(R$)<>2 THEN GOTO 1150
1110 IF ASC(MID$(R$,2,1))-64<1 THEN GOTO 1150
1120 IF ASC(MID$(R$,2,1))-64>12 THEN GOTO 1150 ELSE GOTO 1160
1130 IF VAL(R$)<1 THEN GOTO 1150
1140 IF VAL(R$)>9 THEN GOTO 1150
1150 PRINT"I don't understand, TRY AGAIN.":GOTO 1020
1160 R=VAL(LEFT$(R$,1))
1170 C=ASC(RIGHT$(R$,1))-64
1180 FOR I=1 TO 5
1190   IF R=R(I) AND C=C(I) THEN GOTO 1230
1200 NEXT I
1210 PRINT"THAT SPACE WAS NOT INCLUDED IN THE LIST"
1220 GOTO 1020
```

```
1230 A1=M(R-1,C):A2=M(R+1,C):A3=M(R,C+1):A4=M(R,C-1)
1240 IF A1<=1 AND A2<=1 AND A3<=1 AND A4<=1 THEN M(R,C)=2 ELSE GOTO
     1260
1250 GOTO 1740
1260 IF A1>3 AND A2>3 AND A2<>A1 THEN GOSUB 2270
1270 IF A1>3 AND A3>3 AND A3<>A1 THEN GOSUB 2270
1280 IF A1>3 AND A4>3 AND A4<>A1 THEN GOSUB 2270
1290 IF A2>3 AND A3>3 AND A3<>A2 THEN GOSUB 2270
1300 IF A2>3 AND A4>3 AND A4<>A2 THEN GOSUB 2270
1310 IF A3>3 AND A4>3 AND A4<>A3 THEN GOSUB 2270
1320 IF A1<4 AND A2<4 AND A3<4 AND A4<4 THEN GOTO 1420
1330 IF M(R,C)>3 THEN GOTO 1740
1340 IF A1>3 THEN I=A1-3
1350 IF A2>3 THEN I=A2-3
1360 IF A3>3 THEN I=A3-3
1370 IF A4>3 THEN I=A4-3
1380 Q(I)=Q(I)+1
1390 S1(I)=S1(I)+100
1400 M(R,C)=I+3
1410 GOTO 1580
1420 FOR I=1 TO 5
1430   IF Q(I)=0 THEN GOTO 1470
1440 NEXT I
1450 IF M(R,C)<3 THEN M(R,C)=2
1460 GOTO 1740
1470 CLS
1480 GOSUB 3370
1490 PRINT" A new Network has been formed!"
1500 PRINT" Its name is ";
1510 RESTORE
1520 READ N,C$
1530 IF I<>N THEN GOTO 1520
1540 PRINT C$;"Broadcasting"
1550 S(I,P)=S(I,P)+5
1560 Q(I)=1
1570 PRINT :PRINT
1580 IF A1=3 OR A2=3 OR A3=3 OR A4=3 THEN S1(I)=S1(I)+500
1590 IF A1=2 THEN S1(I)=S1(I)+100 ELSE GOTO 1620
1600 Q(I)=Q(I)+1
1610 M(R-1,C)=I+3
1620 IF A2=2 THEN S1(I)=S1(I)+100 ELSE GOTO 1650
1630 Q(I)=Q(I)+1
1640 M(R+1,C)=I+3
1650 IF A3=2 THEN S1(I)=S1(I)+100 ELSE GOTO 1680
1660 Q(I)=Q(I)+1
1670 M(R,C+1)=I+3
1680 IF A4=2 THEN S1(I)=S1(I)+100 ELSE GOTO 1710
1690 Q(I)=Q(I)+1
1700 M(R,C-1)=I+3
1710 IF S1(I)>=3000 THEN T1=I ELSE GOTO 1730
1720 GOSUB 3110
1730 M(R,C)=I+3
1740 FOR I=1 TO 5
```

```
1750   B(P)=B(P)+INT(.05*S(I,P)*S1(I))
1760 NEXT I
1770 FOR I=1 TO 5
1780   IF Q(I)=0 THEN GOTO 2030
1790   IF B(P)<S1(I) THEN GOTO 2030
1800   PRINT:PRINT "Your current cash = $";B(P);
1810   PRINT:PRINT "Buy how many shares of ";
1820   RESTORE
1830   READ N,C$
1840   IF I<>N THEN GOTO 1830
1850   PRINT C$;"Broadcasting";
1860   PRINT" AT $";S1(I)
1870   PRINT" You now own ";S(I,P);" (and can afford to buy ";
       INT(B(P)/S1(I));" )";
1880   INPUT R3$:IF LEN(R3$)=0 THEN R3$="0"
1890   IF R3$(1,1)="M" THEN R3$="" ELSE GOTO 1920
1900   GOSUB 2050
1910   GOTO 1800
1920   IF R3$(1,1)="S" THEN R3$="" ELSE GOTO 1950
1930   GOSUB 3230
1940   GOTO 1800
1950   R3=VAL(R3$)
1960   R3$=""
1970   IF R3*S1(I)<=B(P) THEN GOTO 2000
1980   PRINT"You only have $";B(P);" TRY AGAIN"
1990   GOTO 1800
2000   IF R3=0 THEN GOTO 2030
2010   S(I,P)=S(I,P)+R3
2020   B(P)=B(P)-(R3*S1(I))
2030 NEXT I
2040 GOTO 710
2050 CLS
2060 PRINT TAB(22);"THE TOTAL BROADCAST AREA"
2070 PRINT
2080 PRINT TAB(10);"* = Possible Hq. Cities, + = Independent Stations
2090 PRINT
2100 PRINT TAB(13);" A  B  C  D  E  F  G  H  I  J  K  L   Portfolio"
2110 FOR R2=1 TO 9
2120   PRINT"        ";R2;" ";
2130   FOR C2=1 TO 12
2140     PRINT" ";
2150     Z2=M(R2,C2)
2160     IF Z2=0 THEN Z2=Z2+1
2170     PRINT MID$(L$,Z2,1)" ";
2180   NEXT C2
2190   IF R2=1 THEN PRINT "  ABC  ";S(1,P)
2200   IF R2=3 THEN PRINT "  CBS  ";S(2,P)
2210   IF R2=5 THEN PRINT "  NBC  ";S(3,P)
2220   IF R2=7 THEN PRINT "  PBS  ";S(4,P)
2230   IF R2=9 THEN PRINT "  TBS  ";S(5,P)
2240   IF R2=2 OR R2=4 OR R2=6 OR R2=8 THEN PRINT
2250 NEXT R2
2260 RETURN
```

```
2270 F1=A1-3:IF F1<0 THEN F1=0
2280 F2=A2-3:IF F2<0 THEN F2=0
2290 F3=A3-3:IF F3<0 THEN F3=0
2300 F4=A4-3:IF F4<0 THEN F4=0
2310 T=Q(F1)
2320 T1=F1
2330 IF Q(F2)>Q(F1) THEN T=Q(F2) ELSE GOTO 2350
2340 T1=F2
2350 IF Q(F3)>T THEN T=Q(F3) ELSE GOTO 2370
2360 T1=F3
2370 IF Q(F4)>T THEN T=Q(F4) ELSE GOTO 2390
2380 T1=F4
2390 IF F1=T1 OR A1<4 THEN GOTO 2420
2400 X=F1
2410 GOSUB 2520
2420 IF F2=T1 OR A2<4 THEN GOTO 2450
2430 X=F2
2440 GOSUB 2520
2450 IF F3=T1 OR A3<4 THEN GOTO 2480
2460 X=F3
2470 GOSUB 2520
2480 IF F4=T1 OR A4<4 THEN GOTO 2510
2490 X=F4
2500 GOSUB 2520
2510 RETURN
2520 CLS
2530 GOSUB 3370
2540 RESTORE
2550 READ N,C$
2560 IF X<>N THEN GOTO 2550
2570 C1$=C$
2580 PRINT C1$;
2590 PRINT" has just been merged into ";
2600 RESTORE
2610 READ N,C$
2620 IF T1<>N THEN GOTO 2610
2630 C2$=C$
2640 PRINT C2$;"Broadcasting !"
2650 PRINT"Please note the following transactions"
2660 PRINT
2670 PRINT TAB(3);"OLD STOCK = ";C1$; "     NEW STOCK = ";C2$
2680 PRINT
2690 PRINT"PLAYER";TAB(10);"OLD STOCK";TAB(22);"NEW STOCK";
2700 PRINT TAB(34);"TOTAL HOLDING";TAB(53);"BONUS PAID"
2710 FOR I=1 TO P1
2720   GOSUB 610
2730   PRINT TAB(10);S(X,I);TAB(22);INT((.5*S(X,I))+.5);
2740   PRINT TAB(34);S(T1,I)+INT((.5*S(X,I))+.5);
2750   X1=0
2760   FOR I1=1 TO P1
2770     X1=X1+S(X,I1)
2780   NEXT I1
2790   PRINT TAB(53);" $";INT(10*((S(X,I)/X1)*S1(X)))
```

```
2800 NEXT I
2810 FOR I=1 TO P1
2820   S(T1,I)=S(T1,I)+INT((.5*S(X,I))+.5)
2830   B(I)=B(I)+INT(10*((S(X,I)/X1)*S1(X)))
2840 NEXT I
2850 FOR I=1 TO 9
2860   FOR J=1 TO 12
2870     IF M(I,J)=X+3 THEN M(I,J)=T1+3
2880   NEXT J
2890 NEXT I
2900 A1=M(R-1,C):A2=M(R+1,C):A3=M(R,C+1):A4=M(R,C-1)
2910 F1=A3-3
2920 IF F1<0 THEN F1=0
2930 F2=A2-3
2940 IF F2<0 THEN F2=0
2950 Q(T1)=Q(T1)+Q(X)
2960 S1(T1)=S1(T1)+S1(X)
2970 FOR W=1 TO 5000:NEXT W
2980 IF S1(T1)=>3000 THEN GOSUB 3110
2990 F3=A3-3
3000 IF F2<0 THEN F3=0
3010 F4=A4-3
3020 IF F4<0 THEN F4=0
3030 S1(X)=100
3040 Q(X)=0
3050 FOR I=1 TO P1
3060   S(X,I)=0
3070 NEXT I
3080 PRINT :PRINT
3090 M(R,C)=T1+3
3100 RETURN
3110 GOSUB 3370
3120 PRINT"THE STOCK OF ";
3130 RESTORE
3140 READ N,C$
3150 IF T1<>N THEN GOTO 3140
3160 PRINT C$;"Broadcasting has SPLIT 2 for 1 !!"
3170 S1(T1)=INT(S1(T1)/2)
3180 PRINT :PRINT
3190 FOR I1=1 TO P1
3200   S(T1,I1)=2*S(T1,I1)
3210 NEXT I1
3220 RETURN
3230 CLS
3240 PRINT
3250 PRINT "STOCK";TAB(30);"PRICE PER SHARE";
3260 PRINT TAB(50);"YOUR HOLDINGS"
3270 FOR I3=1 TO 5
3280   IF S1(I3)=100 THEN GOTO 3340
3290   RESTORE
3300   READ N,C$
3310   IF I3<>N THEN GOTO 3300
3320   PRINT C$;
```

```
3330    PRINT TAB(30);S1(I3);TAB(50);S(I3,P)
3340 NEXT I3
3350 RESTORE
3360 RETURN
3370 CLS
3380 PRINT TAB(22);"SPECIAL ANNOUNCEMENT!!":PRINT
3390 RETURN
3400 CLS
3410 PRINT TAB(10);"THE GAME IS OVER - HERE ARE FINAL STANDINGS"
3420 PRINT
3430 PRINT"PLAYER";TAB(10);"CASH VALUE OF STOCK";TAB(33);"CASH ON
     HAND";
3440 PRINT TAB(50);"NET WORTH"
3450 PRINT
3460 FOR I=1 TO P1
3470    FOR J=1 TO 5
3480       D1(I)=D1(I)+(S1(J)*S(J,I))
3490    NEXT J
3500 NEXT I
3510 FOR I=1 TO P1
3520    GOSUB 610
3530    PRINT TAB(10);"$";D1(I);TAB(33);"$";B(I);
3540    PRINT TAB(50);"$";D1(I)+B(I)
3550 NEXT I
3560 END
```

Programming Notes

Craig W. Hartsell, of Falls Church, Virginia, sent along this routine that can be used to address envelopes. We like it and think it is ripe for embellishment. He says: "Many times in using a computer in business or technical calculations the requirement for making a label or addressing an envelope forces a resort use of a typewriter or a word processor. The use of a typewriter is a typical solution but is notoriously error-prone, especially for non-secretaries. The other solution of using a word processor is overwhelming the task with power resulting in a substantial loss of time in setting up and shutting down the software for an essential but still minor task. In the case where the demand is repetitive, another solution often adopted is the use of pre-addressed gummed labels. Unfortunately for most professional business activities, this is a mite tacky and to be avoided if at all possible. The Address.Bas program is designed to cover the above circumstances and to quickly produce a professional looking label or addressed envelope. By loading the program on the DOS or other system disk it is readily accessible. After loading and listing the user is presented with four data entry lines (50-80) providing a four line address possibility. An error routine is provided to enforce a limited line length. This basic approach is easily modified for different needs including storing commonly occurring addresses, adding more lines, movement of the typing to different physical locations and the like."

```
10 ' ADDRESS.BAS CRAIG W. HARTSELL
20 ' USE YOUR PRINTER TO ADDRESS ENVELOPES
30 ' KEEP EACH LINE BETWEEN THE QUOTES
40 CLS
50 X$="YOUR NAME GOES HERE       ":GOSUB 100
60 X$="YOUR ADDRESS GOES HERE    ":GOSUB 100
70 X$="CITY STATE ZIP GO HERE    ":GOSUB 100
80 X$="                          ":GOSUB 100
90 END
100 X=LEN(X$):IF X>25 THEN GOTO 130
110 PRINT X$
120 RETURN
130 PRINT"LINE TOO LONG "
140 PRINT"":LIST
```

CodeWorks Helpline

As many of you already know, we are always willing to help if you have problems getting our programs to run. Sometimes though, the exchange of correspondence seems to go like this:

"I have a problem. Your program errors in line xxx."

"Please send us a listing of the program as you typed it in. What is the error message?"

"I think it is a syntax error. How can I send the listing if the program will not run?"

"Load the program, then turn on your printer and type LLIST."

Well, that may be a slight exaggeration, but it points out the problem in communicating. If you send us a problem, please send a paper listing of the program the way you typed it in so we can check it against the original. It would be very helpful if you have not renumbered the program or made extensive additions and deletions to it. We don't always know what you were trying to do. Also tell us the type of computer and operating system you are using, and if the problem is with printing, also tell us what kind of printer.

We get quite a few that are simply typing errors. These will usually give you a syntax error (but not always.) When you get a syntax error, check your spelling and punctuation. Then if you can't find it, have someone else check it. We all find it easy to overlook our own mistakes. Watch for "I" and "1" interchanges, also for zero and oh.

Another easy to overlook error is the trailing semicolon. It keeps the cursor (or the printhead on your printer) at the end of the current line for something to follow on that same line. If the semicolon is not there, the cursor (or printer) will linefeed to the next line. The trailing comma, not used that much, has the same effect.

Because of the large and varying number of machines and printers out there today, we probably will not be able to answer every question you send us. Our batting average is pretty good up until now though, and we will be more than happy to give your problem our undivided attention if you can't find it yourself. ●

Download

What's Happening on the Download

When are we going to get 1200 baud on the download? Soon. We have not been dragging our feet on purpose. In fact, we have tried several 300/1200 modems and found them lacking. Some would require a change in the way we handle things. One thing we do not want is to change the procedure in mid-stream.

One problem with getting 1200 on line is that when someone hangs up without signing off we stay on line, giving a busy signal to anyone else wishing to call. The Hayes Smartmodem 300 we now use handles that nicely, as does the software supporting it. Since we have identified the problem and have a Hayes 300/1200 baud modem on the way, we expect to have it operating before you receive the next issue.

Here are some of the messages from the download: One asked why we don't ask which charge card you are using. We do not need to. The number itself identifies the company. All Master Cards start with 5xxx and all VISA cards start with 4xxx. (Although we do not accept American Express, they all start with 3xxx.)

"I am having problems signing on as a subscriber.

I am still having problems signing on as a subscriber.

I still can't seem to sign on as a subscriber.

Disregard previous messages, I just fixed the nut on the keyboard."

Some simply tell us you like the board.

"I tried the download last week and want to tell you it worked just great. I have had trouble with other setups and this was a pleasant surprise."

And this happens more often than we thought it would:

"I have forgotten my password. Please reset me."

Some point out problems or improvements we can make:

"Good board. The only problem I noted was the fact that once you have downloaded a given file, if you should make the mistake of hitting RETURN rather than a valid entry, you restart the download of the file just received. I could find no way to terminate except to hang up."

Then there are these:

"I am having a ball with the downloads." and, "No real message, just testing my new modem." and, "I think the idea of having the programs on diskette is great and I hope you follow through on the idea."

And so it goes. It's fun reading the mail, and we thought you might like to see a small sample of it. And yes, we are working on the diskette idea. ●

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

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

CodeWorks

Issue 7 September/October 1986

CONTENTS

CODEWORKS

Issue 7 Sep/Oct 1986

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

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

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

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

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

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

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

Editor's Notes

***CodeWorks* has turned out to be a slightly smaller, but certainly more dedicated, group than we had first envisioned.**

This issue marks the end of our first year of publication. It has been an interesting year, full of anticipation and surprises.

We thank each of you who joined the pioneer spirit of a magazine that dares to live on reader support alone. We think the experiment worked, and hope you have benefited from it.

We set our orginal goals high, and then reached 70 percent of our subscription goal. Not bad for a first year startup. Rest assured, that with your continued support and encouragement, *CodeWorks* will continue.

See the inside of the back cover for renewal information. Please note that we would not be the least bit offended if you encouraged your computing friends to join us.

Thank you for making the experiment work and worthwhile. As always, we intend to be here to serve only you, the readers and users of these newfangled machines called "Computers."

Irv

Forum

An Open Forum for Questions & Comments

When I get a new issue of CodeWorks I punch three holes in it so I can keep it in a three-ring binder. It is always easy to find that way. I notice that the center margins have been getting a little narrower lately. So far, I haven't punched out any of the text, but it has been close at times. I hope you will continue to leave me plenty of room.

I have been getting a lot of use from Card/Bas. The next job I am going to put it to is to catalog all the programs in your fine magazine, as well as a couple others I receive. How about publishing some of the modifications you have no doubt received for that program.

The program "Network/Bas" ran beautifully. I made over three million dollars on one of the first plays! Too bad it wasn't for real.

Your reply to J. L. Lopez on page 5 gave me an inspiration: Why don't you *stop* using the letters "I" and "O" as variables. They make it *easier* to make a mistake!

Samuel Laswell
South Haven, MI

In Issue 6 you printed a great game program I have already enjoyed several hours playing, however, not without making some necessary corrections. The first was in your changes for the TRS-80 Model III. When using your changes for lines 740 and 750, I found that you will never see a choice of either a position in row 1 or in column A. Instead of: 740 R(I)=RND(8)+1 and 750 C(I)=RND(11)+1 they should read: 740 R(I)=RND(9) and 750 C(I)=RND(12). The other errors were: Line 2910 should read F1=A1-3, not F1=A3-3 and line 3000 should read IF F3<0 THEN F3=0, not IF F2<0 THEN F3=0...

...I am writing this letter using Maker.Bas as mentioned in a letter to your from Walt King III...

Keep up the super job you're doing. As always we will be anxiously awaiting the next issue.

Bob Anderson
Orland Park, IL

We ported Network over from an MS-DOS machine to the Model III and played the program all afternoon, making sure it worked - and never caught the errors you found. Talk about not seeing! You are absolutely right on all counts.

Yours was the first letter I pulled out of the folder yesterday morning as I sat down to do these. Your comments about using Maker.Bas to write a letter had me puzzled. I never looked at anything else for the rest of the day, instead, I got out Maker.Bas and started re-writing it to make it into a simple text editor. Judging from the way you hand-lettered the quote marks into your letter, you apparently ran into the same problem I did. It worked for everything except the quote marks. At any rate, I finished the day having written QTEXT.BAS - a quick and simple text editor, and an article to go along with it. It will appear in some, yet undetermined, future issue of CodeWorks. Thank you for the nudge - and why didn't you send in your version? You could have collected our modest author fee for it.

Thanks for the Network.Bas program in your (Issue 6). It runs well, but I have a few questions.

Line 790 should be less than, otherwise an endless loop.

Line 420 uses RR, why? I can't find it in the program, and why randomize time$ in a loop?

Line 1750: B is not dimensioned as an array, but it still works. B(P) takes the last value in the loop, so why use a loop?

...Keep up the good work. Hope the magazine has enough subscribers to keep it going. You should let the readers know about that.

A. W. Wardell
Pawtucket, RI

Line 790 is correct. It looks at the random selections for map positions, and if the value at any position is greater than 1 it means that position already contains either an HQ city or an independent station. In that case, it goes back to pick a different random number for the space.

In line 420, RR is a dummy variable required by the RANDOMIZE statement in many BASIC's. If the RR were not present, the program would stop and ask for a "seed" for the random generator. Some BASIC's allow the use of the RANDOMIZE statement all by itself. Most MS-DOS/GW BASIC machines require the argument. The reason for using TIME$ in a loop is to get a different seed number for the random generator each time a random number is picked. Without it, you would get the same random sequence repeatedly. Minutes and seconds are pulled out of TIME$ to get different seed numbers.

In line 1750, B(P) does not need to be dimensioned since the subscript will never be more than five. Most BASIC's allow up to 11 (0 through 10) subscripts without the DIM statement. (See Beginning BASIC, this issue.) It does not take the last value in the loop, it ***accumulates*** *the values in S(I,P) and S1(I).*

Although I have not written it yet, I am sure that we will talk about the future of CodeWorks elsewhere in this issue.

I like your magazine and wish you luck! The man who was complaining about cost at $2 per issue apparently cannot divide. It amounts to $4 per issue. Why not accept advertising but put it all in the rear of the magazine. Do not make us hunt and peck to find meat and potatoes. If anyone wants to read the advertisements, let them. I read *Computer Shopper* but get mad trying to find the wheat with so much chaff mixed in with it.

A. H. Smith
Gainesville, FL

An editor I once knew told me that we Americans tend to buy magazines by the slick-advertising-pound. The information they contain usually takes a back seat to the main reason they exist - to provide a vehicle with which to sell something. We enjoy the fact that CodeWorks exists solely for the readers who support it, with no outside influences and no ulterior motives.

As an old boatbuilder/carpenter/draftsman I've often wrestled with the "best fit" problems raised by Wood.Bas which appeared in your Issue 3. That program is so well worded and illustrated that I'm tempted to drop all other projects and try it out!

Congratulations on producing a really novel type of magazine that entertains while it educates. One of *its best features* is your total elimination of advertising clutter which always distracts from readability.

Lindsay A. Fowler
Ft. Lauderdale, FL

(Re: Forum, Issue 6) Howard M. Cruff wrote to say he has a Model 12 and asks about PEEK and POKE. I use a Model 4 (Tandy), but formerly did quite a lot of work on a Tandy Model II. In Rosenfelder's *"BASIC Faster and Better"*, there are many routines for Models I, III and IV with notes in the margins regarding changes to make the routines run on the Model II. Also, starting on page 243 (of that book) there is a whole section on Model II Modifications with PEEK and POKE for the Model II as the first subject discussed. Apparently Howard hasn't seen this book, and while he might not want to spring for the $29.95 price at Radio Shack, he might be able to borrow one. It is well worth the price, however.

My compliments to all of you - every issue has got some real goodies in it. Keep it up.

Dexter Walker
Birmingham, AL

Received a copy of your publication. I liked what I saw and took the time to read it from cover to cover. Still liked what I saw. Hand loaded the Extract and Search programs. Neither ran - programming errors in both. Re: Extract, line 320, this jump puts the program in a loop - must be removed. Re: Search, line 340, colon after Y/N should be a quote. Colon produces an error when run. These problems seem to me to be editing problems...

...So very tired of trying to use published (magazine) programs, they almost never work and have to be debugged. It is easier to write the things myself. Thought you'd like to know.

F. Bullock
Gales Ferry, CT

You are right about the missing quote in Search.Bas, and it was an editing mistake. But the quote does not go in place of the colon, it should go in place of the semicolon. You are wrong about the loop in Extract.Bas. It works as written.

What I would like to see in your magazine are some examples of how to use an array, both variable and string, saved to disk using the For...Next statement and then how to load the same from disk using the sequential file OPEN"O" and OPEN"I". This procedure is kept quiet since I can't find any information in any manuals. I'm sure others could benefit from this information.

Raymond J. Jasek
Spooner, WI

Well, let's see...

Try issue 2, page 22, lines 160 to 310 and lines 330 to 490. Issue 4, page 15, Article on sequential files, pages 37 and 38, lines 2740 through 3220. Issue 5, Merge/Sort, page 20 and Convert.Bas, page 34. Then try this issue, the NFL programs both use sequential files and arrays.

I have tried to use the Card.Bas program and after entering 10 files (*records?*) and attempting to enter the 11th I get "subscript out of range in 880". I do not see where I keep getting this error message. I would like to work out this bug because this program has great potential for me. I would also like to see a way of printing out the labels by reversing the names and removing the comma...

Gene M. Lindley
Miami, FL

Since you said you had a Tandy Model 1000, you do not need line 105. Remove or remark it. Also make sure that V=200 in line 110. Also check to see that the last item in line 120 is P(V) and not P$(V).

A very simple way out of your name problem is to put the person's first name in field 1 and last name in field 2. Now you can sort on field 2 for an alphabetical last name sort. Then in the "Print label routine" from line 2165 on, LPRINT the first name field, follow it with a semicolon (to keep the cursor on that line), and print the last name field. Then go on to the next line to print the address and the rest of the label. Now, since you will probably not sort on the first name field, you can include "Mr." or "Ms." before the first name, and a middle initial after it. This way, you can print labels that would say, for example, Mr. John M. Doe, where the "Mr. John M." would be in field 1 and "Doe" in field 2. Make the appropriate changes to the field headings in lines 570 through 640.

In reference to Card.Bas, here is another way to initialize the file before the program is run for the first time: Load the program (don't run it), then at the Ready or OK prompt, type A$(0)="ZZZ":GOSUB 160 and press RETURN or ENTER. The program itself will then initialize the file and come up running with the menu on the screen. If you already have a Card.Dat file, DO NOT use this procedure or it will wipe your existing file off the disk and if you have no backup of it, it will be gone for good.

CodeWorks looks like an exceptional deal and the format is no (commercial) nonsense. That's why I like Public TV (and pay for it!)

Hardenbrook Mold Service
Van Nuys, CA

You couldn't have picked a better

time of the year to say that. It's pledge (renewal) time already, but unlike the Public TV channels, we only do it once per year, not four times.

I wish to compliment you on your excellent publication. I find it of value and learn much from it. The clarity of explanation in your comments concerning various aspect of a particular program are most refreshing.

I note in the Jul/Aug issue that you have some interesting things planned for the future. This is fine, and I wish you well, but please don't forget some of the older concepts you used to whet our computer appetites. I am referring specifically to the Poker program you have mentioned several times. I am very interested in software that is based on gambling games, and the promise of that program, to be released in some future issue, was one of my reasons for subscribing.

I remember other computer publications that promised discussion, or programs, on certain things, but never delivered. I lost confidence in them, as I'm sure did many other readers. Integrity seems to be the issue.

So, while I am delighted you have so many new things planned, please don't forget the older programs mentioned. Me, I'm itching to get into that Poker program. Thanks for a great and unique publication.

Arthur Melanson
Audubon, NJ

POKER.Bas will appear as the feature program in the very next issue. It was originally scheduled for this issue but because of the timing of the NFL Football season and our NFL programs in this issue, it had to be advanced to the Nov/Dec issue. We feel reasonably confident that you will like it. Also scheduled for the next issue is the report generator for Card.Bas. Following that issue we will have a random file mini-database which will allow much larger files to be handled than Card.Bas did. The correlation program we mentioned is in the checking out stages. We usually don't even mention a program until it is at least written and working. Writing a program is just the first step in the process. After that it has to be checked on other machines, then the program needs to be cleaned up and listed out for paste-up and the article written. Sample runs need to be made to include with the article when necessary. Rest assured, however, that we will publish all the programs we have promised.

I was just going through Busmod.Bas in Issue 5 and have a question perhaps you can answer. In the routine that determines Sensitivity of Data, as you are looping through and increasing each variable by 10%, I don't see the variable being reset. If it is being reset, where? If not, why does that not distort your results?

As a veteran mainframe programmer-turned-manager, I welcome your magazine as a chance to get back to the bits and bytes that I enjoy so much. I particularly like the way you present your programs in context, as solutions to real problems, not hypothetical situations. Keep up the good work.

Gail A. MacLean
Norwalk, CT

The P(X) and P1(X) arrays are filled once with values calculated from the values in the DATA statements. Since there is no accumulation of data, i.e., no P(X)=P(X)+something, any repeat running of the sensitivity would simply recalculate the same values and write them over what is already in the array. The only way to get different values would require editing the values in the DATA statements, after which a RUN command would be necessary which will clear all previous variables in any case.

I have been looking at the program in Issue 6 ... located at the bottom right of page 38. Line 130 advises when the address lines are too long. The "LINE TOO LONG!" is displayed such a short time that it cannot be read. I modified the program to overcome this difficulty and also to separate the "LINE TOO LONG" from the address. The following are the modifications:

```
130 PRINT:PRINT"LINE TOO LONG"
135 FOR T=1 TO 700
136 NEXT T
```

I enjoyed the Network program. It took a lot of careful typing to load it into my Tandy Model III. I located a few errors in my typing. Once they were corrected the program worked nicely. I have already amassed over a million dollars!

Cdr. Ralph W. Lindahl
Wenatchee, WA

Please note the comments of Bob Anderson, earlier in this Forum, concerning errors in Network and the Tandy Model III.

I forgot to ask his name, but a reader who called via telephone mentioned that the ASCII dump program in the Jul/Aug issue, page 28, could be made to double the amount of memory you could see. Change line 210 to DEFDBL A,B and change the 32767 in line 190 to 65534. If you don't have 65K of memory, you will probably get an error when you try to PEEK above your memory limit.

In the strange coincidence department: I received a letter with a listing of CAL.BAS and a sample output that had all the year and month headings in place as well as the weekday headings, but no numbers for the days of the month. The spaces were there for them, but no numbers. I finally tracked it down to line 630, where the reader had typed in MS$ instead of M$. Within the same week, another such letter arrived with the identical symptoms, from another part of the country entirely. It was like deja vu all over again, the second reader had made the identical typo as the first, and in the same place too.

Which brings up an important point. If you have a problem and write us about it, please send an LLIST of the program the way you typed it in, along with something to show what the program is or is not doing right. We can usually get to the problem quickly that way and get back to you by return mail.

Thanks for all the great input.

Irv

Random Files

The 2nd of a series - Indexing

Terry R. Dettmann, Associate Editor. In this second installment on random files, Terry presents the technique of leaving files in place, but building an index with which to find things.

Last issue we did some basic operations with random access files. Nothing complicated, just getting and saving information. This time, we're going to add a little more sophistication to our knowledge and deal with a technique known as *Indexing*. As we get everything together, little by little, we're going to be putting together a random version of the Card.Bas program which will have an unlimited capability for record storage (well, limited to the size of your disk system).

On to indexing though. An index to a file of information is little more than a special way of remembering the order of information in the file. For example, let's say we store information in a file in alphabetic order. We would normally use the information in that order, but would have a real problem when we want to add a new item to the file. If we have 10 records, then add the 11th, we have to sort them in order to save them again. For 11 records, that isn't really bad even though sorting is a very time intensive operation.

But what if we have 10,000 records? A full sort on 10,000 records would involve a lot of time. We can save at least some of the time by using special indexing techniques. Probably the simplest, is to build an index of the file by sorting the information while referring to only part of the information. If we're sorting an address list by zip code for example, we only need to look at the zip code field, not the rest of the record. That alone will save us some time. However, in very large data bases, the major part of your time will be used in moving data to and from the disk. Let's look at a typical sort operation where we're actually going to move the records in the file to put them in order.

When we're moving records within the file, we have to read the record from the disk and write it back if its position has changed. Since we do this for two records at a time in swapping their positions, this eats a lot of system time. If we could leave the records where they are and not move them, we would only have to access the disk to read records, never to write them. This would save us even more time.

An index keeps track of the record numbers in the order we want to get them back, even if that order isn't the one we stored them in. For example, let's say we have the file shown below:

Record	Name
1	**Luke Skywalker**
2	**Han Solo**
3	**Leia Organa**
4	**Chewbacca**
5	**C3PO**
6	**R2D2**
7	**Obi Wan Kenobi**

Obviously, the records aren't stored in alphabetical order. The correct order, alphabetically, is:

5 4 2 3 1 7 6

This list of numbers is an index for that file. If we have the names in memory in an array (call it NM$), then the numbers could be stored in an index array (call it IDX) like this:

```
IDX(1) = 5
IDX(2) = 4
IDX(3) = 2
IDX(4) = 3
IDX(5) = 1
IDX(6) = 7
IDX(7) = 6
```

Now, if we wanted the first record of the file, we

could print NM$(1), but if we wanted the first record alphabetically, we would print NM$(IDX(1)). It works. If you want to see this approach in action, look at the Card.Bas program. The array P(I) is an index to the data that Card.Bas keeps. If you look carefully at the sorting procedure, you'll see that only the index numbers are changed. The actual data stays in the same place throughout.

In Card.Bas, the indexing technique makes sorting much faster than it would otherwise be because string movements cost more time than moving a number. With random files, we have exactly the same problem.

We have included a small demonstration program with this article to show sorting a random access file and recovering it using an index. Let's go through it so you can see how it's done.

There is very little initialization (lines 110-140). Variables NR (maximum number of records) and NX (next record into memory) are set up and the random access data file is created. We create the array IX (index to the data base) in memory and we'll use it here for the time being. Later, we'll come back and find out what we can do by putting the index on the disk instead of in memory.

Lines 180-300 are the program menu, asking you to choose from the 6 possible options for the program. If you are just starting out, you'll want to choose option 1 and create data to play with. The create data routine (lines 320-400) simply enters two lines of data for the file (up to 64 characters each) and stores it in the file at the next available location. Pressing ENTER when prompted for the first field will return you to the menu.

Option 2, indexing, sorts the data on disk by rearranging the index in memory. The sort has been written to be inefficient so you can see what happens when you try to move a lot of data back and forth between the disk and memory. We could restructure the sort, but you'll find it interesting to hear your disk labor through the sort. Don't do this too often, but do it to learn what the system will do.

Option 3 allows you to display, on your screen, the information stored in the file. You should try this before and after indexing to see how things change. When you're entering the data, it's nice to put the record number in each record as you create it so you can see what it will be. You could make the program print it, but we think you'll believe it more readily if you type it into the record itself.

Options 4 and 5 are for loading and saving the index. Notice that we are saving the index in a sequential file, not a random one. Just because we're dealing with random files, we don't need to limit ourselves to them. We could have stored the index in a random file. In fact, we will later when we build a more sophisticated indexing program, but recognize that even sequential files can be useful in random file programs.

Remember that indexing is a way of building a listing of which records to process and in what order. When indexing, we can use selection criteria to choose only those records we want, order the records by some criteria (alphabetical, etc.), or do anything we want to lay out the order we want to access the records in. This technique is used extensively in computer programming and is well worth the effort to learn and use. When we get to the random Card File program, indexing will take on major importance. ■

```
100 REM ************************************************
101 REM *                                              *
102 REM *           RANDOM ACCESS FILE DEMO 2          *
103 REM *            FILENAME: RANDEMO2.BAS            *
104 REM *                                              *
105 REM ************************************************
110 NR=100:NX = 1
120 OPEN"R",1,"RANDOM.DAT"
130 FIELD 1,64 AS X$,64 AS Y$
140 DIM IX(NR)
150 '
160 REM --- main menu
170 CLS:PRINT"RANDEMO2.BAS - Random Files Demonstration"
180 PRINT:PRINT
190 PRINT"     0. End"
200 PRINT"     1. Enter Data"
210 PRINT"     2. Index Data"
220 PRINT"     3. Display Data"
```

```
230 PRINT"      4. Load Index"
240 PRINT"      5. Save index"
250 PRINT:PRINT
260 INPUT"Option";OP
270 IF OP=0 THEN CLOSE:END
280 IF OP<0 OR OP>5 THEN PRINT"OOPS --- NO SUCH COMMAND":GOTO 260
290 ON OP GOSUB 320,420,540,700,630
300 GOTO 160
310 '
320 REM --- enter data
330 PRINT"Enter information in field 1 and 2"
340 PRINT"Leaving field 1 blank will terminate entry"
350 LINE INPUT"FIRST FIELD: ",F1$
360 IF F1$="" OR NX>NR THEN RETURN
370 LINE INPUT"SECOND FIELD: ",F2$
380 LSET X$=F1$:LSET Y$=F2$
390 PUT 1,NX:IX(NX)=NX:NX=NX+1
400 GOTO 320
410 '
420 REM --- index data
430 DF=NX-1
440 IF DF<1 THEN RETURN
450 DF = DF/2
460 SW=0
470 FOR I=1 TO NX-1-DF
480   GET 1,IX(I):TX$=X$:TY$=Y$
490   GET 1,IX(I+DF)
500   IF TX$>X$ THEN T=IX(I):IX(I)=IX(I+DF):IX(I+DF)=T:SW=1
510 NEXT I
520 IF SW=1 THEN 460 ELSE 440
530 '
540 REM --- display data
550 FOR I=1 TO NX-1
560   GET 1,IX(I)
570   PRINT"record #";I;TAB(15);X$
580   PRINT TAB(15);Y$
590 NEXT I
600 LINE INPUT"Press ENTER to continue ";ET$
610 RETURN
620 '
630 REM --- save index
640 OPEN"O",2,"RANDOM.IDX"
650 FOR I=1 TO NX-1
660   PRINT#2,IX(I)
670 NEXT I
680 CLOSE 2:RETURN
690 '
700 REM --- load index
710 OPEN"I",2,"RANDOM.IDX"
720 FOR NX=1 TO NR
730   IF EOF(2) THEN 760
740   INPUT#2,IX(NX)
750 NEXT NX
760 CLOSE 2:RETURN
```

NFL86.Bas

The CodeWorks Oracle projects winners

Staff Project. When you say "American" we all tend to think of apple pie, Motherhood and Girl Scout cookies. You may as well add NFL Football to the list. In this program, our NFL "Oracle" tries to look at team strengths and project not only the winner, but the point spread as well.

When the frost is on the pumpkin and there is a nip in the air you know it's time for NFL football again. It's a great time of the year, and the good old American pastime of following the football season enhances it considerably.

One of the things that makes football games so interesting is that they are not always easy to predict. It has been said that on any given day, any given team can beat any other team. It's still true.

NLF86.Bas is a program that attempts to project the winner and point spread for all the games in weeks four through 16 of the season. The name "Oracle" smacks of prediction, however, the program is really one that projects from current data rather than predict. It works with quantitative data rather than qualitative values. Miami playing in sub-zero weather in Buffalo, for example, is not considered by this program. Such judgments are left to you. The program simply presents you with the figures of what a team has done for the season and in the last three weeks. These figures are then compared to each other and also to the figures of the opponent, and provide you with information you can build your own interpretation on.

There are two programs involved; the projection program is called NFL86.Bas, the statistics management program is called NFLSTAT.Bas. We used two programs instead of just one because some computers may not have enough space for a combined version and the data arrays they create. Note that you do not necessarily have to use NFLSTAT.Bas. The statistics file can be created and maintained with a word processor. If you have MS-DOS, you could use EDLIN to create and maintain it. We tried it, and even CARD.Bas from Issue 2 (or the Sampler issue) can be used to make and maintain it. The statistics file has a simple format: Team number, week number, number of 1st downs, score, points allowed, (followed by Enter or Return.) Keep all the stats for one week in a contiguous chunk though, because the NFL86.Bas program expects to see them that way.

If none of the above ways of keeping the statistics is for you, then you can get them from our download. We will update the file and have it ready to download by Tuesday afternoon of each week after week three. The file name in the program (and on the download) is STAT.DAT.

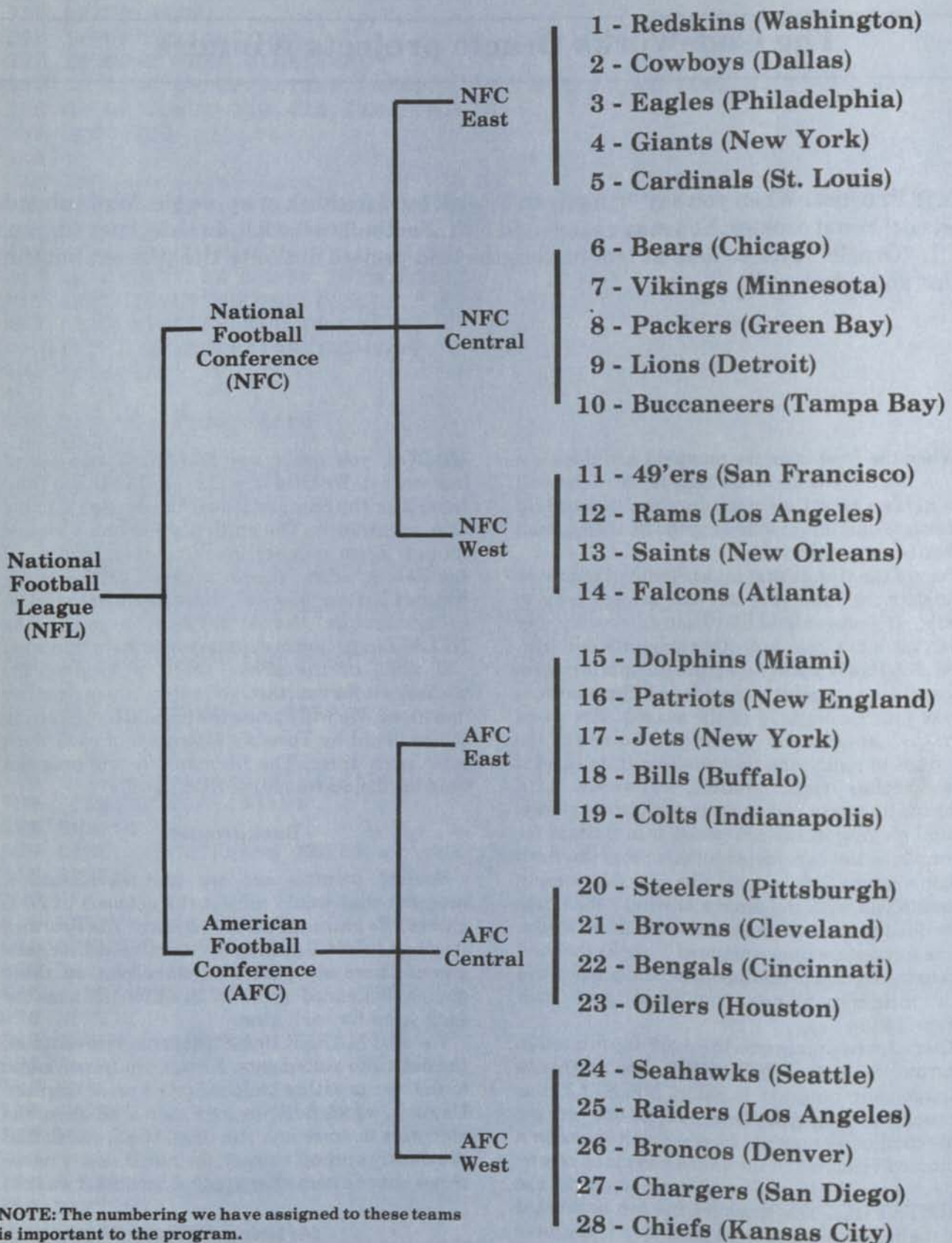
Background

Several months ago we first envisioned a program that would project the outcome of NFL games. We gathered all the copies of *The Sporting News* and started pulling out the statistics for each game. There are plenty of stats kept on these games. We ended up with 28 different items for each team for each game.

We used MAKER.Bas (from Issue 1) to enter all the data into statements. Next, Cam Brown came to the rescue with a multiple correlation program. Using it, we tried to correlate each (and all) of the statistics to score and win/loss. Hope, which had previously sprung eternal, dwindled slowly as we threw out one item after another because it was not

(Article continues on page 12)

National Football League (NFL)



NOTE: The numbering we have assigned to these teams is important to the program.

Figure 1

The CodeWorks NFL ORACLE PROJECTION FOR WEEK 8

Key to column headings
1- Teams plus Oracle's Winner projection
2- Oracle's overall rating number (not a score)
3- Number of games won this far in the season
4- Last 3 weeks average 1st downs
5- Last 3 weeks average points scored
6- Last 3 weeks average points allowed
7- Season average 1st downs
8- Season average points scored
9- Season average points allowed
10- Actual score (you fill in after the games)
11- Actual point spread (fill in this too.)

1	2	3	4	5	6	7	8	9	10	11
FALCONS	220	1	21	24	30	19	21	30	10	
COWBOYS by 15	294	5	22	23	19	22	24	16	24	14
GIANTS by 8	269	4	24	25	22	21	22	16	21	
SAINTS	233	3	15	20	25	16	21	26	13	8
BILLS	173	1	13	13	24	16	12	24	17	
EAGLES by 22	278	3	21	22	14	16	14	14	21	4
PATRIOTS by 1	245	4	18	18	13	17	17	18	32	
BUCS	243	0	19	28	33	19	21	31	14	18
DOLPHINS by 15	253	5	22	24	27	23	26	20	21	
LIONS	181	4	13	12	29	13	18	22	31	X
STEELERS	233	3	14	18	20	18	21	16	21	
BENGALS by 5	257	2	21	27	34	22	30	34	26	5
VIKINGS	243	4	23	16	16	20	22	20	9	
BEARS by 19	335	7	22	25	12	21	30	15	27	18
BRONCOS by 15	273	5	21	19	13	21	25	20	30	
CHIEFS	202	3	14	10	22	15	20	21	10	20
NINERS	258	3	20	23	22	20	25	20	23	
RAMS by 5	282	7	13	20	12	15	21	12	14	X
PACKERS	262	3	21	23	16	19	20	23	10	
COLTS by 1	264	2	18	22	17	17	16	22	37	27
REDSKINS	247	3	18	18	10	19	14	21	14	
BROWNS by 8	282	4	19	21	15	19	19	15	7	X
OILERS by 7	221	2	19	23	26	15	17	22	20	
CARDS	190	3	17	9	26	19	22	26	10	10
SEAHAWKS	251	4	18	22	20	20	24	26	14	
JETS by 8	290	5	20	21	15	20	22	14	17	3
CHARGERS	251	3	21	23	22	22	24	26	21	
RAIDERS by 7	284	5	17	21	14	18	22	19	34	13

This is a sample of week 8 of the 85-86 season.

An "X" in the right-hand column indicates a game picked incorrectly.

(NFL86 from page 9)

even slightly significant. It was about that time that Cam offered the suggestion that we simply count all the blue and all the green cars in the parking lot at the game to find the winner. We all agreed that it would be as good as trying to figure in the third down conversion ratio. Football, we found out shortly, was a very complex and difficult to predict game.

After sifting through all the data several times, what emerged was this: The team with the highest score wins. The magnitude of the winning number is not important. You can win with a score of three and lose with a score of 48. It all depends on what the other team had. That was so obvious that it hurt. We did find, though, that the number of first downs a team made was tied to some degree with their final score. That, points made and the number of points the defense allowed the other team to get were the most significant of all the factors. In a way it really helps to have so few items to consider. It makes the statistics easier for anyone to gather (local newspaper), and cuts down the size of the program and the data arrays it must carry.

The next problem was how to project a winner and a point spread from this data. We looked at some other programs and found that there were too many times when they claimed the game too close to call. We didn't want any of that, the rationale being that if we were already out on a limb, why not go all the way and project a winner, if even by one point.

Empiricism still reigns supreme! We tried averages, moving averages, means and medians, to no avail. Basing next week's performance on last week's outcome is no good. The team may have been off, or playing a much superior team last week. What finally sifted down was looking at the last three week's average, then flavoring that to a lesser degree with the overall season average. That way, you can see a team generally moving up, down or holding steady. When you look at each team this way, you can compare their numbers with their opponent for next week and generally see who has the power. It tells who has a larger edge and how much of an edge. From this, we can draw conclusions as to who will win and by how many points.

Sounds good, but how can you explain how the Oilers (1985- 86 season, week 7) with an overall rating for that week of 172, beat the Bengals (rating 272) by a score of 44 to 27? You can't. Try as we may, we couldn't find anything in the numbers that served as a leading indicator of a true upset. That's why football is still such a great game. There are obviously many subjective factors that go into predicting a winner. What they are and how to evaluate them is what makes Jimmy the Greek run.

General Notes

Because of the way the program works, you cannot make any projections until the statistics for weeks one through three are in. This means that we can project week four and after. This also allows us to leave out the schedule for weeks one through three in the data statements at the end of NFL86.Bas.

To simplify the program, we have numbered all the NFL teams (see sidebar). You need to know this number when updating the statistics file; everywhere else in the program that number will identify the actual team by name.

The program will not allow projections if all the data for all teams is not present for all the previous weeks. Games played on Thursdays always count for the *following* weekend. Games played on Monday always count for the *previous* weekend. The program output will show each game as a pair of teams. The first one listed is always the visiting team, the second is the home team.

NFL86.Bas Program Notes

The program starts by setting up several arrays in line 170. Array A(x,x) contains the statistics read in from the disk file STAT.DAT. It only needs 420 items because the stats for week 16 are not needed. The B(x,x) array will hold the last three week averages for each of the 28 teams. This array is six deep because we will calculate the won/lost for each team and store it in the sixth position. The F(x,x) array will hold the season averages for each of the 28 teams. The T$(x) array holds the team names and the P(x) array holds the season schedule.

The team names are contained in data statements in lines 190 through 220. It is important they remain in the order shown since their position number is the same as the team number in the statistics file. These names are read into the T$ array in lines 240 through 270. The season schedule is then read into the P array in lines 290 through 320.

The CodeWorks heading is printed next. Line 400 sets the printer (yes/no) flag to no printed output. Line 410 asks which week number we want

to project, and the lines immediately following check to see that that week number is within the range we want. In line 440, if we want printed output, the printer flag is set to 1.

In line 450 we make W1 through W4, which are the projection week less 1 through 4. This makes these week numbers easier to handle later in the program. Line 470 initializes WN to equal the number of items we should read in from the STAT.DAT file for any given projection week number. It is used in the following code from lines 490 through 600. In these lines we read from 1 to WN, and if the loop counter I does not equal WN then line 590 will tell us we do not have enough data for the given projection week. The program then ends so that you can correct the stat file.

The loop between lines 620 and 770 finds the season average for each team and puts that average into the F(x,x) array. The variable X is always the team number. The inner loop from 650 through 710 scans the A(x,x) array looking for the proper information to extract for the given team. In line 660, if the team number in the stat array, A(x,x) does not equal the team number we want, we go to NEXT I and keep looking. If we do find the correct team number, line 670 does a check to see if the week number in the stat array is equal or greater than the projection week. This allows us to go back and rerun projections on previous weeks if we want to. The little J loop between 680 and 700 accumulates the stat information for the given team. It only needs to be concerned with positions three to five, since the team number in position one and the week number in position two are immaterial at this point. After all the data is accumulated, line 720 attaches the team number in position one. Now another little J loop from lines 730 through 750 goes through the N(J) array and puts the average number for the season into the F(x,x) array. Since the N(J) array is accumulating (see line 690) we need to clear it out before we do the next team or it will contain left over data from the previous team. This is done in line 760.

Finding each team average for the last three weeks (see code lines 790 through 950) is almost identical to the previous section of code. This time though, we fill the B(x,x) array. Unlike the previous section, in this one we only look at the last three weeks (line 850). One other difference can be noted here: In line 840 we are looking at the teams for the entire season and checking to see if their score (in A(I,4)) is larger than the number of points they allowed (in A(I,5)) and if it is, then we add one to the B(X,6) position. B(X,6) tells how many games so far in the entire season team X has won.

Now that we have all the needed information tucked away in the proper arrays, we can start playing the teams against one another.

The main loop in this section is from line 1200 through 1400. Just prior to the loop (in line 1190) we set the point from which to read in the schedule (P(x)) array. Variable SI becomes that starting point and is calculated in line 1190. The S loop then reads the schedule array, two teams at a time and makes the visiting team variable X and the home team variable X1 in line 1210. The team names then become X$ and X1$ in line 1220. This way, we can evaluate the first pair of teams in the schedule, print their results and go on to the next pairs.

Lines 1230 and 1240 sum rating points for each team. These lines were derived empirically and go like this: (take a deep breath) Add the team's season average first downs to their last three week first downs. Then add two times their average season score plus four times their last three week score. Then add 40 less their season average points allowed to 40 less their last three week points allowed. In addition, the home team (in line 1240) gets an extra 20 added, which will be approximately a 2 point home team advantage in the final outcome. Taking 40 less points allowed gets all the numbers going in the right direction, since more points allowed are not as good as less points allowed. Variables S0 and T0 then contain numbers in the 100 to 400 range, which are used to determine not only the relative strength of the team, but how many points to assign to the point spread.

Line 1260 says that if the two rating numbers are equal, to assign the home team as the winner by one point. This point is in addition to the two already assigned to the home team, and represents what the odds-makers generally give for home team advantage. This also keeps the program from making "too close to call" judgments. After looking at the other numbers for each team, you can make that call for yourself if you like. Lines 1270 and 1280 add the string value of the point spread to the appropriate team. In those lines, the large rating numbers we calculated earlier are now divided by 5 and one is added. The reason we need to add one is so that a team will not be declared winner by zero points.

Based on the 1985-86 season, this program was rather generous in its point spread projection. It projected over the actual point spread 42% of the time. You can make the point spread projection more pessimistic by dividing by a larger number than five in lines 1270 and 1280. For example, dividing by 10 will cut the point spread in half.

If you opt for printer output it will look like Figure 1. If you do not opt for printer, the screen will show you headings and four pairs of teams, and a prompt to press Enter for more. If your video screen is too small for four games, then change the 4 in line 1370 to 3 or 2. As it now stands, the screen will not display as much information as the printer output will. The LPRINT CHR$(12) in line 1410 is a printer "top of form" command. Change it to suit your printer or leave it out altogether.

The data statements which follow line 1420 are the 1986-87 NFL schedule for weeks four through 16. Each pair of data lines represent one week, with the number of the visiting team first and the home team next. If you are entering this program through your keyboard make sure these numbers are all there and are correct. If, for example, you had a number one followed by number one, you would have the Redskins playing the Redskins and the home team Redskins would win!

It takes the program a little while to accumulate all the averages. We compiled this program with the Microsoft Quickbasic compiler and found a vast improvement in processing time.

Results

We feel just a little insecure about presenting this program since there is no easy way to give you a sample run. Also, there is no guarantee whatever that it will perform for next season as it did for the last two seasons. We may all be very surprised or very disappointed. Who knows? But that's the fun of NFL football. Here is what it *should* do: It should pick the winner for the 13 weeks about 66% of the time. It should pick the point spread, plus or minus three points, about 30% of the times when it picked the correct winner. Running it against last season, it picked the Bears to win every game they played. It missed the Bear's upset by the Dolphins in week 13. It picked all 14 games correctly in week six last year. The worst it did was to pick only seven of 14 in three different weeks. Again, that's history. What it will do this year is a big question mark. We hope you have as much fun with it as we did putting it together - *but don't bet the farm on it.*

Changes you can make

The basic structure of the program is there. If you do not agree with our way of determining the winner and point spread, you can make quite a few changes easily. For example, you could gather different statistics than we did. By changing the dimensions of the arrays, you can use more statistics than we did. The entire rating scheme in lines 1230 through 1280 can be manipulated any way you want.

Although not as easy to do, you can also make provision for user input of such items for each game as, for example, weather conditions, coaching changes, injuries or trades. How you assign values to such factors would be an interesting study in itself.

NFLSTAT.Bas Program Notes

NFLSTAT.Bas is a program you can use to create and maintain the NFL statistics used in the NFL86.Bas program. It is quite similar (from lines 100 through 400) to the NFL86.Bas program and the discussion of that program applies here too. Lines 130 and 140 are remark lines that tell you how to establish the file initially. Once the file is established, ***do not*** do this again or it will wipe out what you already have in the file.

The program is a simple database program without too many bells or whistles. Variable L1 is established in line 400 as the number of entries that were read in from the data file. An entry in this case consists of: Team number, week number, number of 1st downs, score and points allowed. Each entry represents one team's statistics for one week. Since 28 teams play each week, we can check to see if the stats are complete. This is done in line 480, where we take the entries MOD 28. If the result is zero, we have the right number of entries, if it is not zero then there must be extra or missing data in the file. For those of you who do not have MOD in your BASIC, change line 480 as follows:

```
480 X1=INT(L1-28*INT(L1/28)):IF X1<>0 THEN etc.,etc..
```

Also change line 1040:

```
1040 X1=INT(I-14*INT(I/14)):IF X1=0 THEN etc., etc..
```

Update the file

It is best to collect all the stats (including the Thursday and Monday games) prior to updating the file. It is best done in one session. With this program, the team number and week number will be inserted into the file automatically. You need only enter the 1st downs, points scored and the points allowed. This is done in lines 640 through 730.

Edit an item

Given a team number and week number, the section of code from lines 780 through 950 will search for that item and show it to you on the screen. You then need to re-enter all the information for that entry, including the team and week numbers. If the program cannot find what you asked for, you will be informed and given the opportunity to go back to the main menu in line 860.

View the file

This option lets you scan the entire file, 14 teams at a time, from beginning to end. With 14 items and the heading, the display should just fit a 16-line video screen. The width is also adjusted so as to fit a 64 column screen. The 14 lines are dealt out by the For...Next loop starting at line 1000. When I reaches 14, line 1040 comes into play and stops the display, giving you the opportunity to read it and then press Enter for more. When this happens, the screen is cleared, a new heading is printed and 14 more lines are dealt out. If the number of lines is not 14, then the ELSE in line 1040 will cause a jump around the heading in line 1050 and continue with NEXT I. Following the "no bells" concept, once you start to read the file you must read all the way to the end.

Save file and END

You must use this option to save the file back to diskette or your updates will be lost. As we noted earlier in this article, there are several other ways to make and maintain the stat file. If none of the others appeal to you, perhaps this one will do the job. ■

The Projection Program

```
100 REM ** NFL86.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 NFLSTAT.BAS  -  See CodeWorks Issue 7 for details *
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$(INT(((S5-T5)+.5)/5)+1)
1280   IF S5<T5 THEN X1$=X1$+" by"+STR$(INT(((T5-S5)+.5)/5)+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 86-87 SCHEDULE FOR WEEKS 4 THROUGH 16 FOLLOWS *
1440 DATA 14,10,6,22,9,21,8,7,28,18,12,3,13,4
1450 DATA 16,26,17,19,20,23,27,25,11,15,24,1,2,5
1460 DATA 18,17,22,8,21,20,2,26,23,9,19,11,25,28
1470 DATA 15,16,7,6,4,5,3,14,10,12,1,13,27,24
1480 DATA 18,15,6,23,26,27,9,8,28,21,12,14,7,11
1490 DATA 13,19,17,16,3,4,5,10,24,25,1,2,20,22
1500 DATA 6,7,2,3,9,12,8,21,23,22,19,18,25,15
1510 DATA 16,20,4,24,5,1,27,28,11,14,10,13,26,17
1520 DATA 14,12,22,20,21,7,9,6,25,23,15,19,16,18
1530 DATA 13,17,5,2,27,3,11,8,24,26,10,28,1,4
1540 DATA 14,16,18,10,22,9,21,19,2,4,26,25,8,20
1550 DATA 23,15,28,27,7,1,17,24,3,5,11,13,12,6
1560 DATA 6,10,22,23,25,2,12,13,7,9,16,19,4,3
1570 DATA 17,14,20,18,5,11,27,26,24,28,1,8,15,21
1580 DATA 6,14,21,25,2,27,9,3,23,20,19,17,28,26
1590 DATA 16,12,15,18,4,7,13,5,24,22,10,8,11,1
1600 DATA 25,27,14,11,18,16,2,1,26,4,9,10,8,6
1610 DATA 19,23,28,5,7,22,13,12,3,24,20,21,17,15
1620 DATA 8,9,24,2,14,15,18,28,22,26,23,21,12,17
1630 DATA 16,13,3,25,20,6,27,19,10,7,1,5,4,11
1640 DATA 22,16,21,18,2,12,26,28,9,20,23,27,19,14
1650 DATA 15,13,7,8,4,1,17,11,5,3,10,6,25,24
1660 DATA 20,17,1,26,18,19,21,22,8,10,28,25,15,12
1670 DATA 7,23,13,14,3,2,5,4,11,16,24,27,6,9
1680 DATA 12,11,26,24,8,4,14,9,18,23,6,2,19,25
1690 DATA 28,20,13,7,17,22,27,21,10,5,1,3,16,15
```

The Statistics Maintenance Program

```
100 REM * NFLSTAT.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 NFL.BAS from Issue 7, CodeWorks
130 REM * If no file exists then in command mode,type OPEN"O",1,"STAT.
    DAT"
140 REM * and press ENTER, then type CLOSE and press ENTER. This
    creates an
150 REM * empty file called STAT.DAT. You can then run this program.
```

```
160 PRINT"Loading STAT.DAT file from diskette.."
170 ' CLEAR 10000: ' Use only if your machine needs to clear string
    space.
180 DIM A(420,5),T$(28)
190 '
200 DATA REDSKINS,COWBOYS,EAGLES,GIANTS,CARDS,BEARS,VIKINGS
210 DATA PACKERS,LIONS,BUCS,NINERS,RAMS,SAINTS,FALCONS
220 DATA DOLPHINS,PATRIOTS,JETS,BILLS,COLTS,STEELERS,BROWNS
230 DATA BENGALS,OILERS,SEAHAWKS,RAIDERS,BRONCOS,CHARGERS,CHIEFS
240 '
250 REM * READ IN THE TEAM NAMES *
260 FOR I=1 TO 28
270   READ T$(I)
280 NEXT I
290 '
300 REM ** READ IN THE EXISTING STAT FILE **
310 WN=420
320 OPEN "I",1,"STAT.DAT"
330 FOR I=1 TO WN
340   IF EOF(1) THEN 390
350   FOR J=1 TO 5
360     INPUT #1,A(I,J)
370   NEXT J
380 NEXT I
390 CLOSE 1
400 L1=I-1
410 '
420 CLS: ' Use your own clear screen command here.
430 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
440 PRINT"          N F L   W E E K L Y   S T A T I S T I C S"
450 PRINT"        Maintains statistics for 1986-87 NFL Football"
460 PRINT STRING$(60,"-")
470 PRINT
480 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
490 PRINT
500 PRINT TAB(10)"1 - Update the file"
510 PRINT TAB(10)"2 - Edit an item in the file"
520 PRINT TAB(10)"3 - View the entire file"
530 PRINT TAB(10)"4 - Save the updated file and END"
540 PRINT
550 INPUT" Your choice";X
560 IF X<1 OR X>4 THEN 550
570 ON X GOTO 610,780,980,1110
580 END
590 '
600 REM * UPDATE THE FILE ROUTINE **
610 CLS
620 INPUT"UPDATE STATISTICS FOR WHICH WEEK NUMBER";W
630 IF W=<L1/28 THEN PRINT"The file appears to be updated through
    that week."
640 J=L1+1
```

```
650 FOR X=1 TO 28
660   PRINT"For the ";T$(X);" for week ";W
670   INPUT"How many first downs ----------";A(J,3)
680   INPUT"How many points did they score --";A(J,4)
690   INPUT"and they allowed how many points-";A(J,5)
700   A(J,1)=X:A(J,2)=W
710   PRINT
720   J=J+1:L1=L1+1
730 NEXT X
740 PRINT"Press Enter for menu";:INPUT X:GOTO 420
750 END
760 '
770 REM ** EDIT AN ITEM IN THE FILE ROUTINE **
780 CLS
790 PRINT "EDIT DATA - You supply the team number and week number."
800 PRINT
810 INPUT"What team number are you looking for ";X
820 INPUT"What week number are you looking for ";W
830 FOR I=1 TO L1
840   IF A(I,1)=X AND A(I,2)=W THEN 870
850 NEXT I
860 PRINT"That item is not in the file":GOTO 940
870 PRINT T$(A(I,1));A(I,1);"Week->";A(I,2);"1st Dns->";A(I,3);"Score-
    >";A(I,4);"Pts Allowed->";A(I,5)
880 PRINT
890 INPUT"Enter correct team number ---";A(I,1)
900 INPUT"Enter correct week number ---";A(I,2)
910 INPUT"Enter correct 1st downs -----";A(I,3)
920 INPUT"Enter correct score ---------";A(I,4)
930 INPUT"Enter correct points allowed ";A(I,5)
940 INPUT"Press Enter for menu";X:GOTO 420
950 END
960 '
970 REM * VIEW THE FILE ROUTINE **
980 CLS
990 PRINT"TEAM #","WEEK","1ST DOWNS","SCORE";"   ";"PTS ALLOWED"
1000 FOR I=1 TO L1
1010   FOR J=1 TO 5
1020     PRINT A(I,J),
1030   NEXT J
1040 IF I MOD 14=0 THEN PRINT"Press Enter for more..";:INPUT X:CLS:
     ELSE 1060
1050 PRINT"TEAM #","WEEK","1ST DOWNS","SCORE";"   ";"PTS ALLOWED"
1060 NEXT I
1070 GOTO 420
1080 END
1090 '
1100 REM * SAVE THE FILE AND END ROUTINE **
1110 OPEN "O",1,"STAT.DAT"
1120 FOR I=1 TO L1
1130   FOR J=1 TO 5
1140     PRINT #1,A(I,J)
1150   NEXT J
1160 NEXT I
```

```
1170 CLOSE 1
1180 PRINT"THE FILE STAT.DAT IS NOW SAVED"
1190 PRINT"END OF PROGRAM."
1200 END
```

Programming Notes

Some of the "cloned" machines do not have standard BASIC. It looks like regular BASIC and acts like it most of the time, but there are subtle differences. One we recently became aware of was a Japanese BASIC running under MS-DOS. It required the number sign (#) before the buffer number in opening sequential files for either input or output. Standard Microsoft BASIC usually allows the number without the number sign in this case, but requires it for PRINT #1 or INPUT #1 statements.

We all know that the statement PRINT FRE(0) will tell us how many available bytes are left in memory. But did you know that the same command will perform the "memory management" function? Before it tells you how much memory you have left, it goes through and clears and reorganizes the string space and all superfluous garbage in memory. This clearing action would take place by itself when memory gets near to full. You can prevent long delays in this clearing by using FRE(0) inside your program to force it. During memory management, you are effectively locked out. No keys on the keyboard will respond. By using FRE(0) you can create much shorter (but more frequent) periods of such lockout. Memory tends to fill up rapidly when you start reassigning strings (as in switching them in a sort).

MS-DOS BASIC's have a SAVE"filename",P option which saves the file (or program) in an encoded binary format. After this save, the only operations that can be performed on the file are RUN, LOAD and CHAIN. Be careful with this option! It can be a real trap. If you have not saved another version of the same program normally, you cannot go back to list or edit it.

Another trap to avoid is in making a menu program that calls other programs (as in DRILL.Bas in this issue.) Each of the called programs will return you to the menu program. If you are working on one of the called programs and have not yet saved it, it is very easy to inadvertently answer "N" or "No" to the "Continue" question, and wipe out all the work you have done on the program. During development of such programs, it is best to change the "RUN Menu" line in the called programs to something that will not load and run the main program, then change it back when everything is checked out.

We have never seen this documented but ran across it by accident one day. On MS-DOS machines, if you hold down the ALT key and type a number on the numeric key pad, when you release the ALT key the screen will display the ASCII symbol for the number you typed. It won't work with the regular number keys, only the keypad numbers. It's very handy to quickly check out what some of those high ASCII numbers represent.

Creating an empty file on diskette from command mode is very easy. You should be in BASIC with the ready prompt. Then type OPEN"O",1,"filename":CLOSE and press RETURN. This creates an empty file called "filename" on your diskette. If you want to initialize a file with a sentinel in it (as in CARD.BAS), you can do it like this: From the BASIC ready prompt, type OPEN"O",1,"CARD.DAT" then press RETURN. Then type PRINT#1,"ZZZ" and press RETURN. Then type CLOSE and press RETURN. That will initialize the file and put the ZZZ sentinel in it. Be careful though, if a file already exists with that name, it will be wiped out when you do this!

Puzzler #7

and answers for #5 and #6

Puzzler 5 in the last issue was a "black box" problem. The length of the line was given as 40 characters. The problem was to input A$ and include in it the control character $UL, which was supposed to create a leader (underline) line across the width of the line. The control was to be inserted wherever in the line a leader was desired. It was to work before words in the line, after words in the line or between words.

After _____
_____ before
between _____ words

After receiving dozens of replies to earlier Puzzlers, we were rather surprised that there were only four entries for this one. All four are listed here for your perusal and comment. We must admit, with some chagrin, that our own way of doing it lacked the technique shown by any of these four.

Are puzzle solvers good puzzle makers? Our mail on puzzlers runs from those who say they are too easy to a few who say that the puzzles are not well formulated. We welcome your ideas on the subject and if you know any good programming or logic puzzles, please share them with the rest of us.

Puzzler 6, the "throw in" puzzler to get our puzzle numbers and issue numbers back into sync, had a simple answer. It was to put a space between the $ and the UL for transmission to the typesetter, then after it was transmitted, remove the space. We will go along with the rest of you who said that it didn't qualify as a puzzler. It didn't, but it served its purpose. Now on to puzzler 7.

Puzzler 7

There are many instances in computing when spaces need to be removed from either the beginning or end of a string, sometimes from both. This would be true when converting strings from fixed length fields to another format, or when right- and left-justifying text strings for printing. This is the subject of our next puzzler.

Given a string of any length, including a null string, what is the most efficient code that will remove the spaces on either end (or both ends) of the string?

Read "efficient" code as "elegant" or "clever" or "shortest" code. We haven't done this one yet either, so we will be working on it right along with you. We have included the case of the null string because that may be a possibility in the real world and would need to be dealt with. Our issues normally close one month before the cover date, i.e., Nov/Dec 86 issue closes on 1 October 86, so entries we receive before 1 October will be considered in the Nov/Dec 86 issue. ■

```
100 REM * PUZ5.BAS MARK GARDNER N. HOLLYWOOD,CA
110 LL=40
120 PRINT"For a leader line, insert $UL at the place"
130 PRINT"where you want it to appear in the line."
140 PRINT
150 LINE INPUT"Input A$ ";A$
160 L=INSTR(A$,"$UL")
170 IF L=0 THEN GOTO 150
180 B4$=""
190 IF L>1 THEN B4$=LEFT$(A$,L-1)
200 AF$=""
210 IF L<(LEN(A$)-2) THEN AF$=RIGHT$(A$,LEN(A$)-L-2)
220 A$=B4$+STRING$(LL-LEN(B4$)-LEN(AF$),"_")+AF$
230 PRINT A$
```

```
100 REM * PUZ5.BAS JOHN ANDERSON ARLINGTON MA
110 LL=40
120 PRINT"For a leader line, insert $UL at the place"
130 PRINT"where you want it to appear in the line."
140 PRINT
150 LINE INPUT"Input A$ ";A$
160 LSTR=LEN(A$)
170 PMAC=INSTR(A$,"$UL")
180 IF PMAC THEN O$=LEFT$(A$,PMAC-1)+STRING$(LL+3-LSTR,"_")+RIGHT$(A$,
    LSTR-PMAC-2) ELSE O$=A$
190 PRINT O$


100 REM * PUZ5.BAS MERTON L. DAVIS CAMDEN, SC
110 LL=40
120 PRINT"For a leader line, insert $UL at the place"
130 PRINT"where you want it to appear in the line."
140 PRINT
150 LINE INPUT"Input A$ ";A$
160 L=LEN(A$)
170 I=INSTR(A$,"$UL"):UL$=STRING$(LL-L+3,95)
180 PRINT LEFT$(A$,I-1);UL$;RIGHT$(A$,LL-I-LEN(UL$)+1)


100 REM * PUZ5.BAS KEN BUSCH SAN DIEGO, CA
110 LL=40
120 PRINT"For a leader line, insert $UL at the place"
130 PRINT"where you want it to appear in the line."
140 PRINT
150 LINE INPUT"Input A$ ";A$
160 SL=LEN(A$):AL=SL-3:UL$=STRING$(LL-AL,"_")
170 FOR Q=1 TO SL
180   IF MID$(A$,Q,3)="$UL" THEN GOSUB 220
190 NEXT Q
200 PRINT B$+UL$+C$
210 PRINT:GOTO 150
220 B$=LEFT$(A$,Q-1)
230 C$=RIGHT$(A$,AL-Q+1)
240 Q=SL
250 RETURN

100 REM * PUZ5.BAS * Puzzler for Issue 6 *** CodeWorks *
110 LL=40
120 PRINT"For a leader line, insert $UL at the place"
130 PRINT"where you want it to appear in the line."
140 PRINT
150 LINE INPUT"Input A$ ";A$
160 P=INSTR(1,A$,"$UL"):IF P<>0 THEN J=LEN(A$)-3
170 IF J>LL THEN J=LL
180 FOR I=1 TO LEN(A$)
190   T$=MID$(A$,I,1)
200   IF MID$(A$,I,3)="$UL" THEN I=I+3:T$=MID$(A$,I,1):PRINT
      STRING$(LL-J,"_");
210   PRINT T$;
220 NEXT I
```

Beginning Basic

Arrays? Why get complicated; don't I have a computer full of variable names to use?

That's what you are likely to hear from people who have just started learning BASIC. They are right. You do have a computer full of variables to use, and if the program lends itself to that level of usage, so much the better. But with an array of even one dimension you can start to treat groups of variables representing similar functions on a much higher and more efficient level.

For example, you may have five values to store in memory for some program. You could assign them names like A, B, C, D, and E. Now you need to write code to handle each of them individually, both when you input the data and when you compute and output it. Granted, with only five such variables, it would not be a monumental task, but what happens when you need 100 variables? Instead of creating five different variables with five different names, we can just as easily create one subscripted variable. Let's call it A(x). Now x can be any number we like (or have space in memory for). The x in this case is called the subscript. Now we can assign our five values to A(1) through A(5). Even though the A remains the same, the subscript number makes a different variable of each of the subscripts. The flag-waving advantage of this method is that now we can use a simple loop to input, process or output the data in the array. Furthermore, we can add data to the array (make it hold more items) simply by increasing the subscript number. This removes the necessity of rummaging around in the alphabet, looking for a new and unused letter.

Note that when A(x) is assigned, it is a totally *different* variable than A (without a subscript), or A$ (string). And yes, before you ask, A$(x) would create an array for string variables. Now what about the number of the subscript? Where does it start? When you assign the A(x) array, BASIC sets aside spaces in the array starting at subscript number 0 whether you use it or not. Some computers have a BASIC statement called "OPTION BASE". It sets the minimum array subscript to either a zero or one. The default is zero, and if your computer does not have OPTION BASE, your subscripts will start with number zero. What about the highest number?

BASIC automatically assigns 11 spaces to a subscripted variable (zero through 10). If you try to use A(11) you will be greeted with a "Subscript out of Range" error. To use an index of 11 or more, you need to *dimension* the array with the DIM statement (preferably early in the program.) A given array may be dimensioned only once in the program. There are exceptions, but they are beyond the scope of this article. Telling BASIC to DIM A(100), for example, would give you 101 spaces in the A array (zero through 100).

You can look at the subscript as though it were an *index* number. It is. When you dimensioned the A array, BASIC set aside a whole section of spaces and called them all the A array. You can visualize them as a series of cubbyholes, each with a number on them. The index (subscript number) is simply the number of the cubbyhole. You can now begin to fill the cubbyholes with useful information. Assuming that you set the dimension of the A array to A(100), you could then use the following code to fill the array:

```
10 DIM A(100)
20 FOR I=1 TO 100
30    INPUT A(I)
40 NEXT I
```

This is certainly preferable to assigning 100 different variables and asking for each one separately. You have just reduced 100 input statements to one, plus a line for the DIM statement and two for the loop. That's the power of an array.

Another property of the array scheme is that the index number can be computed. Yes, you can use arithmetic, as in A(I+2), which would be array index two down the line from where I currently is. In sorting we generally want to look at the current item and compare it to the next item. In that case we could set L equal to I+1 and then compare A to A(L). That would compare the informatic *A(I)* to the information *in A(L).* Don't t indicies or subscripts lead you away from that they are simply cubbyhole identifica *not* the information itself.

In computer jargon we talk about "stuffi array. This simply means that we are putting so numbers (or strings) into assigned array spaces.

So far, we have talked only about single dimension arrays. Like a line, they have only one dimension: length. In the next issue we will introduce arrays that have two and three dimensions. ■

Drill.Bas

A collection of math and spelling exercises

Bob Henkel, Tacoma, Washington. This type of program has been done before, but we especially liked the way Bob made the problems get harder and easier as a function of your correct answers. These programs have been used extensively by Bob in actual classroom situations.

In January, 1981, when I purchased my first computer, one of the first programs I wrote was the one for multiplication practice. It seemed a natural thing for a math teacher to do. Since then it has been updated several times. Items such as a timing loop, in conjunction with INKEY$, scoring of problems and automatic increase and decrease of difficulty of the problems depending on the score were added.

In the fall of 1985, with a move to a junior high school, addition, subtraction and division were added to the set.

In an effort to make further use of the computer and encourage student progress, I wrote the two programs on mental arithmetic. One is strictly addition of 20 single digit numbers. The other gives random addition and subtraction of single digit numbers with two out of three chances for addition (so the answer usually gets larger.)

Since I was also teaching a spelling class, I tried to devise a program for spelling practice. This turned out to be quite a chore. The first attempt was to use a phrase as a clue. The job of thinking up clues and typing them into the computer was great and the results were very poor. The clue seldom elicited the correct word. Attempt number two was to have the entire word flash on the screen at once. This was undesirable. If the word appeared too long a time, the spelling was too easy. If the word appeared too short a time, the word was unrecognized. The third attempt proved to be successful. It was to have the word scroll slowly across the screen two letters at a time.

The menu part of the program was my first attempt to load one BASIC program from another. I have discovered that trying to write programs is one of the best ways to learn about programming. Further additions to the program are only in the thought process. Fractions are a good possibility and the graphics necessary to complete the task would be a valuable learning experience. These programs were originally developed on a Tandy Model 4. Any suggestions for additions and improvements would be appreciated. *Send them to: Bob Henkel, 10613 25th Avenue East, Tacoma, Washington 98445*

Program Overview

The first four menu items are arithmetic drill. Each program consists of 20 problems. The student is to select the difficulty of the problems to start. If no more than one problem is missed the problems get harder, while missing five or more causes the problems to get easier. There is a limited time allowed for each answer.

If a problem is answered wrong or not answered quickly enough, the correct answer appears on the screen and a delay loop allows the student to concentrate on the correct answer for a short time. At the end of each set of 20 problems a score is given for the last 20 as well as a score for all problems already done. Also, at the end of each 20 problems, the student is given the option of continuing or changing to a different type of problem.

The menu option 5 is spelling practice. Each word scrolls across the screen, two letters at a time. Then the word is to be typed from the keyboard. Three attempts to spell each word are allowed. Scoring is five points for correct spelling on the first try, three points for the second try and one point for getting it correctly on the third try. If the word is not spelled correctly on the first two tries, the word will scroll across the screen again. Incorrect spelling of the word on the third try will display the word and then go on to the next word.

There is a menu option for entering or correcting spelling word files. It is menu item 8, and the program is called UPDATE.BAS. The files in

which the spelling lists are stored are sequential and are all named UNITab, where 'a' refers to the grade level and 'b' refers to the lesson number.

Menu optons 6 and 7 are practice in mental addition, or mental addition/subtraction. Each has 20 numbers. Mental addition gives 20 numbers, one at a time, to be added mentally. At the end, the answer is to be typed on the keyboard, and is then compared to the correct answer.

Mental addition/subtraction starts with a random number between 26 and 35. Then 20 numbers appear, one at a time, on the screen. Some are to be added to the running total, others subtracted. Again, the answer is to be typed on the keyboard and compared to the real answer. These two options only use single digit numbers for the 20 which appear on the screen.

The programs, as presented here, are written for GW BASIC under MS-DOS. Appropriate changes for machines that use PRINT@ and have smaller screens are noted along with each program where those changes apply. The delay timing loops should be easy to spot, and may need to be changed for computers which run at different clock speeds than the one they were developed on. ■

```
100 REM * DRILL.BAS * MAIN MENU PGM * CODEWORKS MAGAZINE 3838 SOUTH
110 REM * WARNER ST. TACOMA, WA 98409 206-475-2219 VOICE AND
120 REM * 206-475-2356 300/1200 BAUD MODEM
130 REM * WRITTEN FOR CODEWORKS BY BOB HENKEL, SPANAWAY, WASHINGTON
140 CLS:'USE YOUR PARTICULAR CLEAR SCREEN COMMAND HERE
150 'CLEAR 1000:'USE ONLY IF YOU NEED TO CLEAR STRING SPACE
160 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
170 PRINT"      I N S T R U C T O R' S   D R I L L   P R O G R A M"
180 PRINT"            a collection of exercises by Bob Henkel"
190 PRINT STRING$(60,45)
200 PRINT
210 PRINT TAB(10)"1-Addition";TAB(30)"6-Mental Addition
220 PRINT TAB(10)"2-Subtraction";TAB(30)"7-Mental
    Addition/Subtraction
230 PRINT TAB(10)"3-Multiplication";TAB(30)"8-Update Spelling Files
240 PRINT TAB(10)"4-Division";TAB(30)"9-End Session
250 PRINT TAB(10)"5-Spelling
260 PRINT
270 LINE INPUT"Enter the number of your choice ";X$
280 X=VAL(X$)
290 IF X<1 OR X>9 THEN 270
300 ON X GOTO 310,320,330,340,350,360,370,380,390
310 RUN"ADD.BAS
320 RUN"SUB.BAS
330 RUN"MULT.BAS
340 RUN"DIV.BAS
350 RUN"SPELL.BAS
360 RUN"MADA.BAS
370 RUN"MADAS.BAS
380 RUN"UPDATE.BAS
390 CLS
400 PRINT"End of Session."
410 END
```

Many machines (notably the Tandy Models I, III and IV) do not require the RANDOMIZE statements in lines 120 and 130 of most of these programs. Those machines also will need a different RND statement, and in some cases, PRINT@ instead of LOCATE. These will be called out where they occur in the listings.

```
100 REM * ADD.BAS * ADDITION PROGRAM USED WITH DRILL.BAS *
110 CLS:'Use your own clear screen command here.
120 REM * Seed the random number generator from Time$
130 A=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE A
140 PRINT TAB(10)" ----- Addition Problems -----"
```

```
150 PRINT
160 PRINT"You will be given 20 addition problems with a limited"
170 PRINT"time to answer each problem. If you miss no more than"
180 PRINT"one problem, the problems will get harder. If you miss"
190 PRINT"more than five, they get easier.
200 PRINT"You cannot correct errors, so be careful."
210 PRINT:INPUT"What is the largest addend to be used";LA
220 IF LA<3 THEN LA=3
230 CLS:PRINT"Largest addend is";LA
240 FOR X=1 TO 20
250   A=INT(RND*(LA-1)+1)
260   B=INT(RND*(LA-2)+2)
270   PRINT A;" + ";B;" = ";
280   IF A+B<10 OR A*B<10 THEN 310
290   FOR Y=1 TO 4000:T$=INKEY$:IF T$="" THEN NEXT Y
300   IF T$="" THEN 320 ELSE 310
310   FOR Z=1 TO 3000:U$=INKEY$:IF U$="" THEN NEXT Z ELSE 320
320   PRINT T$+U$;" ";
330   IF VAL(T$)*10+VAL(U$)=A+B THEN SC=SC+1 ELSE PRINT;" the correct
      answer is";A+B;:FOR Y=1 TO 4000:NEXT Y:V$=INKEY$:W$=INKEY$:
      Z$=INKEY$
340   PRINT
350   T$="":U$=""
360 NEXT X
370 PRINT
380 CT=CT+1:TL=TL+SC
390 PRINT"Score on last 20 problems is";SC*5;"%":PRINT"Score on all
    problems is";INT((TL*5/CT)+.5);"%"
400 IF SC>18 THEN LA=LA+1
410 IF SC<15 THEN LA=LA-1
420 SC=0:PRINT"Would you like to do another set (Y/N)?"
430 CU$=INKEY$:IF CU$="" THEN 430
440 IF CU$="Y" OR CU$="y" THEN 230 ELSE IF CU$="N" OR CU$="n" THEN
    RUN"DRILL.BAS" ELSE 430
```

Alternate use:
A=RND(LA—1)+1
B=RND(LA—2)+2

```
100 REM * SUB.BAS * SUBTRACTION PROGRAM USED WITH DRILL.BAS *
110 CLS:'Use your own clear screen command here.
120 REM * Seed the random number generator from Time$
130 A=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE A
140 PRINT TAB(10)" ----- Subtraction Problems -----"
150 PRINT
160 PRINT"You will be given 20 subtraction problems with a limited"
170 PRINT"time to answer each problem. If you miss no more than"
180 PRINT"one problem, the problems will get harder. If you miss"
190 PRINT"more than five, they get easier.
200 PRINT"You cannot correct errors, so be careful."
210 PRINT:INPUT"What is the largest subtrahend to be used";LA
220 IF LA<3 THEN LA=3
230 CLS:PRINT"Largest subtrahend is";LA
240 FOR X=1 TO 20
250   A=INT(RND*(LA-1)+1)
260   B=INT(RND*(LA))
270   PRINT A+B;" - ";A;" = ";
```

Alternate use:
A=RND(LA—1)+1
B=RND(LA)

```
280    IF B<10 THEN 310
290    FOR Y=1 TO 4000:T$=INKEY$:IF T$="" THEN NEXT Y
300    IF T$="" THEN 320 ELSE 310
310    FOR Z=1 TO 2000:U$=INKEY$:IF U$="" THEN NEXT Z ELSE 320
320    PRINT T$+U$;" ";
330    IF VAL(T$)*10+VAL(U$)=B THEN SC=SC+1 ELSE PRINT;" the correct
       answer is";B;:FOR Y=1 TO 4000:NEXT Y:V$=INKEY$:W$=INKEY$:
       Z$=INKEY$
340    PRINT
350    T$="":U$=""
360 NEXT X
370 PRINT
380 CT=CT+1:TL=TL+SC
390 PRINT"Score on last 20 problems is";SC*5;"%":PRINT"Score on all
    problems is";INT((TL*5/CT)+.5);"%"
400 IF SC>18 THEN LA=LA+1
410 IF SC<15 THEN LA=LA-1
420 SC=0:PRINT"Would you like 20 more (Y/N)?"
430 CU$=INKEY$:IF CU$="" THEN 430
440 IF CU$="Y" OR CU$="y" THEN 230 ELSE IF CU$="N" OR CU$="n" THEN
    RUN"DRILL.BAS" ELSE 430
```

```
100 REM * MULT.BAS * MULTIPLICATION PROGRAM USED WITH DRILL.BAS *
110 CLS:'Use your own clear screen command here.
120 REM * Seed the random number generator from Time$
130 A=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE A
140 PRINT TAB(10)" ----- Multiplication Problems -----"
150 PRINT
160 PRINT"You will be given 20 multiplication problems with a limited"
170 PRINT"time to answer each problem. If you miss no more than"
180 PRINT"one problem, the problems will get harder. If you miss"
190 PRINT"more than five, they get easier.
200 PRINT"You cannot correct errors, so be careful."
210 PRINT:INPUT"What is the largest multiplier to be used";LA
220 IF LA<3 THEN LA=3
230 CLS:PRINT"Largest multiplier is";LA
240 FOR X=1 TO 20
250   A=INT(RND*(LA-1)+1)
260   B=INT(RND*(LA-2)+2)
270   PRINT A;" x ";B;" = ";
280   IF A*B<100 AND A*B>9 THEN 320
290   IF A*B<10 THEN 340
300   FOR V=1 TO 4000:H$=INKEY$:IF H$="" THEN NEXT V
310   IF H$="" THEN 350
320   FOR Y=1 TO 3000:T$=INKEY$:IF T$="" THEN NEXT Y
330   IF T$="" THEN 350 ELSE 340
340   FOR Z=1 TO 2000:U$=INKEY$:IF U$="" THEN NEXT Z ELSE 350
350   PRINT H$+T$+U$;" ";
360   IF VAL(H$)*100+VAL(T$)*10+VAL(U$)=A*B THEN SC=SC+1 ELSE PRINT;"
      the correct answer is";A*B;:FOR Y=1 TO 4000:NEXT Y:V$=INKEY$:
      W$=INKEY$:Z$=INKEY$
370   PRINT
```

Alternate use:

```
A=RND(LA-1)+1
B=RND(LA-2)+2
```

```
380    H$="":T$="":U$=""
390 NEXT X
400 PRINT
410 CT=CT+1:TL=TL+SC
420 PRINT"Score on last 20 problems is";SC*5;"%":PRINT"Score on all
    problems is";INT((TL*5/CT)+.5);"%"
430 IF SC>18 THEN LA=LA+1
440 IF SC<15 THEN LA=LA-1
450 SC=0:PRINT"Would you like another set of 20 (Y/N)?"
460 CU$=INKEY$:IF CU$="" THEN 460
470 IF CU$="Y" OR CU$="y" THEN 230 ELSE IF CU$="N" OR CU$="n" THEN
    RUN"DRILL.BAS" ELSE 460


100 REM * DIV.BAS * DIVISION PROGRAM USED WITH DRILL.BAS *
110 CLS:'Use your own clear screen command here.
120 REM * Seed the random number generator from Time$
130 A=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE A
140 PRINT TAB(10)" ----- Division Problems -----"
150 PRINT
160 PRINT"You will be given 20 division problems with a limited"
170 PRINT"time to answer each problem. If you miss no more than"
180 PRINT"one problem, the problems will get harder. If you miss"
190 PRINT"more than five, they get easier.
200 PRINT"You cannot correct errors, so be careful."
210 PRINT:INPUT"What is the largest divisor to be used";LA
220 IF LA<3 THEN LA=3
230 CLS:PRINT"Largest divisor is";LA
240 FOR X=1 TO 20                                  Alternate use:
250   A=INT(RND*(LA-1)+1)                          A=RND(LA-1)+1
260   B=INT(RND*(LA-1)+1)                          B=RND(LA-1)+1
270   PRINT A*B;" / ";A;" = ";
280   IF B<10 THEN 310
290   FOR Y=1 TO 4000:T$=INKEY$:IF T$="" THEN NEXT Y
300   IF T$="" THEN 320
310   FOR Z=1 TO 2000:U$=INKEY$:IF U$="" THEN NEXT Z ELSE 320
320   PRINT T$+U$;" ";
330   IF VAL(T$)*10+VAL(U$)=B THEN SC=SC+1 ELSE PRINT;" the correct
      answer is";B;:FOR Y=1 TO 4000:NEXT Y:V$=INKEY$:W$=INKEY$:
      Z$=INKEY$
340   PRINT
350   T$="":U$=""
360 NEXT X
370 PRINT
380 CT=CT+1:TL=TL+SC
390 PRINT"Score on last 20 problems is";SC*5;"%":PRINT"Score on all
    problems is";INT((TL*5/CT)+.5);"%"
400 IF SC>18 THEN LA=LA+1
410 IF SC<15 THEN LA=LA-1
420 SC=0:PRINT"Do you want to do another set (Y/N)?"
430 CU$=INKEY$:IF CU$="" THEN 430
440 IF CU$="Y" OR CU$="y" THEN 230 ELSE IF CU$="N" OR CU$="n" THEN
    RUN"DRILL.BAS" ELSE 430
```

```
100 REM * SPELL.BAS * FOR USE WITH DRILL.BAS PGM * SPELLING DRILL
110 DIM WO$(20)
120 CLS
130 PRINT
140 PRINT TAB(10)"----- Spelling Drill -----"
150 PRINT
160 INPUT"What is the number of the spelling lesson.";A$
170 INPUT"What is the grade level (4,5,6,7,8)";GL$
180 IF VAL(GL$)<4 OR VAL(GL$)>8 THEN 170
190 U$="UNIT"+GL$+A$
200 OPEN"I",1,U$
210 FOR X=1 TO 20
220   INPUT #1,WO$(X)
230 NEXT X
240 CLOSE
250 CLS
260 PRINT"When you see a word spelled letter by letter at the"
270 PRINT"center top of the screen, you enter the correct spelling"
280 PRINT"for that word.
290 PRINT"If you get the word on the first try you get 5 points."
300 PRINT"If you get the word on the second try you get 3 points."
310 PRINT"If you get the word on the third try you get only 1 point."
320 PRINT
330 INPUT"Press ENTER when you are ready to start.";A
340 CLS
350 PRINT"These are the words you will by asked to spell."
360 PRINT
370 FOR X=1 TO 20 STEP 2
380   PRINT X;TAB(5);WO$(X);TAB(30);X+1;TAB(35);WO$(X+1)
390 NEXT X
400 PRINT
410 INPUT"Press ENTER when you are ready to continue.";A
420 CLS
430 FOR X=1 TO 20
440   FOR Y=3 TO 1 STEP -1
450     L=LEN(WO$(X))
460     FOR Q=1 TO 400:NEXT Q: ' one of many delay loops
470     FOR Z=1 TO L
480       LOCATE 3,Z+30:PRINT MID$(WO$(X),Z,2)
490       FOR Q=1 TO 200:NEXT Q: ' sets speed of the letter's
          appearance
500       LOCATE 3,Z+30:PRINT" ": ' one space between quotes here
510     NEXT Z
520     CLS
530     LOCATE 10,5:INPUT"Type the word that appeared on the screen.";
        AN$
540     IF WO$(X)<>AN$ THEN 600
550     SC=SC+2*Y-1:PRINT"Score..... =";SC
560     PRINT"Good work, you spelled <";WO$(X);"> correctly."
570     PRINT"Watch the top center of the screen for the next word."
580     FOR Q= 1 TO 1000:NEXT Q
590     GOTO 680
600     IF Y=3 THEN PRINT"You missed the word on the 1st try."
610     IF Y=2 THEN PRINT"You missed the word on the 2nd try."
```

```
620     IF Y=1 THEN PRINT"The correct spelling is: ";WO$(X)
630     FOR Q=1 TO 1000:NEXT Q
640     IF Y=3 OR Y=2 THEN PRINT"Watch the top center of the screen."
650     IF Y=1 THEN PRINT"Your three tries are gone. Watch for the
        next word."
660     FOR Q=1 TO 2000:NEXT Q
670   NEXT Y
680 NEXT X
690 PRINT"Your score on the last 20 words is ";SC:CT=CT+1
700 TS=TS+SC:SC=0
710 PRINT"Your total score is ";:PRINT USING "###.#";TS/CT
720 PRINT"Do you wish to try another lesson (Y/N)"
730 A$=INKEY$:IF A$="" THEN 730
740 IF A$="Y" OR A$="y" THEN GOTO 140
750 IF A$="N" OR A$="n" THEN RUN"DRILL.BAS" ELSE 720


100 REM * MADA.BAS * MENTAL ADDITION PROGRAM USED WITH DRILL.BAS *
110 CLS:'Use your own clear screen command here.
120 REM * Seed the random number generator from Time$
130 A=VAL(MID$(TIME$,4,2)+MID$(TIME$,7,2)):RANDOMIZE A
140 PRINT TAB(10)"----- Mental Addition -----"
150 PRINT"This is a program to practice mental addition."
160 PRINT"You will be given 20 numbers to add mentally.
170 INPUT"Press ENTER when you are ready to start";A
180 Z=0
190 FOR X=1 TO 20
200   Y=INT(RND*9)+1                Y=RND(9)
210   LOCATE 6,40:PRINT "+";Y       64-column PRINT@    80-column PRINT@
220   Z=Z+Y                         414                 520
230   IF X=20 THEN 270
240   FOR T=1 TO 3000:NEXT T
250   LOCATE 6,40:PRINT"    "      414                 520
260   FOR Y=1 TO 300:NEXT Y
270 NEXT X
280 INPUT"What is your answer";AN
290 IF AN=Z THEN PRINT"Great Job!" ELSE PRINT"The correct answer is:";
    Z
300 PRINT"Do you wish to try again (Y/N)"
310 A$=INKEY$:IF A$="" THEN 310
320 IF A$="Y" OR A$="y" THEN 110 ELSE RUN "DRILL.BAS"


100 REM * UPDATE.BAS * FOR USE WITH DRILL.BAS PROGRAM TO UPDATE
    SPELLING FILES
110 DIM WO$(20)
120 PRINT
130 CLS
140 PRINT TAB(5)"----- Update Spelling Lesson Files -----"
150 PRINT
160 PRINT TAB(10)"1- Create a new Spelling file"
170 PRINT TAB(10)"2- Edit an existing Spelling file"
180 PRINT TAB(10)"3- END and Return to Drill Program
190 PRINT TAB(10)"4- END Session and Quit
```

```
200 PRINT
210 PRINT TAB(10)"The number of your choice is ";:INPUT A
220 IF A<1 OR A>4 THEN 130
230 ON A GOTO 240,430,560,570
240 PRINT"What is the number of this new spelling lesson";:INPUT A$:
    A=VAL(A$)
250 PRINT"What is the grade level of this lesson (4,5,6,7,8)";:INPUT
    GL$
260 IF VAL(GL$)<4 OR VAL(GL$)>8 THEN 250
270 CLS:PRINT"Enter your words as each number appears."
280 FOR X=1 TO 20
290 PRINT X;TAB(5);:INPUT WO$(X)
300 NEXT X
310 CLS
320 FOR X=1 TO 20 STEP 2
330 PRINT X;TAB(5);WO$(X);TAB(30);X+1;TAB(35);WO$(X+1)
340 NEXT X
350 U$="UNIT"+GL$+RIGHT$(A$,LEN(A$))
360 PRINT"Now saving file ";U$
370 OPEN "O",1,U$
380 FOR X=1 TO 20
390   PRINT #1,WO$(X)
400 NEXT X
410 CLOSE 1
420 GOTO 130
430 INPUT"ENTER the number of the lesson to correct";A$
440 INPUT"What is the Grade Level (4,5,6,7,8)";GL$
450 U$="UNIT"+GL$+A$
460 OPEN "I",1,U$
470 FOR X=1 TO 20
480   INPUT #1,WO$(X)
490 NEXT X
500 CLOSE 1
510 PRINT"Press ENTER to continue, or retype the word."
520 FOR X=1 TO 20
530   PRINT X;TAB(5);WO$(X);:W$=WO$(X):INPUT WO$(X):IF WO$(X)="" THEN
    WO$(X)=W$
540 NEXT X
550 GOTO 370
560 RUN"DRILL.BAS"
570 CLS:PRINT"Update Session is ended.":END
```

SOX

A new way of making loops

The ability to program loops is one of the more powerful attributes of any langauge. Here is a new way to do it with the control of the loop all in one line of code. It can also be made to create non-linear sequences.

Sox what? You have never heard of it before. It was discovered at CodeWorks, by Jay Marshall, several months ago.

SOX stands for **S**tring **O**rdinal Inde**X**ing. It works using strings, and the ordinal positions within a string. In case you forgot, ordinal numbers are first, second, third, etc., while the cardinal numbers are 1, 2, 3, etc.

Using SOX, you can format an entire video screen in *one line of code*. You can format both the horizontal and vertical print positions on your printer using it. How about non-linear progressions? Standard loops always count in even increments, either increasing or decreasing. With SOX you can program, again in one line, a totally non-linear series which repeats. It also works well with program flow control, where what happens in a subroutine needs to change the flow of the program. It can be used very effectively to eliminate long series of IF THEN statements.

Granted, all these things can just as well be done with standard loops, but there are times when SOX can eliminate many lines of code. Aside from that, it puts the entire control sequence in one string, where it can easily be reached and (admittedly, not so easily) modified. It also has a few shortcomings. But enough of this, what is it anyway?

To illustrate the principle, let's take a very simple example.

```
Q=ASC(MID$("ABCDA",Q+1,1))-64
```

You have seen something similar before, but not quite. There are a couple of differences here. First of all, assume that variable Q starts at zero (which it will when you RUN a program.) Every time your program encounters the above line, Q will be incremented by one in the line itself. Now, when Q is zero and we hit this line, it will be incremented by one (to 1), and the line will find the ASCII value of capital letter A (65) from the literal string ABCDA, and subtract 64 from it, leaving Q=1. The next time we encounter this line, Q is already equal to one, so it now gets bumped to 2 and we pull out the value for B less 64, which is 2. This continues each time we reach this line, until we get to the literal D (where Q=4). Here, the loop will close itself because after D we have another A in the literal string. It does it because Q takes the value from the *ordinal* position of the literal string, and the next letter after D is A (value 65-64=1). The next time through, Q+1 will put us at string position two, which is B (66-64=2), and so on. This line will give us a 1, 2, 3, 4, 1, 2, 3, 4, 1, 2,.... sequence. No big deal, a simple For...Next will do it easier. But wait, there's more.

The two things that set this line of code off from a normal line is the "Q+1" and the "-64" at the end of the line. This line of code, plus a GOTO, is an incrementing loop. It loops because the literal string starts with A and ends with A, effectively resetting it. Notice that the loop will not produce a zero. The incrementing device is the factor Q+1 inside the line. The -64 brings the ASCII value of the literal string back into correspondence with the place positions of the literal string itself. Note that, if prior to reaching this line, you initialized Q to be 2, then the loop would start at 3 *the first time only* and then fall into the 1, 2, 3, 4 sequence. Further, the literal string "ABCDA" inside the line need not be there. It can be defined elsewhere, say as A$="ABCDA", and then the line could read: Q=ASC(MID$(A$,Q+1,1))-64. You can even have several lines defining different strings, and somewhere during processing elsewhere, equate them to A$ for a whole new sequence. Now let's try

some variations on this theme.

Type in the following little program. We are going to use it for several examples:

```
10 CLS
20 A$="ABCDEFGHIJA"
30 Q=ASC(MID$(A$,Q+1,1))- 64
40 LOCATE Q,1:PRINT Q;STRING$(50,95)
50 GOTO 30
```

This example is for GW BASIC. Those with 64-column, 15 line screens, change line 40 to: PRINT@ Q*64,;Q; etc., and those with 80-column screens change it to PRINT@ Q*80,;Q; etc. In addition, those computers using memory mapped video may need to add a semicolon at the very end of line 40 to keep the cursor from being destructive.

This little program will print ten lines of underline on the screen from line 1 to 10. It will place the number of the line at the beginning of the line.

Now let's make it print from the bottom up. Change line 20 to: 20 A$="JJABCDEFGHI". Before running, Q=0, so the first time through line 30, Q will take the value 10 from the J in A$. But that will position the next "read" of the string to ordinal position 10, where the Q+1 will point to the "I" in A$, which will make Q=9. This will continue, stepping backwards along A$ until "J" is reached, which sends Q back down the line to repeat the sequence. Now how about some non-linear sequences?

Before we can do that, we need to define something as a "place holder" in A$. To see what that means, change line 20 to: 20 A$="AC.DG..A" The periods in A$ are there as place holders. This new A$ will repeat the sequence 1, 3, 4, 7. Try it and watch your screen. If you need a sequence that goes like this: 1, 8, 12, 2, 5, 15 change A$ to "AHE..O..L...B..A", and it will give them to you *in that order*.

Instead of printing on the screen, we could have an ON Q GOSUB statement, which would take us to various subroutines in which the value of Q could be incremented or decremented prior to returning depending on the outcome of the processing in the subroutine. Entire control sequences can be written in one string this way.

One that comes to mind is a sequence that controls TAB positions for LPRINT, although it would need to be increasing from smallest to largest, since you cannot tab backwards on most printers.

It is possible to have more than one loop operating in one string. The variable Q never stops at a place holder position. In those positions, you can set up another sequence, and then, external to the loop, force Q from one of the repeating sequences to the other or others. One restriction is that the two (or more) loops cannot share a common value.

What about zero? If you need a sequence that includes zero, set up A$ to go from 1 to one more than you need. Then, let another variable equal Q, once Q is calculated, and let the other variable equal Q less one, as in: S=Q-1.

We have set up A$ to include the capital letters A through Z, the following six ASCII symbols, and then the lower case letters a through z. It gives an interesting range to work with, and could possibly be extended to the higher ASCII characters as well.

Until now, we have looked at the ASC(MID$ of a string of letters. We can do similar things with the VAL(MID$ of a string of numbers. The following line will produce the numbers 1 through 12 in ascending sequence and repeat it:

```
Q=VAL(MID$(".0102030405060708091011120 1",
                Q*2+2,2))
```

Note that the first position of the literal string is a place holding period. It is necessary because we are looking at two positions at a time and since initially Q is equal to zero, we want to start down the string at position 2.

To make this scheme count from 12 to 1 and repeat, the string would look like this: ".12120102030405060708091011"

Like the code shown earlier, the above string will start at 1 assuming that Q is initialized at zero when the RUN command is given. Prior to entering this code, however, Q may be initialized to any number within the range of the literal string. The count will then start at the *next* number, that is, the space number in the string plus one.

If you try to force Q to a number resulting in zero or outside the range of the string ordinal positions, you will encounter the "Illegal Function Call" error. You are assured of seeing plenty errors of this type as you play with this idea.

Deciding what goes where inside the string is relatively easy after you get the "feel" for how it works. This is especially true for simple linear increasing or decreasing series where all integer values are present. How to decide where and what letters go into the string for non-linear series is not so easy. It helps to simply play with three or four letters to begin with. Somehow, letters seem to be easier to understand than numbers.

If the SOX line is Q=ASC(MID$("ABCDEFGH A",Q+1,1))-64, it counts from 1 to 8 and repeats. Let's make it count 2, 4, 6, 8 and repeat by changing the literal string within this statement. Assuming that Q was initialized at zero when we RUN, it appears that the first number we want in our series *must* be in the first string position. Since we want a 2, let's put a B in the first position. Now Q+1 will get us to that position, which contains the B we put there. This results in Q taking the value of 2. The next time we get to this line of code (with Q=2), we will be positioned at the second position of the string. However, the Q+1 will move us up one position to the third position. So we put a place holder (period or any other character) at position two in the string and put the letter representing a 4 (letter D) in the third position. Now Q takes on the value of 4. On the next pass through this line of code, Q will equal 4, which puts us at the fourth position of the string and the Q+1 will push us to the fifth position. So, the fourth position should be another place holder character and the fifth place should hold the letter F if we want a 6 from it. Next time we hit position six, plus the 1, puts us into position 7 where we want to see an 8 so we put the letter H in the seventh position. The last time around, with Q=8 because H=8, will put us into the eighth position and the Q+1 will advance us to the ninth position, where we want to put another B to make the cycle repeat. The complete string for the 2, 4, 6, 8 sequence is "B.D.F.H.B"

It appears that the first number we want must always go into the first position of the string. The next letter then must go into the string position represented by the *ASCII value less 64* of the previous letter plus one. In the above example, the D went into the position pointed to by B (which equals 2) plus 1, or position 3. The H went into the position pointed to by the F (6) plus 1 equals position seven. The closing B went into the position pointed to by the H (8) plus 1, or position nine. The cycle repeats because when Q takes on position nine, its value is again 2 and the Q+1 puts it at position three, where it takes the value of 4 from the D there.

The B at the first position in the string is used only once, the first time the line of code is encountered in the program. After that, it is simply skipped over like any other place holder character. In fact, this is true even for reverse counts, as in "HHABCDEFG", where the first H causes Q to be 8 and moves our "invisible pointer" down to the F, which is incremented by one by the Q+1 to G where Q takes the value of 7. The H in position one is used the first time the line of code is encountered while the second is a place holder. After the first time the second H becomes operative and the first serves as a place holder.

At first it seemed to be that the number of spaces in the string must equal the value of the largest number you wish to represent plus one. But then we ran into these that made us reconsider: The sequence 1, 3, 4, 8 is represented by the string "AC.DH..A" (eight spaces). The sequence 1, 3, 4, 7 is represented by the string "AC.DG..A" (eight spaces), but the sequence 1, 6, 7, 8 is represented by the string "AF....GHA" (nine spaces).

SOX can be used as loops by themselves, with one GOTO. They can also be used very effectively inside For...Next or While...Wend loops. It is possible for one SOX loop to be nested within another SOX loop. Learn how to do the single loop first, the nested SOX are difficult to set up.

You can jump into and out of a SOX loop with ease. There is no stack to worry about and no string assignment (unless you start to equate the string inside the statement to other strings outside of it.)

We have yet to find an application for SOX that couldn't be done by conventional means. In many cases, however, SOX made a significant reduction in program lines. What SOX really needs is an application that can't be done any other way. It appears to be a solution looking for a problem. If you didn't get the jist of this whole affair through this discussion, type in the sample and play with it. It's an interesting concept, and if you find a unique application for it, let us know. ●

.....

The two programs which follow are both memory dump programs using SOX techniques. The first is for GW-BASIC and MS-DOS, and allows you to see all of memory, even if you have 630K! It provides a rolling display of hex-dump after you provide it with the segment numbers (0 through 15) and the starting and ending address (0000 to 65535) in each segment.

The second program is a repeat of the ASCII dump program from last issue, this time using the SOX technique to roll the display below the heading on the screen. It should work on any computer that uses the PEEK command. Because this is a repeat program, it will not be on the current issue download menu.

Hex dump program for MS-DOS machines

```
100 DEFINT C-Z :REM * MUMP.BAS * HEX MEMORY DUMP FOR MS-DOS *
110 CLS:L=3
120 INPUT"Enter Segment #(0-15)";BLK
130 INPUT"Enter Start Address  ";A1
140 INPUT"Enter Ending Address ";A2
150 DEF SEG=BLK*2^12
160 A1=((A1/2) AND &HFFF8)*2
170 CLS:PRINT"Block ";BLK;HEX$(BLK),"From ";A1;HEX$(A1),"To ";A2;HEX$(A2)
180 WHILE A1<=A2
190   LOCATE L,1:L=ASC(MID$("ABDDEFGIIJKLNNOPQSSTUVDFYZ",L,1))-63
200   PRINT HEX$(BLK);".";RIGHT$("000"+HEX$(A1),4);":  ";
210   FOR A=A1 TO (A1+15)
220     PRINT" ";RIGHT$("0"+HEX$(PEEK(A)),2);
230   NEXT A:PRINT":   ";
240   FOR A=A1 TO (A1+15)
250     CH=PEEK(A)
260     IF CH<32 OR CH>126 THEN PRINT".";ELSE PRINT CHR$(CH);
270   NEXT A:A1=A1+16:PRINT ,,;
280 WEND:BEEP
290 IF INKEY$="" THEN 290 ELSE 110
```

ASCII dump program

```
100 REM ADUMP/BAS * ASCII DUMP PROGRAM * CODEWORKS
110 REM * 3838 S. WARNER ST. TACOMA, WA 98409. (206)475-2219
120 CLEAR 1000
130 CLS:' This is a CLEAR SCREEN command.
140 PRINT STRING$(22,"-");" The CodeWorks ";STRING$(23,"-")
150 PRINT "            A S C I I   D U M P   P R O G R A M "
160 PRINT "     Displays memory - Use Space Bar for Pause/Continue"
170 PRINT STRING$(64,"-")
180 PRINT
190 PRINT "MEMORY RANGE FROM 0000 TO 32767."
200 PRINT
210 DEFINT A,B,C,I:KY=1
220 INPUT "STARTING MEMORY (DECIMAL) ADDRESS";A
230 INPUT "  ENDING MEMORY (DECIMAL) ADDRESS";B
240 IF B=0 THEN B=A+2000
250 L=ASC(MID$("E....FGHIJKLMNE",L+1,1))-64
260 PRINT @64*L,RIGHT$("    "+STR$(A),5);": ";
270 FOR I=A TO A+7
280   PRINT"  "+RIGHT$("    "+STR$(PEEK(I)),3);
290 NEXT I
300 PRINT "    ";
310 FOR I=A TO A+7
320   C=PEEK(I)
330   IF C<32 OR C>126 THEN PRINT ".";ELSE PRINT CHR$(C);
340 NEXT I
350 A=A+8
360 KY$=INKEY$:IF KY$<>""THEN KY=3-KY
370 IF KY=2 THEN 360
380 IF A<B THEN 250 ELSE 100
390 STOP
```

FMAKER.Bas

A general utility for making files

Staff Project. There are so many times when you need a quick way to make a sequential file that we finally wrote it. Then we added the ability to do limited editing on the data. There are so many other things we could have added, it was hard to stop. We really didn't want a full-blown file editor, but it is something to think about.

CP/M and MS-DOS have the ability to create and enter data into a file from the console (keyboard). Many other machines do not have this utility, and so we have designed this short program to do the job. This utility, FMAKER.Bas (for filemaker) should work on any machine using Microsoft BASIC.

The program can be used to create a sequential file, enter data into it, and recall the data and do a limited amount of editing before replacing it. The editing is limited to line editing, which means that the entire line in error needs to be re-entered. It is intended to create small sequential control files. In MS-DOS, for example, it can be used to create AUTOEXEC.BAT files. When we present our report generator for CARD.BAS, this program can be used to generate the report formats needed.

The program starts with the usual opening menu and three options. The one string array used is A$, and is dimensioned at 100. The three menu options are to create a new file, change an existing file and to quit. Lines 210 through 230 do error checking for input within the proper range.

The section of code that controls the input of data into a new file is between lines 240 and 340. F$ takes on the filename you enter. The use of period and end (.end or .END) terminates the entry of lines, but does not become a part of the data itself. The data is input through the loop between lines 290 and 320. In line 310 a check is made to see if the input is equal to .end or .END, and if it is, control jumps to line 330 where variable N is set equal to I less 1 to get rid of the last .end or .END entry now that it has done its job.

The little loop in line 340 simply prints out the list of items you have entered. The code following, from lines 350 through 390, opens the sequential file on disk and prints the items into it.

Option 2 of the menu allows you to recall a file and change it. This code resides between lines 420 and 570. The first part of this section, from lines 420 through 480, first asks what the filename is, then opens the file and reads the data into the A$ array in memory. In line 450, inside the loop, an end of file (EOF) check is made so that we read to the end of file and then exit the loop. After the data is read in, line 490 again sets the variable N equal to I less one. Earlier, we did this to get rid of the .end sentinel. This time we do it because the I counter is already advanced to the next number and we do not want a blank line added to our data.

Now that the data is in the A$ array in memory, we present it on the screen using the lines from 500 through 520. In this case, we print the value of I first and then the data following it. This gives us a line number reference to use if we should want to change any of the lines.

Since we did not use array position zero (note that the loops all start from one) we can use the zero to tell us that no more lines need be changed. Otherwise, in lines 530 and 560, when we input the line number to be changed, it becomes A$(I), and puts our changes back into the proper place. The way the lines loop back between 530 and 570 provide for multiple changes without going back to the menu after each change.

When all the changes are made and you enter a zero, the program flow goes back to line 350, where the updated file is written back to the diskette. After this, the program flow takes you back to the menu in line 400.

Since we are using the LINE INPUT statement, virtually any data can be entered using this program. LINE INPUT accepts an entire line (a maximum of 254 characters) from the keyboard, including delimiters (such as commas, quotation marks, etc.).

In our next issue we plan to present the report generator (ReportCard) that will work with CARD.BAS from Issue 2. That program will depend on a small control file to tell it what to do. This program, in addition to other uses, can be used to generate that control file. ■

```
100 REM * FMAKER.BAS * CodeWorks Magazine Sequential file maker *
110 DIM A$(100)
120 CLS: 'This is a clear screen command
130 'CLEAR 5000:'Use only if your machine needs to clear string space
140 PRINT STRING$(22,45);" The CodeWorks ";STRING$(23,45)
150 PRINT"                        F I L E   M A K E R
160 PRINT"   creates and allows limited change to sequential files"
170 PRINT STRING$(60,45)
180 PRINT TAB(10)" 1 - Create a new file
190 PRINT TAB(10)" 2 - Change an existing file
200 PRINT TAB(10)" 3 - Quit
210 PRINT" Your choice";:INPUT X
220 IF X <1 OR X>3 THEN 210
230 ON X GOTO 240,410,580
240 CLS
250 INPUT"What is the name for your new file";F$
260 PRINT"Input your lines of data, terminate each line with ENTER"
270 PRINT"Use period and end (.end or .END) to quit entering data."
280 PRINT
290 FOR I=1 TO 100
300   LINE INPUT A$(I)
310   IF A$(I)=".end" OR A$(I)=".END" THEN 330
320 NEXT I
330 N=I-1
340 FOR I=1 TO N:PRINT A$(I):NEXT I
350 OPEN "O",#1,F$
360   FOR I=1 TO N
370     PRINT #1,A$(I)
380   NEXT I
390 CLOSE 1
400 GOTO 120
410 CLS
420 INPUT"Enter the filename you wish to work with";F$
430 OPEN "I",#1,F$
440   FOR I=1 TO 100
450     IF EOF(1) THEN 480
460     LINE INPUT #1,A$(I)
470   NEXT I
480 CLOSE 1
490 N=I-1
500 FOR I=1 TO N
510   PRINT I;A$(I)
520 NEXT I
530 INPUT"Which line to change (0 for none).";I
540 IF I=0 THEN 120
550 LINE INPUT"Enter the entire new line ";A$(I)
560 INPUT"Enter 0 if done, next line number for more changes";I
570 IF I=0 THEN 350 ELSE 550
580 CLS:END
```

Renewal time: It's hard to believe, but a whole year has zipped on by us. It's that time when we need to ask all of you to renew your subscription to **CodeWorks**.

Based on your input, we see the next year as a continuation of the style of articles and programs we presented this year. We have a very nice list of projects to work on during the comming months.

Those of you who would like to renew via our download may do so by using the DEMO "Signup" option. If you do that, simply put your current subscriber number after your name so that we can determine that you are a renewal and not a new subscriber.

When you renew, you will retain your current subscriber number. This will be true on all renewals received here before the Nov/Dec 86 issue gets sent out (about the 20th of October.)

Because we didn't quite come up to the number of subscribers we had planned for, there are still a few hundred complete sets of the first year of CodeWorks issues available. If you are missing any of this year's issues, be sure and let us know while they are still available.

Your early renewal will allow us to set our operating budget for the coming year. We thank you for your continued support and encouragement.

Subscription ORDER FORM *986*

Computer type:

Do you have a modem?
If so, what baud rate?

Comments:

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

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

Please Print Clearly:

Name _____

Address _____

City _____ State _____ Zip _____

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

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

Download

What's Happening on the Download

Just after the last issue went into the mail we finally installed the 1200 baud option to our download system. The most significant change you will see now on the download is the initial prompt (at 300 baud) that asks you to press RETURN for a speed check. This lets our system know what baud you are using. If you call us at 1200 baud, the same message appears but will probably be unreadable. Simply press RETURN (or ENTER) twice to wake us up at 1200 baud. Be aware that at the higher baud rate the opportunity for noise to get into the transmission is somewhat greater.

There seems to be some confusion with many new subscribers about just how to get on to the download system. There have been messages left on the DEMO side of the board asking what the "other" telephone number for subscribers is. There are also several messages that seem to imply that the subscriber number on your label is the password. Neither of these assumptions are true. Here is how it works:

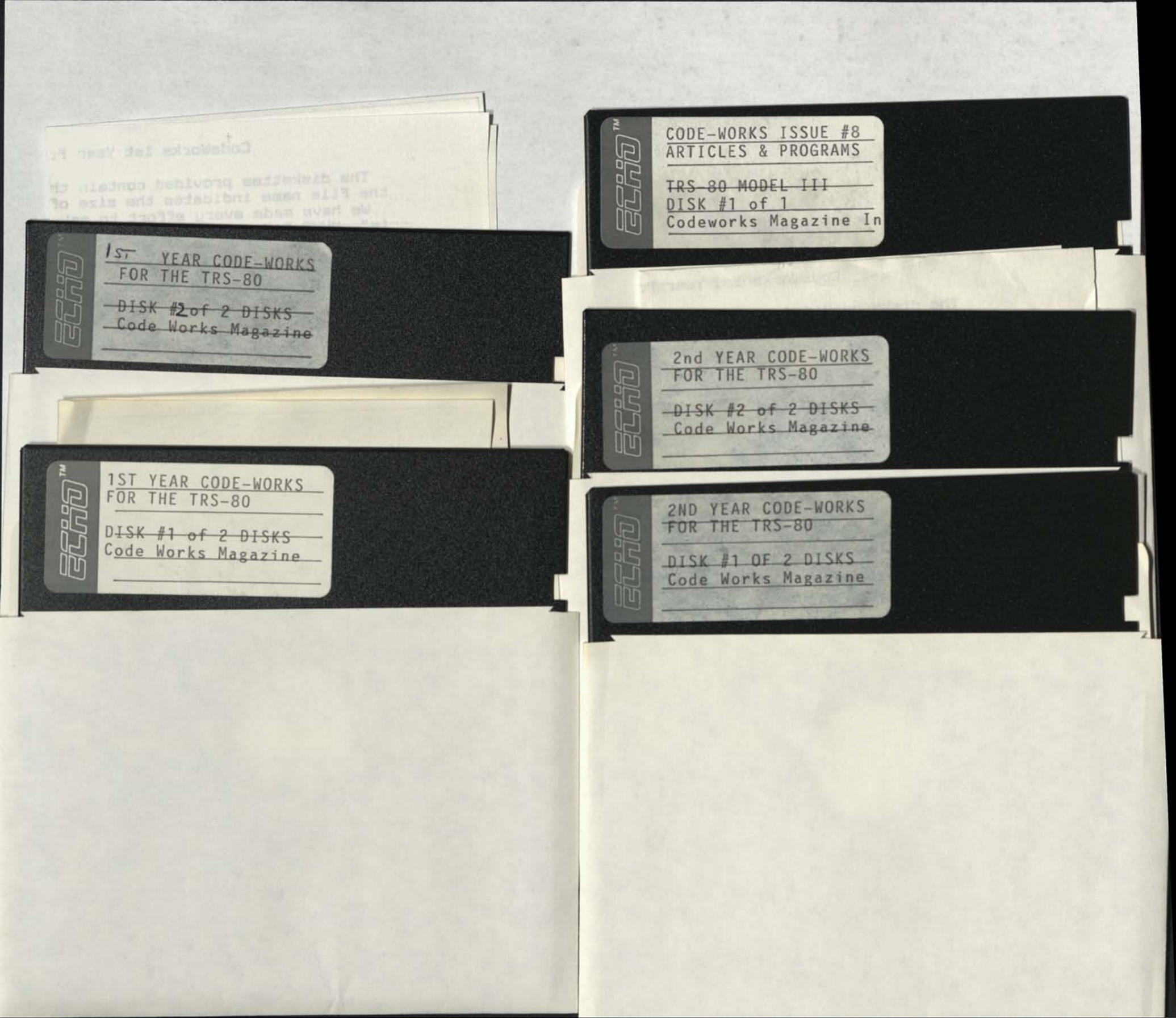
When you dial our number (206) 475-2356 and we connect, the first message you should receive is "Press RETURN for speed check". At this point, press RETURN once (or twice, if nothing happens the first time), and you should see the message "Speed checked at XXX baud", where XXX is either 300 or 1200. The next thing we send is the "Please login" and "User Name:" prompt. At this point, you should respond with your ***last name*** followed immediately with your ***subscriber number***. Your name can be upper case, lower case or mixed. Follow your name immediately (without spaces) with your subscriber number. Your subscriber number is on your mailing label near the upper right. It may be a number from 1 digit to 5 digits long. If it happens to be followed by a slash on your label, ignore the slash when you enter your number on the download.

Our computer has your name and number stored in it. When you enter your name and number, a comparison check is made to see that you are, indeed, a subscriber. If this is the very first time you have called our system, you will now be asked to assign yourself a password. The password *must* be at least six characters long, and may be alphanumeric. You will be asked to enter your password twice for verification. Do not forget what password you used. We do not know what it is because the system encrypts it and we cannot decipher it. Do not use spaces inside your password.

You will now see the Message of the Day and the menu of options. If the system cannot determine that you are a subscriber with a valid number and password, it will unceremoniously dump you into the DEMO side of the download system (which is a small sampler for non-subscribers.) ■

CodeWorks
3838 South Warner Street
Tacoma, Washington 98409

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


1ST YEAR CODE-WORKS
FOR THE TRS-80
DISK #2 of 2 DISKS
Code Works Magazine
1ST YEAR CODE-WORKS
FOR THE TRS-80
DISK #1 of 2 DISKS
Code Works Magazine
CODE-WORKS ISSUE #8
ARTICLES & PROGRAMS
TRS-80 MODEL III
DISK #1 of 1
Codeworks Magazine In
2nd YEAR CODE-WORKS
FOR THE TRS-80
DISK #2 of 2 DISKS
Code Works Magazine
2ND YEAR CODE-WORKS
FOR THE TRS-80
DISK #1 OF 2 DISKS
Code Works Magazine