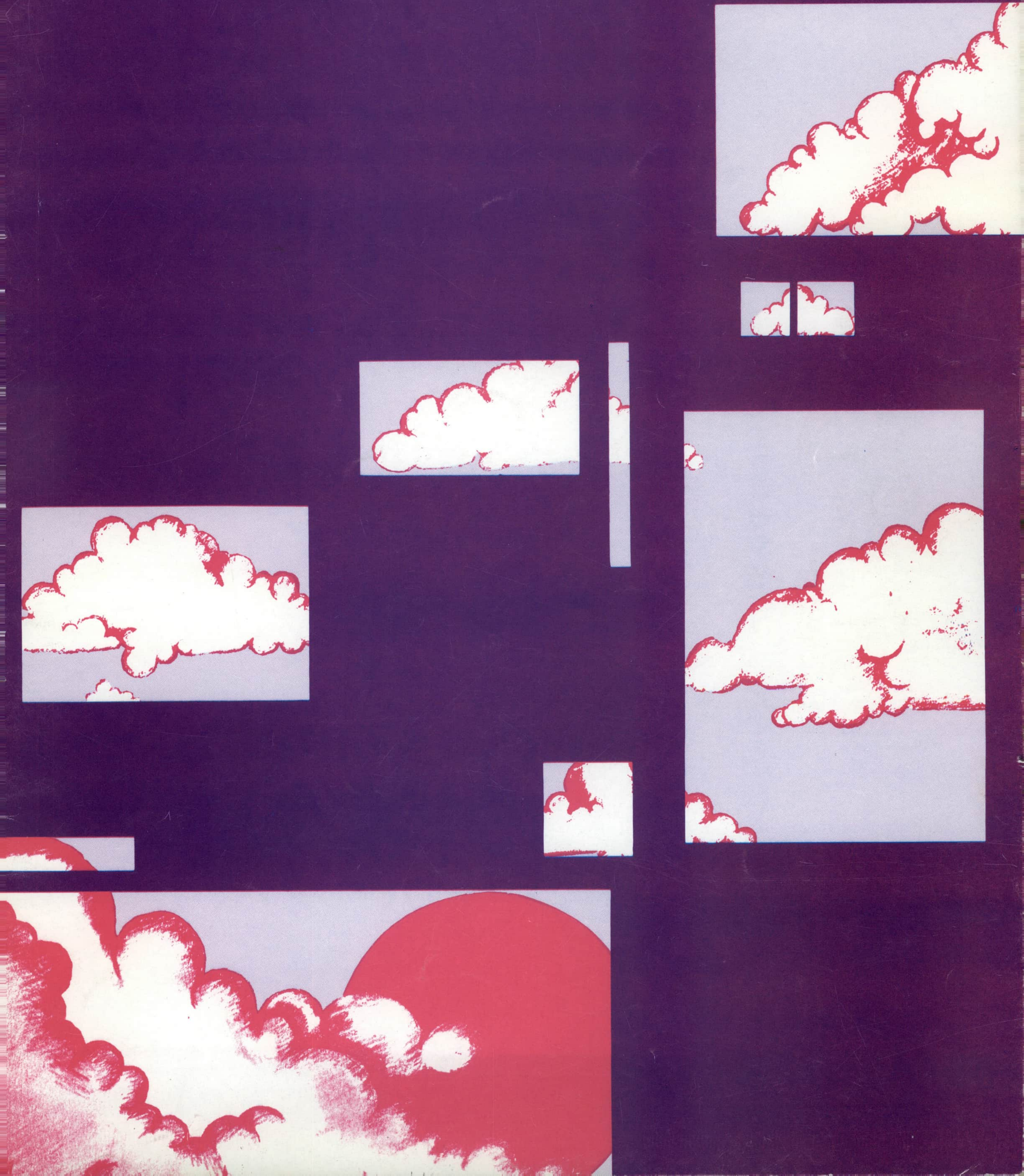
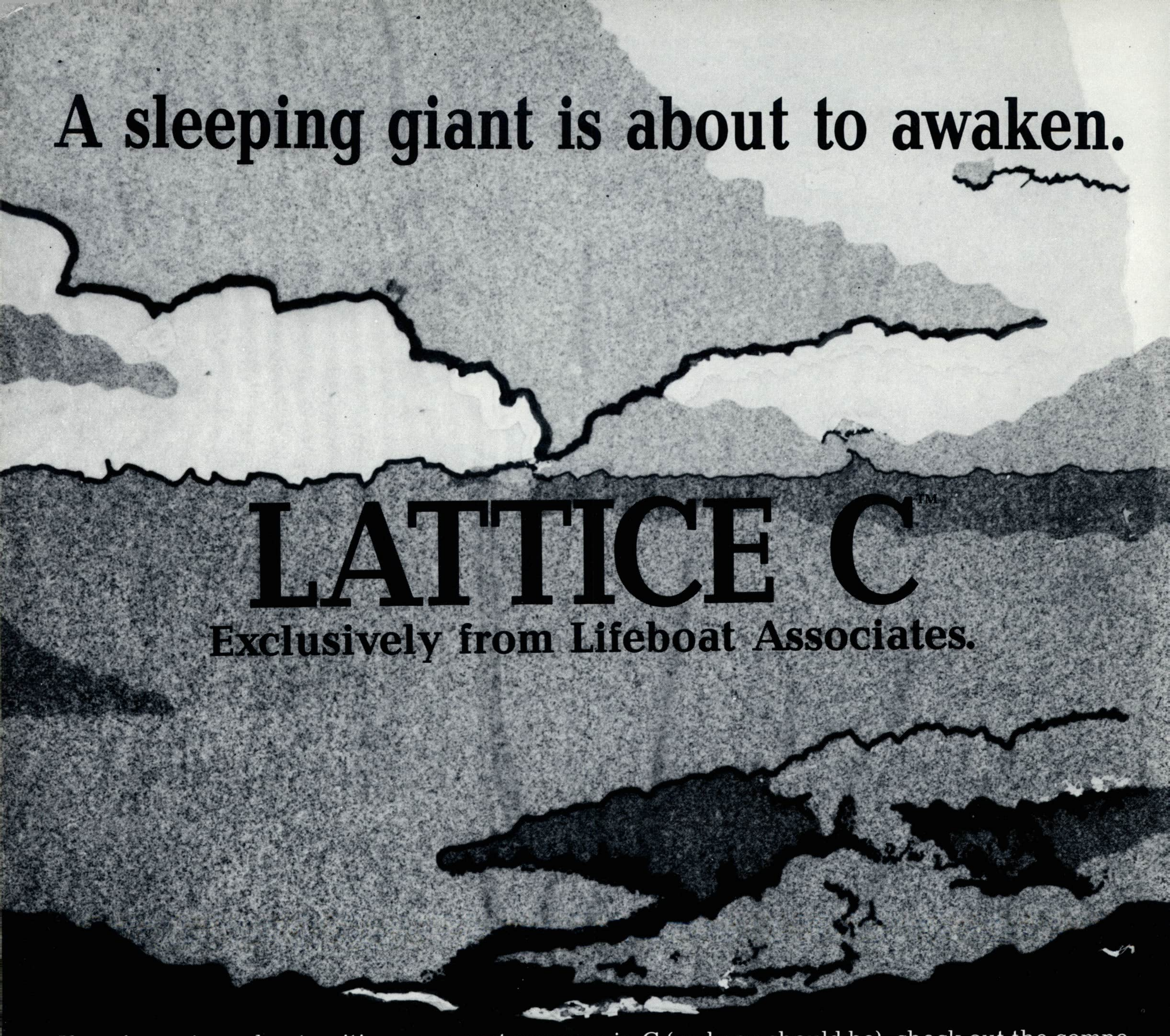
LIFELINES™

The Software Magazine™

$3.00 May 1983 Volume III, No. 12 (ISSN 0279-2575, LISPS 597-830)




A sleeping giant is about to awaken.
LATTICE C™
Exclusively from Lifeboat Associates.

If you're serious about writing your next program in C (and you should be), check out the competition, then take a good look at Lattice C. We think you'll come to the same conclusion we did: Lattice C is the finest 16-bit C compiler on the market today.

Completeness...
Lattice C is a full implementation of the C language as defined by Kernighan and Ritchie (the standard in the C world) including all standard keywords, functions, and commands.

Performance...
Lattice C is the fastest sixteen -bit C compiler you can buy. Fast in execution, and fast in compiling. Lattice C produces compact code, with many load modules as small as 5,000 bytes.

Industry Acceptance...
Major manufacturers and software developers already recognize the strength of Lattice C. New products created with Lattice C include spreadsheets, word processors, and data base managers.

A Total Development System...
The serious software developer will also want **C-Food Smorgasbord™**, a complete C toolkit including BCD decimal arithmetic package, screen and i/o utilities.

Lattice C runs on a wide variety of computers including the IBM®-PC, Wang PC™, TI Personal Computer™, DEC Rainbow™, Victor 9000™, and other compatible 16-bit microcomputers operating under PC-DOS®, MS™-DOS, CP/M-86™, Concurrent CP/M™, SB-86™ and the Intel™ UDI system. Price: **Lattice C** $500 **C-Food Smorgasbord** $150

Lifeboat Associates
1651 Third Avenue, NY, NY 10028 (212) 860-0300
TWX: 710-581-2524 (LBSOFT NYK) Telex: 640693 (LBSOFT NYK)

Lifelines
The Software Magazine

Publisher: Edward H. Currie
Editor in Chief: Susan Sawyer
Production Manager: Kate Gartner
Typographers: Paul Blockhaus, Rosalee Feibush
Printing Consultant: Sid Robkoff/E&S Graphics
New Versions Editor: Lee Ramos
Technical Editor: Al Bloch
Advertising Manager: Carolann Abrams
Cover: Kate Gartner

Lifelines (ISSN 0279-2575,USPS 597-830) is published monthly at a subscription price of $24 for twelve issues, when destined for the U.S. Canad, or Mexico, $50 when destined for any other country. Second-class postage paid at New York, New York, and other locations. POSTMASTER, please send changes of address to Lifelines Publishing Corporation, 1651 Third Avenue, New York, NY 10028.

dBASE II™ made easy!

QUICKCODE™

The dBASE II Program Generator

Now dBASE II is made easy with Quickcode by Fox & Geller. QUICKCODE is a program generator, a computer program which writes computer programs.

FAST AND SIMPLE

With QUICKCODE you can generate a customer database in 5 minutes. Its that fast. All you have to do is draw your data entry form on the screen. It's that simple!

NO PROGRAMMING REQUIRED

QUICKCODE writes concise programs to set up and maintain any type of database. And the wide range of programs cover everything from printing mailing labels and form letters, to programs that let you select records based on your own requirements. There are even four new data types that are not available with dBASE II alone.

YOUR CONTROL

And since you work directly with your information at your own speed and your own style, you maintain complete control. Telling your computer what to do has never been so easy.

QUICKCODE, by Fox & Geller. Absolutely the most powerful program generator you've ever seen. Definitely the easiest to use.

Ask your dealer for more information on QUICKCODE and all the other exciting new products from Fox & Geller.

FOX&GELLER

Fox & Geller, Inc. Dept. LIF 001 **604 Market Street Elmwood Park, N.J. 07407 (201) 794-8883**

Feature

Edix, A Database Manager

by Ron Watson

Among those who work with computers every day, the one subject guaranteed to start an argument faster than politics, religion or structured programming is text editors. Everyone is sure that the one they prefer is absolutely the best possible and that all others are entirely despicable, beyond contempt. The inflexibility of their opinions would leave a Middle Eastern diplomat talking to himself. One voices an opinion on any text editor at his or her own peril.

In the course of any month, I find myself working with at least a half dozen text editors, on main frames, minis and micros. I have customers with all sizes of hardware, and every one has a text editor. Shielded by this armor of extraordinary experience, I feel qualified to offer an opinion on this very emotional subject. I have observed that each has its own strengths and weaknesses. Having worked on so many, it is apparent that many are designed, not from an original idea, but in an attempt, I suppose, to do the job better, or to bring the capabilities of an existing editor from one machine to another. EDIX, by Emerging Technology, the one being reviewed here, resembles the DEC EDIT product, and was probably written by people familiar with it.

The first prerequisite to a fair and reasoned judgement is to become familiar enough with the product so that the differences between it and those to which one is more accustomed do not cloud the opinion. Any editor is entirely incidental to the job being done, and to be really useful, it must not intrude upon the thought process any more than absolutely necessary. Having to look up every command in the reference manual can be a terrible distraction, so an opinion offered before considerable familiarity is achieved is really not fair.

I have had the product for over a month now and during that time used it to write a magazine article, a BASIC program of about 1500 lines, and ten pages of documentation for the program. I also used it for all the little jobs one has for a text editor. During this time, I continued to use my old favorites on other machines, of course, but whenever possible I used EDIX on the IBM PC.

First, let's go over the documentation and training support provided with the software. The manual is the same size as most of the software available for the IBM PC, except they used a padded binder with no slipcase. There are only 62 pages, which is good, but there are few charts or diagrams and the type style is a bit small, which is not so good. The manual is strictly for reference, however, as is explained in the introduction. A programmed tutorial is provided, so no training section is required in the manual.

What a tutorial! Every software package should come with a facility as well done as this one. Sit down at the machine, run the program in its tutorial mode, and in less than an hour you've learned enough to be proficient in all the important aspects of its use. And if you forget how to invoke a function, just hit PF7 and get as much or as little help as you might require. This program wins the "user friendly" derby hands down.

They have made full use of the PC's alt-key, function key and cursor control capabilities. The function key assignments may be changed by the user, but I found no need for this as the standard assignments were fine. Besides, I didn't want to bother with editing the help screen file to match up with my changes.

The help function, tutorial, and key utilization make this an excellent tool for the occasional user, but what about us old salts, the ones who become so familiar with an editor that our hands seem to know what to do without being told?

Every editor I have used seems to express the intentions of the designers through its strengths and weaknesses. Since many of the designers are also programmers, most editors are at their best when preparing or editing a program. The features that make the editor unique usually address some real or imagined deficiency the designer perceived to exist in the products available before he decided that he could do a better job than his predecessors. Any project is subject to certain limitations of resources, however, and a balance must be found between the features to be provided and the resources available. Features that are clumsy or difficult to understand will not be used.

Going through a manual for any text editor, one can find any number of features and capabilities costing untold thousands of dollars to develop that are not known or even needed by one user in ten. Like everyone who ever developed a program for general use, the people at Emerging Technology had to decide what should be included and what should not.

They came to some interesting conclusions, and it's not always obvious why.

Probably the most striking feature of this editor is its split-screen capability. With a single key stroke, the screen can be divided horizontally and/or vertically in half, into as many as four parts or windows, and each window may be assigned to one of twelve internal buffers. Each buffer may be associated with its own external file, making it possible, on the surface, to deal with a great deal of data in one edit session.

This is quite a set of capabilities, but in a month of serious work, I found very little use for the split screen, and never with more than two windows. Two buffers were handy on occasion, three or four might be useful, but twelve is extravagant. A buffer may only be assigned to one window at a

(continued on next page)

time, so the split screen cannot be used to see two parts of the same file at once. It might have been more useful without this limitation. I have worked with other editors that allow split screens and multiple buffers, usually on mainframe systems, and have observed very few users actually making much use of either. Most users, even the more technically sophisticated, seem to find keeping track of several buffers somewhat confusing and susceptible to error, and the utility of a split screen is hampered by the necessarily small size of the window. I think Emerging Technology has misjudged the need for these things. They do make a great demonstration, though, and if there are no serious deficiencies in more important capabilities, then there is no harm done.

There seems to be a full complement of commands available. Files may be read and written, strings may be found and changed, lines may be inserted and deleted....What about this line deletion? There is no block delete command? One at a time? Well, that's not so bad. There is a block move that allows text to be moved into another buffer that could be used as a block delete, and there would always be a spare copy of the deleted text around...so long as there is plenty of memory available.

Which brings up a slightly curious situation. The maximum file size that can be edited on a 64k machine appears to be about 5k, while the maximum on a 128k machine is about 40k. The 40k limit is mentioned in the manual, but it's not clear why adding 64k to the available main memory only increases the buffer space by 35k. All those windows and buffers must cost something. The program will make use of as much as 128k, which is as good or better than most programs available for the PC today, but there is no way to find out if you are nearing the end of available buffer space. The alt-P command will show what files and windows are associated with which buffers, but does not show space used or space available. The first indication one gets is when the screen begins to roll furiously and a message appears informing one to save his files and terminate the program immediately. This was the only part of the program's function that I would call unprofessional.

Files that are too large to fit into available memory must be broken up to be edited. No provision has been made to manually or automatically read or write partial files. This could be pretty unpleasant on a 64k machine with the 5k limit, but considering the easy availability of larger memory sizes on the IBM PC, it is probably not a serious problem.

Text is entered and changed in the usual manner, with one important exception: the insert function is always on and there is no way to turn it off. This may or may not seem strange to you; the DEC EDIT program works this way. I found it clumsy at first, but determined to give this method every opportunity to redeem itself. After a month of trying, I surrendered. It was just too uncomfortable for me. The extra key strokes required to delete data when an overstrike would have served were a serious distraction, and it was particularly difficult to maintain any column orientation. This may be a particularly personal criticism; no doubt there are plenty of people who prefer entering everything in insert mode. I have surveyed twelve regular terminal users and not found one yet, but I suggest you try it for yourself before deciding.

There is no way to define a right margin for the screen display. So, each text record always uses one and only one line on the screen. Data beyond the visible right margin of the screen is brought into view by scrolling the window to the right with the cursor key, the same way Visicalc displays a wide spread sheet. This seemed alright at first, but when I tried to edit a BASIC program containing spaces on long lines designed to make them readable on the BASIC editor's screen, I found it nearly impossible to keep the needed spaces in the appropriate places. Another discomfort is caused if the file contains a few long lines scattered among a majority of short lines; you can't see the right side of the long lines without scrolling the short lines off the screen. Since there is no way to set a right margin for text entry, you have to watch the screen while you type if you want to keep each line within a certain boundary. This probably makes the program unacceptable for any high-volume text entry environment.

The tab key turned out to be of little use. The tab stops are not changeable and keyed tabs do not expand into spaces on the screen. The tabs are kept as tab characters in the buffer until the file is written out, when they are replaced by the appropriate number of spaces. The area preceding the tab on the screen is not accessible, but by deleting the tab, one deletes all the space it occupies. The back tab key is not usable. While the overall function of the tabs is not too bad, not being able to set the tab stops meant they had little use for me.

The block move and copy work quite well. The alt-k command is used to mark the beginning and end of the text to be processed and then an alt-m or alt-c command is given to move or copy the marked block to the position indicated by the cursor. Data may be moved or copied from any buffer into the current buffer, but only one buffer may contain a marked block at any time. A third alt-k command is used to clear the existing marks, but the cursor must be in the buffer containing the marked block before this can be done.

The global search and translate (their word for replace) commands are quite comprehensive, though a method to search without regard to shift case was not included. The search command uses an argument string that may contain several wild-card characters and two search arguments are actually available at once. (This is what happened to the insert function key: it is used to continue a search in progress or to indicate approval for a replacement to be made.) A search proceeds from the current cursor position and continues to the end of the buffer and around again as long as you keep hitting the insert key or until a complete pass is made without finding anything. The current search string is retained until replaced with another, and the previous string is always available. The search stops each time the required string is found, and all the normal editing functions are available. If the translate command was used, you have the option of replacing the matched text, skipping to the next match or exiting the command. There are many wild-card characters available for the search, including special codes that will match only the beginning or end of a line. This is all well done and well documented in the manual, but will probably not be used by many.

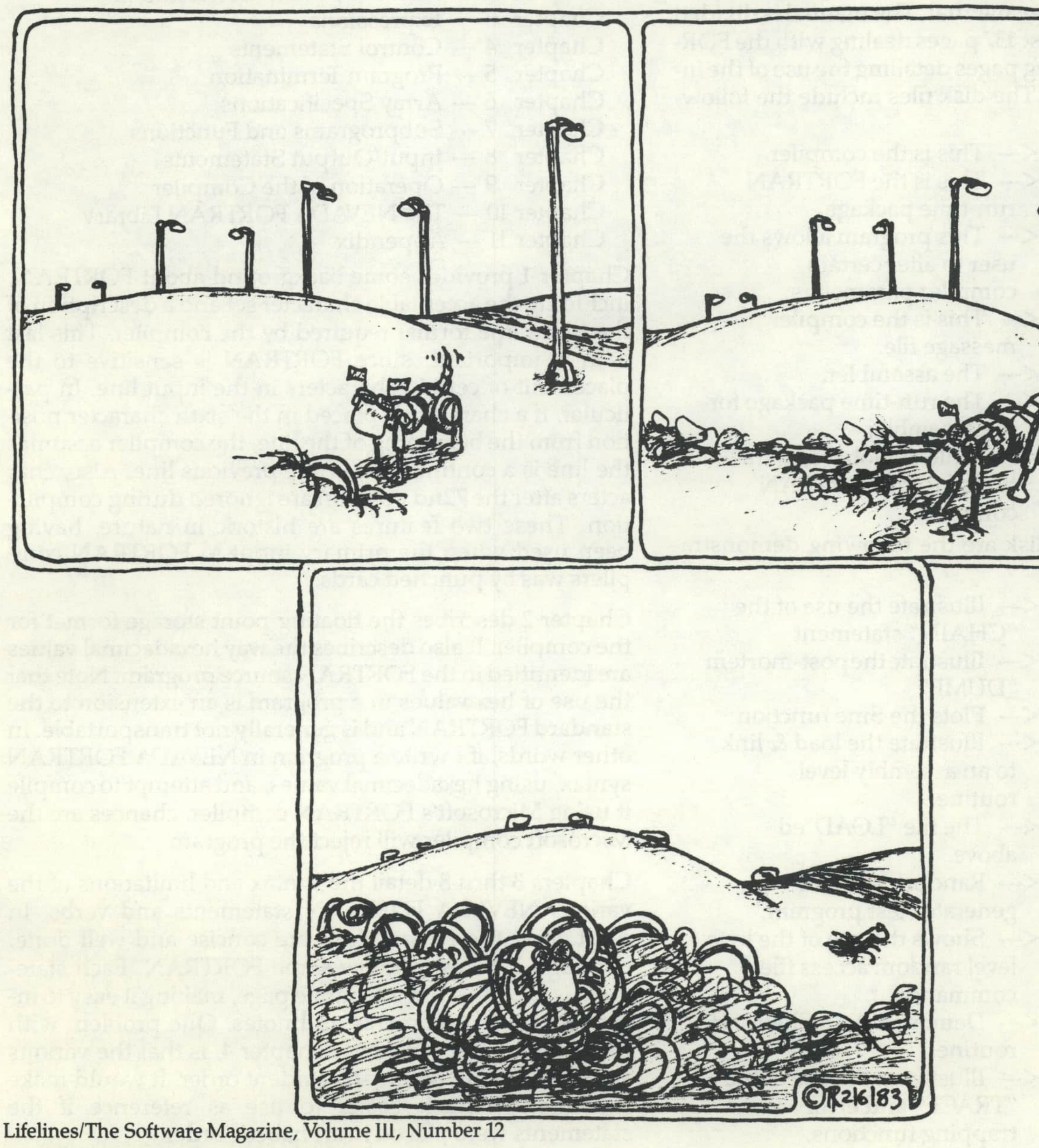
The I/O commands work by assigning a file name to a buffer so that all subsequent read (alt-r) or write (alt-w) commands from the buffer refer to that file. If the program is invoked with a file name parameter, that file name is automatically assigned to buffer number one. There is no single command to exit the program and write the output file: a separate alt-w command must be given from each buffer that has been changed before the alt-x command is given to exit the program. If you attempt to issue the alt-x command while there are unwritten changes in any buffer, a warning is given and you must respond to it before the program will end. Incidentally, the response required for most questions is the usual (y/n) for yes or no, but the program allows only lower case answers.

The program works quite fast in most circumstances. Diskette reads and writes were done quickly, and search times were quite respectable. The screen update speed was more than adequate in most cases.

It is important to note that during the entire time I used the program, not one program error was encountered. That includes everything working as specified in the documentation. While I didn't actually try everything described in the manual, I did test the limits of the program and made a real effort to confuse and confound it with unusual command sequences. Rarely have I encountered a new application on any computer that was as bug-free as this one seems to be. Emerging Technology certainly deserves some recognition for high quality product testing.

CONCLUSION

This is a well produced product, quickly learned and easy to use. There is a certain lack of capability to tailor the program to one's personal preference, and to be really successful in a market as competitive as this, Emerging will have to provide some customization options. If the current configuration seems to fit your needs, this program should be seriously considered, particularly if you are a new user, or one who uses an editor only occasionally. An oldtimer may find the shortcomings intolerable, but then an oldtimer would probably find any editor other than the one he/she is now using intolerable. ■



Feature

A Review of Nevada Fortran

by Thomas Hill

Introduction

Recently (January '83) Ellis Computing announced the availability of their NEVADA series of languages (FORTRAN, COBOL, and PILOT) for the unheard of price of $29.95 each. Never missing a chance at a bargain, I immediately purchased copies of all three. This particular article will address the NEVADA FORTRAN. In future reviews I will be looking at Microsoft's FORTRAN and the FORTRAN compiler from Supersoft. The final installment will provide a comparative look at all three compilers.

What's In The Box?

The NEVADA FORTRAN arrives on a single density 8" CP/M disk with a 174 page manual. The manual is divided into two sections, the first 137 pages dealing with the FORTRAN and the remaining pages detailing the use of the included 8080 assembler. The disk files include the following:

FORT.COM	<--- This is the compiler.
FRUN.COM	<--- This is the FORTRAN run-time package.
CONFIG.COM	<--- This program allows the user to alter certain compiler parameters.
ERRORS	<--- This is the compiler message file.
ASSM.COM	<--- The assembler.
RUNA.COM	<--- The run-time package for the assembler.
FDEFS.ASM	<--- A file of global declarations for the FORTRAN compiler.

Also included on the disk are the following demonstration programs:

CHAIN.FOR	<--- Illustrate the use of the "CHAIN" statement
DUMP.FOR	<--- Illustrate the post-mortem "DUMP"
GRAPH.FOR	<--- Plots the Sine function
LOAD.FOR	<--- Illustrate the load & link to an assembly level routine.
LD.ASM	<--- The file "LOAD"ed above.
RAND.FOR	<--- Random number generator test program.
SEEK.FOR	<--- Shows the use of the byte level random access file commands.
SORT.FOR	<--- Demonstration Shell sort routine.
TRACE.FOR	<--- Illustrates the use of the "TRACE" and error trapping functions.

The Documentation

The manual accompanying the NEVADA FORTRAN is well bound and punched for a three-ring notebook. The material assumes some knowledge of FORTRAN IV, although a person with a good background knowledge of programming will probably have little difficulty making sense of the material. The lack of an index is a little annoying, but the Table of Contents is sufficiently detailed that the user should not have problems finding what is needed.

The FORTRAN section of the manual is divided into the following chapters:

Chapter 1 --- The FORTRAN Language
Chapter 2 --- Number System Conventions
Chapter 3 --- Expressions
Chapter 4 --- Control Statements
Chapter 5 --- Program Termination
Chapter 6 --- Array Specifications
Chapter 7 --- Subprograms and Functions
Chapter 8 --- Input/Output Statements
Chapter 9 --- Operation of the Compiler
Chapter 10 --- The NEVADA FORTRAN Library
Chapter 11 --- Appendix

Chapter 1 provides some background about FORTRAN, including the acceptable character set and a description of the input line format required by the compiler. This last item is important, since FORTRAN is sensitive to the placement of certain characters in the input line. In particular, if a character is placed in the sixth character position from the beginning of the line, the compiler assumes the line is a continuation of the previous line. Also, char acters after the 72nd position are ignored during compilation. These two features are historic in nature, having been used when the primary input to FORTRAN compilers was by punched cards.

Chapter 2 describes the floating point storage format for the compiler. It also describes the way hexadecimal values are identified in the FORTRAN source program. Note that the use of hex values in a program is an extension to the standard FORTRAN and is generally not transportable. In other words, if I write a program in NEVADA FORTRAN syntax, using hexadecimal values, and attempt to compile it using Microsoft's FORTRAN compiler, chances are the Microsoft compiler will reject the program.

Chapters 3 thru 8 detail the syntax and limitations of the various NEVADA FORTRAN statements and verbs. In most cases the explanations are concise and well done, although it helps to know some FORTRAN. Each statement is explained on a separate page, making it easy to insert your own comments and notes. One problem with these chapters, particularly chapter 4, is that the various explanations are not in any evident order. It would make the manual much easier to use as reference if the statements were placed in alphabetic order.

Chapter 9 describes the use of the compiler. The compiler is invoked by:

```
FORT <filename>.LAO $<options>
```

where FORT is the compiler, <filename> is the name of a FORTRAN source program, "L" indicates the destination of the compiler produced listing,"A" indicates the destination of the intermediate .ASM file, and "O" indicates the destination of the final .OBJ file. The <option> list following the dollar sign is used to instruct the compiler to NOT produce an assembly file, paginate the listing file, blank-pad the FORTRAN source lines, and override default values for certain internal compiler parameters. Execution of the compiled program is accomplished by the command:

```
FRUN <filename>
```

where FRUN is the FORTRAN run-time package. The run-time package occupies memory from 0100H thru 3FFFH. It loads the FORTRAN compiled program into memory at 4000H for execution. Note that the final compiled program IS NOT an executable .COM file.

Chapter 10 describes the NEVADA FORTRAN subroutine library. It includes routines to open, close, rename, and delete CP/M files, perform direct console input and console status checks, input and output to specified ports, chain to other programs, load modules assembled with the assembler, and perform a number of bit oriented functions upon FORTRAN variables.

Finally, Chapter 11 describes the differences and non-standard extensions which make NEVADA FORTRAN different from the ANSI 1966 standard FORTRAN. Many of these extensions have to do with the CP/M environment, such as raw port input and output and hexadecimal representation of values. Chapter 11 also presents a list of the built-in functions. Most of the standard FORTRAN functions are represented, including the trigonometric functions, the log functions, and the maximum and minimum functions. Non-standard functions include a random number generator and a bit oriented function which allows setting, resetting, or testing of individual bits of a variable.

One particularly interesting feature is the ability to include in-line 8080 assembly code in the FORTRAN program. This code is passed to the assembler during the compile operation and appears in the intermediate assembly source file.

Using The Compiler

The program in Listing 1 was used as an input file. This program is part of a larger set of FORTRAN subroutines designed to perform a statistical operation called "factor analysis." I won't attempt to explain what an "eigenvalue" is, nor how it is used, since that would entail a rather thick book. Suffice it to say that the calculation of eigenvalues is a non-trivial application of the FORTRAN language and has been studied by many people. Four test arrays were submitted to the program and the results were checked for accuracy against known values. The time of compilation and execution of the program was also measured. Compilation took one minute, ten seconds, while the assembly step took 58 seconds. This compilation was performed upon a Z80 based system, using a 4MHz clock and double density/double sided 8″ floppies controlled by a DMA based disk controller. Execution time of the program with a five-by-five test array was 15 seconds, a respectable showing. Answers from the program in listing 1 are correct to 5 decimal digits when compared to the known true answers.

The compilation of the program was performed using the command:

```
FORT TSTEIGEN
```

This form of the compiler command sends the compiled listing to the default drive, along with the intermediate assembly file and the final .OBJ file. The compiler produces as output an .ASM text file. By using the proper control options when invoking the compiler, this file may be left intact for perusal and possible hand optimization. In a normal compilation the compiler automatically invokes the assembler after a successful compilation, passing the name of the file to the assembler. When the assembler completes its task, the assembly source file is erased, unless the programmer explicitly instructs the compiler/assembler to leave the file intact.

One annoying quirk of the compiler is its handling of "dangling" blank lines in the FORTRAN source file. If blank lines are present after the final valid line of FORTRAN code, the compiler assumes another segment of the FORTRAN program is upcoming. When the end of the file is reached, the compiler will abort with a fatal error, complaining of not finding an END statement. Since it is very easy to build up blank lines at the end of text files, it is necessary to explicitly erase any that have accumulated before submitting the source program to the compiler.

To test the compatibility with FORTRAN programs created using a 'standard' compiler, several programs and subroutines were extracted from volume 3 of the "Collected Algorithms of the ACM" series. Listings will not be presented here, since the subject material involved esoteric mathematics and were of interest only to the author (and others involved in computing such things as Bessel functions, Legendre polynomials, and roots of polynomials of degree 10 and higher.) If you wish listings, send me a request via Lifelines and I'll mail them to you.

To resume the train of thought, the test programs were submitted to the NEVADA FORTRAN unmodified from the original source code, as published in the volume mentioned above. Of the four programs tried, three compiled without errors, and no modifications were needed to produce the results listed with the published algorithm. The fourth required slight modifications in several areas:

1) The compiler had to be instructed to reserve space for 150 symbols during compilation (the default is 50). The reason for this apparently had to do with the large number of variables declared as COMMON.

2) Certain variables describing the floating-point representation of values had to be altered to prevent over- and under-flows.

3) The NEVADA FORTRAN required re-ordering of the respective declarations of COMMON blocks versus the declaration of array variables. This may just be idiosyncrasies of the respective compilers involved.

(continued on next page)

Compiler Restrictions

The NEVADA FORTRAN does not support the FORTRAN variable types of COMPLEX and DOUBLE PRECISION. The lack of COMPLEX data types is not restrictive, since these can be supported via other array constructs, if needed. The double precision data type, however, limits the compiler's usefulness in some areas. This becomes apparent when some program requires the definition of the parameter commonly defined as "ETA", which is the smallest value which can be added to one (1.0) and result in a value larger than one. Using single precision variables the value of ETA can be no smaller than 1.0E-6. Using double precision, ETA may be as small as 1.0E-12.
Also lacking is the "EQUIVALENCE" statement, which is used to provide a form of variable overlay. In most FORTRAN programs this will not be missed. Format specifications "D" and "P" are not implemented: The "D" format spec is intended for double precision variables and the "P" spec is used to perform automatic scaling upon input or output values. With proper programming this should not be needed.

Extensions To Standard FORTRAN

One of the first extensions to standard FORTRAN that one notices is the huge range of integers and real variables. In NEVADA FORTRAN integer values may range from 99999999 to -99999999, while real values may range from 1.0E-127 to 1.0E+126. This is accomplished by storing numeric values internally in six byte packed BCD format. Note that the real values are still limited to 8 digits of precision, only the dynamic range has been increased. Other extensions include random access files at the byte level, free-form input and output statements, multiple RETURNs from subroutines, in-line assembly code, complete runtime control over error trapping, and the extension of the IF statement to include a "THEN-ELSE" structure.

One nice feature (which I am unable to take advantage of, unfortunately) is the ability to configure the compiler to use a North Star Floating Point board. This should greatly increase the number crunching speed of the compiler, although the manual indicates that use of the N* board reduces the dynamic range of reals by half.

The Assembler

The assembler which is included with the NEVADA FORTRAN package is intended primarily for use with the FORTRAN (or COBOL). It is automatically invoked by the compiler during program compilation. The programmer may use it to create assembly level modules suitable for incorporation into a FORTRAN program by the proper ORG statements. One result of this use of the assembler is that it does not produce CP/M type .COM files. A file of type .OBJ (this file type is apparently used by all NEVADA compiler products) is produced by the assembler. In order to execute this "assembled" program, use the command:

```
RUNA <filename>[.ZLC]
```

where the <filename> is the primary name of the assembled file and the [ZLC] are defined as:

Z --	Zero memory before program execution.
L --	Load the program but do not execute, return control to CP/M.
C --	Create a standard .COM file and return to CP/M. The program is saved as a .COM file, but is not executed.

The assembler is a relatively standard package, using 8080 mnemonics. Input file format is somewhat restrictive, particularly to me. The assembler requires labels to appear in the first column of the input line and comment lines MUST appear with the semicolon in the first column. This is a direct contradiction to my method of program writing, which makes extensive use of the tab character as the first character of a line. The usual repertoire of pseudo-ops are available to the programmer, including the standard "DS", "DB", "DW", "EQU", and "ORG". Also present are the following:

ASC --	Define ASCII string. The first non-blank character is considered the string delimiter, and the string will end at the next occurence of the delimiter, or a carriage return.
ASCZ --	As above, but a byte of 00H will be appended to the end of the string.
DDB --	Define double byte. Similar to the DW pseudo but the bytes will be stored in the reverse order: high byte first, followed by low byte.
COPY --	Read source code from a separate disk file. The assembler will use the code from this file until it is exhausted, then return to the main file.
IF <exp>	Standard IF conditional assembly switch. No ELSE construct is supported.

Also included are listing control flags for pagination control and page titles.

Arithmetic expressions are supported for the four basic operators. Operator precedence is not supported, neither is parenthetical control.

Conclusions

The NEVADA FORTRAN package is an excellent product, conforming sufficiently close to standard FORTRAN to make possible the use of much existing software with a minimum of modifications. The large dynamic range of the numeric variables and the use of BCD arithmetic aids in reducing round-off errors during computations, and makes integer manipulations possible for values far beyond other FORTRAN compilers. The extensions are, in general, well thought out and implemented in a logical fashion. The manual is well written and informative, provided some prior knowledge of FORTRAN exists. The lack of double precision is somewhat limiting, although ways exist to work around this. Execution speed is acceptable, and final file sizes are very compact. Part of this can be attributed to the use of a separate run-time package, but even considering this, the size of the NEVADA FORTRAN program when compared to other compilers is significantly smaller.

The runtime error recovery and post-mortem variable dump verb provide excellent debugging support, as does the free format output verb. Compilation speed is slow, since after the FORTRAN compiler is finished, the assembler must do its work, but this can be borne. This compile pause gives dedicated programmers time to prepare a cup of tea (or coffee, if that's your poison).

The assembler is acceptable, once one becomes accustomed to the input format and the fact that you must use the RUNA command to execute your program, at least the first time. The assembler was turned loose on the SETIO program previously described, and passed with flying colors after leading tabs were removed from the input file.

Final Words

In general, I would consider this to be an excellent FORTRAN, especially for the purchase price. I would like to take this opportunity to thank Ellis Computing for taking the route of reducing prices of mature software. I think that this would encourge the use and dispersion of programs written in their languages, and the temptation to provide copies to friends (freely granted by Ellis Computing) is greatly offset by the fact that the final cost of photocopying the manual will come perilously close to the $29.95 a real manual and disk costs.

```
OPTIONS X,B,E
C
C     TEST DRIVER FOR THE EIGEN SUBROUTINE.
C     THIS VERSION IMPLEMENTED FOR NEVADA
C             FORTRAN.
C
          REAL INP(10,10),EIGVAL(10),B(10),C(10),
      X   P(10),Q(10),R(10),W(10),Y(12),IN(10)
C
C         HAVE TO DECLARE THE WORK ARRAYS
C         AS COMMON SINCE THEY ARE NOT
C         PASSED AS ARGUMENTS IN THE
C         SUBROUTINE CALL.
C
          COMMON /ARRAYS/ B,C,P,Q,R,W,Y,IN
C
          INTEGER EVAL,RC,XC
C
          CALL OPEN(4,'ARRAY.DAT')
C
          READ(4,700) RC,XC
700       FORMAT(2(I2))
          DO 716 I = 1,RC
          READ(4,705,END=95,ERR=720)
          (INP(I,J),J=1,RC)
705       FORMAT(F10.2)
716       CONTINUE
          GO TO 95
C
720       STOP 'READ FILE ERROR'
C
C         LOOK FOR ALL THE EIGENVALUES
C
95        EVAL = RC
C
C         INITIALIZE THE EIGENVALUE ARRAY
C
          DO 100 I = 1,10
100       EIGVAL(I) = 0
C
C         PRINT THE INPUT ARRAY
C
          WRITE(1,300)
300       FORMAT('INPUT ARRAY IS:')
          DO 112 I = 1,RC
          WRITE(1,310) (INP(I,J),J=1,RC)
310       FORMAT(F10.2)
112       CONTINUE
C
C         CALL THE EIGEN SUBROUTINE
C
          CALL EIGEN(INP,EIGVAL,RC,EVAL)
C
C         PRINT THE ANSWERS
C
          WRITE(1,200)
200       FORMAT('COMPUTED EIGEN VALUES
            ARE:')
          WRITE(1,202) (EIGVAL(I),I=1,RC)
202       FORMAT(F10.6)
C
          CALL EXIT
          END
OPTIONS   X,B,E
          SUBROUTINE EIGEN(A,E,N,NEV)
C
C         THIS SUBROUTINE COMPUTES THE
C         EIGENVALUES FOR THE N X N INPUT
C         MATRIX A().
C         THE CALLING FORMAT IS:
C
C         CALL EIGEN(A,E,N,NEV)
C
C         WHERE       A IS THE N X N INPUT
C                        MATRIX,
C                     E IS THE (NEV) MATRIX
C                        WHICH RETURNS THE
C                        COMPUTED
C                        EIGENVALUES,
C                     N IS THE DIMENSION OF
C                        THE INPUT MATRIX,
C                   NEV IS THE NUMBER OF
C                        EIGENVALUES TO
C                        COMPUTE,
C
C
          REAL A(10,10),E(10)
C
C         DECLARE COMMON ARRAYS TO
C         ACCESS DEFINED ARRAYS FROM MAIN
C         ROUTINE
C
```

(continued on next page)

```
      COMMON /ARRAYS/
     1  B(10),C(10),P(10),Q(10),R(10),W(10),Y(12),
     2  IN(10)
C
      INTEGER AG,N,NEV,NVEC
      REAL KAP,NORM,LAMBDA,L,MULT
      LOGICAL FIRST,IN
      NM1 = N - 1
      NM2 = N - 2
C
C     BEGIN THE REDUCTION TO
C     TRIANGULAR FORM. THE ORIGINAL
C     MATRIX IS DESTROYED IN THE
C     REDUCTION. THE DIAGONAL
C     ELEMENTS OF THE TRANSFORMED
C     MATRIX ARE STORED IN THE ARRAY C()
C     AND THE OFF-DIAGONAL ELEMENTS
C     ARE STORED IN B(). THE DIAGONAL
C     ELEMENTS OF THE ORIGINAL MATRIX
C     ARE USED TO STORE THE ALPHA
C     VALUES, WHILE THE SUB-DIAGONAL
C     ELEMENTS HOLD THE VECTORS W.
C
      IF (N.LE.2) GO TO 9
      DO 8 I = 1,NM2
      IP1 = I + 1
      SS = 0.0
      DO 1 J = IP1,N
1     SS = SS + A(J,I)**2
      S = SQRT(SS)
      IF (A(IP1,I).LT.0.0) S = -S
      C(I) = A(I,I)
      B(I) = -S
C
C     IF S = 0 THEN ALPHA MUST BE 0 ALSO
C
      ALPHA = 0.0
      IF (S.EQ.0.0) GO TO 8
      ALPHA = 1.0 / (SS + A(IP1,I)*S)
      T = A(IP1,I) + S
      A(IP1,I) = T
      W(I+1) = T
      IP2 = I + 2
      DO 2 J = IP2,N
2     W(J) = A(J,I)
      DO 4 J = IP1,N
      T = 0.0
      DO 3 K = IP1,N
3     T = T + A(J,K) * W(K)
4     P(J) = T * ALPHA
      KAP = 0.0
      DO 5 K = IP1,N
5     KAP = KAP + W(K) * P(K)
      KAP = 0.5 * KAP * ALPHA
      DO 6 K = IP1,N
6     Q(K) = P(K) - KAP * W(K)
      DO 7 J = IP1,N
C
C     COMPUTE THE NEW A(). BY SYMMETRY
C     ONLY HALF THE ELEMENTS NEED BE
C       DONE.
C
      DO 7 K = J,N
      A(J,K) = A(J,K) - (Q(J) * W(K) + W(J) *
     1  Q(K))
7     A(K,J) = A(J,K)
      A(I,I) = ALPHA
8     CONTINUE
9     C(N-1) = A(N-1,N-1)
      C(N) = A(N,N)
      B(N-1) = A(N-1,N)
C
C     THIS COMPLETES THE REDUCTION TO
C     TRIANGULAR FORM.
C
      WRITE(1,330)
330   FORMAT('TRIANGULAR REDUCTION
     1  COMPLETE.')
      WRITE(1,331)
331   FORMAT('COMPUTED DIAGONAL
     1  ELEMENTS ARE:')
      WRITE(1,332) (C(I),I=1,NEV)
332   FORMAT(4(F10.4))
      WRITE(1,336)
336   FORMAT('COMPUTED OFF-DIAGONAL
     1  ELEMENTS ARE:')
      WRITE(1,332) (B(I),I=1,NEV)
C
C     NOW BEGIN THE EIGENVALUE
C     COMPUTATIONS.
C     FIRST COMPUTE THE NORM OF THE
C     TRIDIAGONAL MATRIX.
C
      NORM = ABS(C(1)) + ABS(B(1))
      DO 10 I = 2,N
      T = ABS(C(I)) + ABS(B(I)) + ABS(B(I-1))
10    NORM = AMAX1(NORM,T)
      DO 11 I = 1,N
11    W(I) = B(I)**2
C
C     SET K AND UPPER AND LOWER
C     ESTIMATES.
C
      K = 1
      U = NORM
      DO 12 I = 1,NEV
12    E(I) = -NORM
C
C     BEGIN MAIN LOOP
C
13    L = E(K)
14    LAMBDA = 0.5 * (L + U)
C
C     THE CONVERGENCE TEST
C     IMPLEMENTED HERE ALLOWS THE
C     COMPUTATIONS TO PROCEED UNTIL
C     THE INTERVAL (L,U) CAN BE MADE NO
C     SMALLER.
C
      IF ((LAMBDA.EQ.L).OR.(LAMBDA.EQ.U))
     1  GO TO 30
C
C     BEGIN COMPUTATIONS OF NUMBER OF
C     SIGN AGREEMENTS, AG.
C     THE ORIGINAL FORTRAN LISTING
C     USED A MACHINE DEPENDENT
C     OVERFLOW TRAP DURING THESE
C     COMPUTATIONS. IN ORDER TO
C     REMOVE THIS DEPENDENCY, I HAVE
C     INSTALLED A CHECK AGAINST A
```

```
C         LARGE VALUE FOR RE-SCALING
C         OPERATIONS.
C
          AG = 0
          I = 1
16        S = C(I) - LAMBDA
18        IF (S.GE.0.0) AG = AG + 1
          IF (S.EQ.0.0) GO TO 20
          I = I + 1
          IF (I.GT.N) GO TO 22
C
C         THE LINE BELOW CHECKS FOR
C         POSSIBLE OVERFLOW DURING THE
C         DIVISION. IF OVERFLOW WILL OCCUR,
C         TREAT S AS ZERO.
C
          IF (ABS(S).LE.1E-6) GO TO 19
          S = C(I) - LAMBDA - W(I-1) / S
          GO TO 18
C
C         GET HERE IF OVERFLOW IS POTENTIAL.
C
19        AG = AG + 1
          I = I - 1
20        I = I + 2
          IF (I.LE.N) GO TO 16
C
C         THE COMPUTATION OF AG IS
C         COMPLETE. ADJUST THE INTERVAL.
C
22        IF (AG.GE.K) GO TO 24
          U = LAMBDA
          GO TO 14
24        L = LAMBDA
          M = MIN0(AG,NEV)
          DO 26 I = K,M
26        E(I) = LAMBDA
          GO TO 14
C
C         THE KTH EIGENVALUE IS COMPUTED.
C         STORE IN E(K) AND PROCEED.
C
30        E(K) = LAMBDA
          K = K + 1
          IF (K.LE.NEV) GO TO 13
C
C         THIS COMPLETES THE EIGENVALUE
C         COMPUTATIONS.
C
          WRITE(1,355)
355       FORMAT('COMPLETED EIGENVALUE
          COMPUTATIONS.')
C
          RETURN
          END
```

STOK SOFTWARE, INC.

Humanizing the Computer

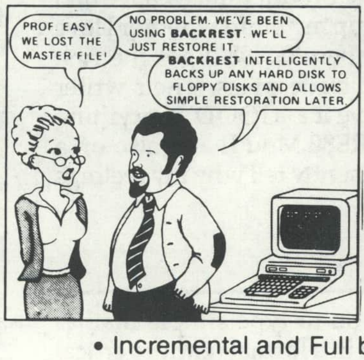
STOK PILOT™

Put your knowledge of your office environment into your computer so that your personnel will be properly guided in your absence.

STOK PILOT is a control language that allows easy development of a menu driven environment as well as an on line instructional utility for any CP/M or MP/M application. It can guide the user through an entire process without requiring the user to enter cumbersome system commands, hence making the system transparent to the user.

STOK PILOT can chain to any "COM" file program, or series of "COM" files, and regain control when the last program ends. This, and other unique features make it easy to design complete turnkey systems.

Disk and manual - $129.95. Manual alone - $14.95.


PROF. EASY - WE LOST THE MASTER FILE!
NO PROBLEM. WE'VE BEEN USING BACKREST. WE'LL JUST RESTORE IT. BACKREST INTELLIGENTLY BACKS UP ANY HARD DISK TO FLOPPY DISKS AND ALLOWS SIMPLE RESTORATION LATER.

BackRest™

Hard Disk Backup, Restore and more!

- Incremental and Full backup.
- True copying of random files.
- Split large files if necessary.
- Migrate or delete selected files.
- Automatically restore bad files.
- Print Management reports.
- Requires CP/M 2.2, CP/M 3 or MP/M.

$99.95

VISA MasterCard

THE RANDOM HOUSE ELECTRONIC THESAURUS®

$140.00

Stok Software Inc.

17 West 17th St.
New York, NY
10011
212 / 243-1444

SuperDO & SuperSUB - $29.00

SuperDO allows the CP/M operator to type a string of commands that will execute one at a time. So you can walk away for a while and let your computer do its thing. Example:

A>DO ASM PROG1; LOAD PROG1; ASM PROG2; LOAD PROG2;DIR

SuperSUB is an enhanced SUBMIT command that will run on any standard CP/M 2.2 system. It runs faster than SUBMIT because it buffers the commands in memory.

Random House and the House design are TM of Random House, Inc. CP/M - MP/M are TM of Digital Research, Inc.

Dealer inquiries invited.

(continued on next page)

Feature

Math* Reviewed

by Robert P. VanNatta

A program appearing lately on the market is a product called MATH*. If you think that this product probably has something to do with some more famous 'STAR' products you are half-right.

It is not a MicroPro product, and if MicroPro thinks they own asterisks as well as Stars as part of their trademark, this product may make some money for the lawyers. Actually, the product is identified as a product of Force Two, Ltd. and is distributed by Lifeboat Associates and (I think) others.

In actuality, it is an overgrown patch for MicroPro's WordStar designed to give WordStar crude mathematical capabilities. It comes in a 28k file called *MATH.COM*. This single file is an install routine and patch combined.

The installation routine is menu driven and can be accomplished by a monkey in five minutes or less. The first question asks you to verify that you really want to proceed; the following four questions ask you to identify the source and destination file names and drive numbers. The result is a new WORDSTAR root file which has grown to 22k in size. Once installed, the *MATH.COM* file can be discarded and the patch becomes self-contained baggage that loads as a part of the WordStar root.

I have waxed hot and cold on the product since I have had it, and frankly, can't fully make up my mind whether I love it or hate it. *InfoWorld* (September 20, 1982) got the first review that I have seen in the trade press. Their writer tested it on an Osborne and gave it a 'GOOD' everything rating. I did my testing on a TRS80 Mod II and also on a TRS80 Model 16, and I shall presently tell why my feelings are not so bland.

What It Does

Math* is designed to permit you to type simple mathematical equations on the screen in a format (either vertical or horizontal) that a grade-school student would use. When you are ready for the answer you type a ≈M in the location of the desired answer, and, presto, the correct answer will appear.

By way of example, if you type 2+2= and then the magic control sequence (a 7Eh followed by an upper or lower case 'm') the answer will be substituted for your control sequence.

All four basic math functions are supported: add, subtract, multiply, and divide. All numbers are limited to a maximum length of 19 digits. The following demonstrates the precision of the calculations:

a. 429981696 * 429981696 = 184884258895036416
b. 184884258895036416 / 429981696.1313 = 42998169
c. 429981696.1313 * 42998169 = 41681563437832.523

By contrast CB-80 BASIC compiler programmed as follows:

```
print 429981696 * 429981696
print 184884258895036416 / 429981696.1313
print 429981696.1313 * 42998169
```

generates the following responses:

```
a. 1.84884258895E 17
b. 429981695.869
c. 1.84884256372E 16
```

The comparison is interesting. On the first equation Math* appears to edge out CB-80 on accuracy. MATH* is correct and CB-80 has truncated the six least significant digits.

But look at the next two! Math* completely falls apart. On the second one, it puts the decimal point in the wrong place and doesn't give a word of warning. Similarly the third equation which ought to be the proof of the second equation goes completely bananas.

Paragraph 3.6 of the documentation states as follows:

"Math* automatically rounds and reduces numbers internally so that the results will fit in 19 digits. This feature is beyond user control."

Elsewhere, you are told that entry of numbers of more than 19 digits, or attempted answers of more than 19 digits length will generate an error message.

In fairness, I must confess that I am not sure that I have a whole lot of use for calculations using numbers as large as appear in these examples. I am irritated, however, that the documentation would lead you to believe that you can expect the correct answer to all math problems if the answer is less than 19 digits.

Documentation

The Documentation consists of 10 pages with a table of contents at the beginning and an index at the end. Unfortunately, the table of contents and the index are the strong points of the manual. I am satisfied that it is not deserving of the 'good' rating assigned by *Infoworld*. The problem that bothers me in rating the manual is the feeling that I don't know for sure what a manual ought to contain. If you believe that a manual is adequate if it describes how to use a product in a brief and concise fashion you will be happy with this manual. I think anyone can learn to use Math* in about five minutes, and a lengthy manual would only slow up the process.

The first five pages of the manual contain installation instructions and some misleading specifications of the program's abilities. Pages 6 and 7 describe the basic math functions and provide examples. Pages 8 and 9 tell how to use the advanced features (these elegant commands permit you to force the answer into a format similar to the basic PRINT USING statement). Page 10 lists the four error messages.

The matters just mentioned are adequately described and unless you would downgrade the manual for claiming an accuracy level beyond the truth, the manual is excellent. Unfortunately, I personally conclude that the manual ought to have more technical information and that it is, therefore, 'POOR'.

First off, from reading the manual there is no way that you can come within a mile of telling whether this product will even work with your particular computer configuration. You are told in the manual that the product is written in Z-80 assembly language and requires a Z-80 processor. You are also told that it may not work if you have custom console drivers. Other than that, you are on your own.

I understand all this to mean that if your computer is based on an Intel 8080 chip you are out of luck. I know not, however, what the score is with respect to the 8085 chip which some of you have. (Ed. Note: The 8085 will not work either.)

More frustrating than that is trying to figure out what a 'custom console driver' is. I first, blandly, assumed that this term referred to those exotic computer systems that were not included on the WordStar menu installation list. I quickly found out that was wrong. I next dumped Math* under DDT and found that it was apparently accessed by means of a JMP instruction patched at 02C0H, which is a WORDSTAR reserved location for optional user console output routines.

Then, thinking that I was a 'certified smart guy,' I examined my version of WordStar only to find that this location was, in fact, unused. Frustration! The long and short of it is that Math* won't work on either a Radio Shack Mod II or 16 with Pickles and Trout CP/M or with this same equipment using Lifeboat CP/M if you use the MicroPro menu installation routine intended for that product.

Fortunately for this reviewer, Lifeboat CP/M happens to emulate an L/S ADM 31 terminal, and WordStar will work on the Radio Shack equipment under Lifeboat CP/M if installed according to the ADM 31 terminal conventions. The difference is that one drives the memory mapped video board and the other bypasses the video board and gives you the less desirable serial terminal effect. (Ed. Note: Lifeboat recommends installing as TRS-II under the FMG option, rather than as ADM-31.)

I believe that a minimum attribute the manual should have is a list of computers whose names appear on the WordStar installation list with which this product won't work. I have just listed a couple of pretty significant ones, and I assume there are others. From my narrow experience I assume that Math* simply doesn't like video boards but I can't tell for sure.

The manual also suggests that the installation routine will sort out those unacceptable terminals and abort the installation. I did not find this to be the case. On my equipment, the installation routines worked fine and reported a successful installation. The problem is that when I use the video board the equation seems 'invisible' to Math* and the "IMPROPER NUMBER" error message displays. There is no technical information provided about the inner workings of Math*, but I suspect that it, in fact, literally 'reads the screen' in order to find the equation for the calculation, and my video board prevents this from working.

Regardless of whether my 'screen reading' theory of operation is correct, it defines another major limitation of the program. This limitation is documented but still serious. It is required that all figures that are to be part of the equation be actually displayed at the time the calculation is attempted.

My first impression of Math* was that its principal utility would be adding long columns of numbers in business reports. My enthusiasm waned a bit, however, when I found that you had to get all the numbers on the screen at the same time. It cannot even handle adding numbers which are concealed by the help menus.

Similarly, display items which are generally invisible to WordStar will cause erroneous results without warning. For example, the WordStar page break display will befuddle Math*. The manual doesn't tell you that, but if you attempt to add a column of figures which is interrupted by the page break display, the figures above the display will be ignored.[1]

Memory Requirements

The manual successfully avoids any mention of the possibility that Math* might have some memory requirements. It, in fact, does. Specifically, Math* is loaded in memory at 4500h. This is quite a way above the normal end of the WordStar root file. I assume the blank space in the middle is used by WordStar to hold its overlays and the like. As nearly as I can tell, the memory area at 4500h and above is normally used by WordStar for storage of the document being processed.

I expected, and was able to confirm by actual testing, that the dynamic storage area available for document storage was reduced by the installation of Math*. On a 64k system I calculated the reduction of free storage area to be about 9%. This does not prevent the operation of WordStar since WordStar automatically creates workfiles as necessary when a document won't completely fit into memory, but is an indication of a modest increase in disk activity when large files are being processed. The work that I have done has all been on high performance eight-inch systems and on such a system, the loss of performance is not noticeable; but I suspect that on those low performance systems it would be one more stone to bear.

It occurred to me while making those observations that this problem would be aggravated on systems which must run on less than 64k of memory. I don't actually have a smaller system, but believe I can emulate one for testing purposes. I used MOVCPM to construct a 48k system, which I then booted. The result was that the regular WordStar worked (MicroPro says it will run in 45k); however, when I attempted to load WordStar with Math* on its back, WordStar crashed with an out of memory error. It is my notion that this is a reliable indication that if you must run WordStar in a 48K or less system you had best save your money and not buy Math*.

(continued on next page)

For lack of anything else to do, I also constructed a 52k system and found that Math* would run in 52k, but that it caused a 25% reduction in dynamic storage area.

Other WordStar Accessories

Math* appears to be compatible with both MailMerge and Spellstar; however, the latter generates an irritating display of the Math* copyright notice when you reload WordStar.

Patches and Improvements

The word 'extention' (sic) is displayed twice in the installation routine. I would personally prefer that 'extension' be spelled in the conventional manner.

If Math* is installed, you get the disgusting opportunity of watching a Math* copyright notice roll onto the screen just ahead of the WordStar copyright notice. Math* accomplishes this by using CP/M function 9 to print a string of line feeds to the terminal. This is a method of clearing the screen that will work on nearly any terminal, but I don't have to like it. This infamous string of linefeeds is found at 4500h. If you are even slightly inclined to use DDT, I would urge a patch beginning at 4500h which included first your terminal clear screen code and then a '$' (the dollar sign is the standard CP/M end of string flag). This will clear the screen and avoid the tedious display of all the line feeds.

The other thing that needs attention is the control sequence for invoking Math*. As mentioned earlier, this requires a ~M to accomplish. (Ed. Note: A more recent version allows the tilde to be replaced by the user's choice of lead-in.) I, at least, wonder how many terminals lack the '~' key altogether. In any event, it is a relatively inconvenient CTRL 6 on my Radio Shack. The 'M' which must follow may not be a CTRL M. This convention deviates from the WordStar convention for two-key sequences in that all WordStar two-key sequences will accept a Control character after the lead-in character. I had hoped to find this control sequence in the WordStar jump table which is used to define all of the regular WordStar function keys. If it had been there, it would have been a simple task to redefine the codes. Unfortunately, it is not. Rather, it seems to be thoroughly buried within Math* itself and no obvious way exists to modify that code.

Expect Math Functions to be Standard

I can't help but think that math functions will soon be standard features on most word processing programs. Math* disappointed me because I had hoped for more. Objectively, it is impossible to say whether Math* is good or bad. It is the first creature of its kind on the market. For this reason, there is no standard by which to judge this product. It may well be that this represents the best possible math program which can ever be grafted onto WordStar. The heritage of WordStar can be traced, through WordMaster, to its origin as that ubiquitous line editor ED.COM. The heritage is that of a program writer, molded into a word processer. I am relatively certain that the idea of making WordStar add and subtract was not in the design criteria around which it was written.

The first fellow who writes an easy-to-use program that combines the features of a good word processing program with the features of a good spreadsheet is surely going to get very rich. (Ed. Note: Consider T/Maker III, soon to be reviewed.)

Conclusions

A word processer that can add should not be expected to replace a spreadsheet program. If it has a place at all, it should be at home in an office where financial reports are so occasionally prepared, that a spreadsheet is not feasible. In order to be cost effective, the math function would have to be versatile so as to permit removal of the ever present adding machine. At the same time it must be a sufficiently natural extension of the word processing package so as to not present a training headache.

It is my assumption that if number crunching is the rule rather than the exception, WordStar would be abandoned altogether for a spreadsheet of some kind.

In terms of ease of use, Math* clearly meets the criteria of being a natural extension of WordStar. Someone who already knows how to use WordStar should be able to use Math* with minimal difficulty after about five minutes of training. The more serious question revolves around whether Math* is sufficiently flexible so as to permit the occasional report writer to scrap the adding machine. In this respect, Math* fails, in my opinion, for the following reasons: 1) columns of figures to be added must all be on the screen at once; 2) WordStar page break display interferes with column adding; 3) no more than a single blank line is allowed in a column of figures; 4) there is no visual confirmation of the figures which Math* considers to be part of the equation; 5) the overflow trapping where over sized numbers are used needs to be improved.

If you are a grade school student wanting to do your homework, Math* would appear to be almost as convenient as a pocket calculator; but I fear that it would fail to help in preparing even the simplest profit and loss statement, or expense account itemization.

Add Some Horsepower

Math* in my opinion needs to trade a little of its ease of use for horsepower. As a long time WordStar user I am accustomed to marking blocks and columns before I do something with that block or column. Math* unnerves me just as WordStar would if it somehow permitted you to move blocks without marking them first. The usefulness of Math* would be greatly enhanced if a calculation field could be defined and highlighted with the block markers ↑KB & ↑KK.[2] Then, if a calculation could be made which would include all numeric values within the block thus defined, irrespective of whether it was on the screen, I would throw my adding machine out the window, call my friends and tell them to buy the product at once, and write a rave review. As it is, I can only yawn. ■

FOOTNOTES

[1] This limitation can be 'worked around' by hiding the page break display.

[2] WordStar version 2.26 cannot mark a column; its use would be precluded.

Feature

A Comparison of C Compilers

by Bruce Hunter

C roots go all the way back to 1970 when Ken Thompson, a principle architect of Unix, wrote a new language called B, based on Martin Richards' BCPL. B was a small typeless language. Dennis M. Ritchie, a co-worker of Thompson's at Bell Labs, completed the metamorphosis of B into the very remarkable language we all know now as C. C, particularly its development, cannot logically be separated from Unix. Ken Thompson started the development of Unix in '69, and by '72, with the aid of Dennis Ritchie, Unix was a reality. C and Unix were, therefore, developed side by side. The operating system and the language were the product of programmers, intended for use by programmers.

Until recently, Bell Labs was forbidden by its corporate charter to produce software for profit. As a consequence, Unix and C were not released as a commercial venture. In 1975, Version 6 of Unix was released, primarily to universities and other non-profit organizations. The world outside Bell was quick to recognize an exceptional system and language. By 1978, the present standard of the language, 'The C Programming Language', had been written by Brian W. Kernighan and Dennis M. Ritchie. Known now as Unix 7 C, it is now the standard by which all other versions of C are judged.

In the process of writing a book for Sybex, 'An Introduction to C', I have reviewed over a half dozen C compilers. As I went through each compiler, I was amazed to discover just how closely they adhered to the definition of the language, i.e. Kernighan & Ritchie's book, in spite of the lack of any official "standard" by ANSI. This is surprising, mainly because computer languages rarely stay in a static or permanent state. They are "living" entities from which multiple offspring are consistently produced, and sometimes the offspring bear only a distant resemblance to the parent language. This is "progress", and I am not one to fight the inevitable. Nor do I propose that creativity in any form be stifled. I merely wish to point out that although the creation of related sets of a language can have its decided advantages, such as a subset of a language making that language accessible to more people, sometimes there are disadvantages involved such as the loss of portability in the various "supersets" of some languages today.

To examine this point in more detail, let's briefly consider some of the subsets of the larger languages. With 8 bit machines still the common denominator for the "man on the street", few can enjoy the languages written for big machines without subsets. The most popular version of Fortran 66 is a subset, Microsoft Fortran. PL/I has followed in the same footsteps, and the most popular version for micros, Digital's PL/I-80, is a subset of Subset G, which itself is a subset of the full set of PL/I. The Department of Defense in their definition of Ada stated that "Ada must be Ada", meaning that only one version shall ever exist and that version shall be the full set. But before DOD was even finished fully defining the language, two subsets were available for sale to the micro world. There are only benefits and no problems to these subsets as long as the subsets are upward compatible.

However, let's look at this phenomenon in reverse, the 'supersets'. The original definitions of two famous languages, BASIC and Pascal, were teaching languages that were small enough to be learned by novice programmers without inflicting too much pain in the process. BASIC and Pascal are marvellous teaching languages, and they are very pleasant vehicles on which to learn programming. What has happened, however, is that once having learned these languages, the new programmers were no more ready to give up their original language than Linus was to give up his blanket. The result has been the creation of larger and larger sets of the original languages to make up for their original 'shortcomings'. By "shortcomings" I don't mean to sound snobbish; I mean they lacked specific programming tools like pointer handling, data structures, etc. There is no harm in the creation of supersets in an insular environment. It's delightful to see the ingenious creations that have arisen from the original, like Pascal MT+. Unfortunately, the moment a programmer starts to program for others, portability becomes a necessity, and supersets have no portability. Try to compile an MBASIC compiler program on a CBASIC compiler, and watch the syntax errors fly.

The problem here is this: who is going to define an enhancement to a superset? Without a defined language standard, there can be no hope for portability, and with no portability, the programmer is limited as to his options which are already limited by the multiplicity of hardware and operating system dependencies prevalent today.

C is neither a large language a nor small one. Unix and C were written on and for 16/32 bit machines, and today's 16 bit C's available to the public are very close to Unix 7 C. An implementation of the full set small enough to run on a typical 8 bit/64 kilobyte machine is almost 'the impossible dream'. However, the majority of today's C programmers have cut their teeth on the 8 bit C subsets, and until 16 bit machines become the most common machine (if they ever do), most of you will learn C on one of these subsets. As a consequence, the 8 bit C subsets are as important to the language as are the full set implementations as far as the general public is concerned. And unlike other powerful languages, like PL/I, there is a definite general interest in C. C is for those who are progressing or who have already progressed well beyond the beginner's stage. It's a grown-up language enabling you to write systems level code, filters, and get your hands right into the operating system. It can enhance programs written in other languages and supplement their weak points. It can save you money on utilities by enabling you to simply write them yourself.

For those of you who are interested in purchasing a C compiler, here are some things to consider. When

(continued on next page)

shopping for a language package, the intended applications will dictate just how full a set is required. If the application is to be an operating system, a word processor, a disk utility, or the like, there will be little need for floating point numbers, but compact object code will be a necessity. If business, scientific, or engineering programs are the goal, then float and double precision and a math library are necessities, and code generation is entirely a secondary consideration. What makes the available C packages different are the size of the implementation, the degree of code optimization, and the speed of execution and compilation.

To specifics: compilers grind their way from source code to object code in a number of passes. They pre-process, parse, optimize, and then will or won't produce intermediate code before generating the final binary object code. The "will or won't" is the key. If a macro is to be produced rather than a free standing program, it will need to be in relocatable code, particularly if it is to be linked with code generated by another language. Programming utilities are a good example. Access Manager-80, a data base creation program by DRI, is a fine example of a series of routines to be linked to a number of languages, and it is in relocatable code.

C packages like C/80, SuperSoft C, Q/C, and Aztec C produce 8080 assembly. The assembly code is in turn passed through a macro loader and converted into relocatable format or a REL file. There are a number of benefits to this. The code can be debugged by SID, DRI's symbolic instruction debugger. It can be linked to anything that is linkable under M80, Microsoft's macro linker, or MAC, Digital's version of the same. The bad news is that it takes a number of passes to do all these things, so it takes longer. Consequently, when it's getting quite late and you are getting very tired and the same damn bug that has been plaguing the program is still there after the umpteenth re-editing, the drudgery of parse, optimize, macro load and link can get to be a bit much.

The flip side of the coin is a two pass system. The parser and compiler are brought up automatically, so one command does both jobs. The resultant code is then linked in one pass. This means just two apparent operations from source code to object code. BD Systems C is a compiler of this type. What is the catch? No REL file. The intermediate code can be linked to any other BDS C program but not to "Microsoft compatible" programs. SID can not be used. BDS does have a utility, however, to produce 8080 assembly. And fortunately, the latest version 1.5 has a debugger of its own. For most people, the lack of REL files is more than made up by the speed of the operation. Which of the two compilation systems produces the most optimized code? That is a matter of how well the optimizer has been written, not the compilation system.

Full sets and subsets, to have or not to have, that is the question. If money were no object, perhaps we all would be running Unix 7 C on a PDP 11/70, and there would be no reason for this article. If you are going to run a subset, then the question is, how much of a subset. There are so many things to consider about each compiler. Each has its positive points and relative disadvantages. Here's a brief look at the history of some of them.

BRIEF HISTORY OF THE DEVELOPMENT OF SOME 8 BIT C COMPILERS

Most people that have been into C for a while have probably cut their teeth on BD Software C. The BDS C compiler is the product of the genius of Leor Zolman. Leor, a native Californian now transplanted to Massachussetts, got into C at MIT in the 70's while working in the computer science lab there. Leor has a penchant for programming games. He discovered the game Othello on Unix and wanted to run it on his own micro. First he attempted to program Othello in assembly, but finding it a pain, he wrote a subset of C to do the job. Having a running C compiler, he realized that he had something that just might be marketable. Lifeboat picked up an exclusive distributership on the package for a number of years, but now the package is also available directly from Leor. Being the oldest C in its class, it is not surprising that there is a BDS C User's Group. Their software is public domain, and probably more people have been introduced to C through BDS than any other single version.

A totally different branch of the C tree started with Ron Cain. Ron wrote a small set of C, aptly called "Small-C". The source code of Small-C was published in Dr. Dobbs Journal (No 45). Small-C, as the name implies, is devoid of some of the Unix 7 C niceties. It does not have floating point or long integer (and indeed, most subsets do not). Two regrettable omissions are the case statement and structures. Nevertheless, the price was right, and Small-C went on to be the progenitor of a handful of other C compilers. If you are looking for an inexpensive introduction to the language, Small-C is available for $17 from the Code Works.

One of the extensions of Cain's Small-C is Small-C Plus, written by Kirk Bailey. It adds some enhancements, and it's available for $25 from Alpha Omega Computer Systems Inc.

A BRIEF EXAMINATION OF SPECIFIC C COMPILERS

C/80

With Small-C's source code available to all, it was inevitable that it would be copied and improved upon. That is the way almost anything good is created. Walt Bilofsky of the Software Toolworks created a version of C from Small-C that by now has little resemblance to it. Still excluding long and float, Walt's version, called C/80, supports just about everything in the language. Integer only, C/80 is nevertheless a most remarkable subset of Unix 7 C. It is a true subset with no atypical enhancements. It includes all the storage classes, pointers, arrays and structures. Therefore, it supports C wildness like pointers to arrays of structures of arrays of…etc. A very interesting feature offered is initializers, a feature not found in many versions costing three times as much. The full switch/case is there, and full buffered and raw file I/O. Bitfields are about the only thing besides float and long that is not present in this package. Walt's package comes complete with his own assembler which produces 8080 assembly. Walt's C/80 compiler is remarkably efficient, doing compilation faster than any of the tested 8 bit compilers with the singular exception of BDS. The entire package works well and quickly, and it produces remarkably tight object code, which means a great deal of work has gone into optimization. You need to supply the linker and macro loader or use ASM.COM, however (you need to do this with SuperSoft as well). The only shortcoming of C/80 is a good but limited function library. The library has all those functions necessary for basic I/O and even formatted output. Alloc is included as well. Past that you will either have to create your own or borrow public domain library routines. The most remarkable thing about C/80 is its price, $49.95. Dollar for dollar, this has got to be the best bargain in computerdom. C/80 is available from the Software Toolworks out here in the LA smog. Other features of the subset are I/O redirection, command line arguments, programming chaining, and a runtime trace feature to aid in hand optimization of the code.

Q/C

Another offspring of Small C is Q/C by Jim Colvin. Jim is a remarkable programmer, programming everything from assembly to PL/I on both minis and IBM mainframes. Q/C does not support the data types float, double, and long. In addition, Q/C does not include cast, size of, unions, multidimensional arrays, typedef, and #define with parameters. Structures are being added to the latest version 1.3 along with other enhancements. However, it has a very fine library using generally typical K&R type functions. One intriguing thing about Q/C is that it uses a compiler optimization technique called peep-hole optimization. The technique involves looking at a small section of the code as it is being generated by its assembly instructions and comparing the instruction sets that it generates for the most efficient. It is the same technique used by Digital Research on their soon to be released C compiler. Its effectiveness is undisputed since Q/C produces some of the smallest code of the C's, along with C/80. The package comes complete with a bound 137 page manual of excellent readability and quality. There is a chapter on compiler internals, and here's another unique feature: the full source code to the compiler is included! Priced at $95, it is available from the Code Works.

CW/C

Unfortunately, I did not personally examine this C, but it is also available from the Code Works for $75. It is a larger extension of Small-C including structures, unions, multidimensional arrays, #ifdef, etc.

INFOSOFT

An extremely powerful C package for the money is Infosoft's C. It is an integer only implementation (like most 8 bit C's) that has most of the Unix 7 features. The only features of Unix 7 C that are missing (besides float, long, etc.) are bitfields, casts, register and typedef. All of the C operators are supported, **extern** and **static** as well. Initializers are allowed in all but a few odd circumstances. A small but adequate function library is included that has the most important functions including **printf**. C isn't worth a nickle without **printf**. Quite interesting is the fact that the full set of precompiler (preprocessor) commands are included, so conditional compilation is possible. Not K&R C, but nevertheless a viable feature of the distribution package, is a series of functions to create and use 32 bit unsigned long integers. A 32 bit unsigned integer has a maximum value of 4.26E+9 which gives accuracy well into the billions and a precision of 9 digits. An entire math package is furnished with it.

Infosoft's package includes its own linker and debugger. Besides a manual, a language reference guide is included as well. It is noteworthy that Infosoft has written a number of operating systems, both single user and multi user, as well as programming utilities, an editor, and the like.

SUPERSOFT C

SuperSoft C is a serious C compiler with versions for 8080, 8086 and Z8000. Its roots are straight Unix 7. It also does not support long and float. Conditional processor commands are not supported, and the package does not include typedef, bitfields, and static storage. Also, SuperSoft does not have a macro assembler or linker. It assumes, perhaps correctly, that most users of a $250 C package will already either own Microsoft's M80 and L80 or Digital's Mac and Link. If a purchaser of the package did not have a macro linker (borrowed from some other legitimate language package or purchased separately), he can always use ASM.COM. The only problem with using ASM.COM with C packages is this: the only way macros can be handled is by preprocessor inclusion or cut and paste assembly, the first being disk space intensive and the second tedious.

The good news about SuperSoft is a fine function library, larger than the library functions described in K&R. It is not only Unix 7 compatible, but it has good compatibility with BDS C as well. SuperSoft developed their C as a tool for the development of other language and utility packages. The first two compilation passes of the compiler are the parser and the optimizer. This involves an extra step for the operator, but its stated purpose is to produce tighter code. The actual linkage of the code can be repetitive and tedious, but the use of the CP/M Submit facility takes most of the fuss out of it. I have never had a great deal of luck with Submit files, so what I did was compile under MicroShell using its shell file. When this package is used with M-80 or Mac, the code produced comes out as REL files. This allows the creation of a library of macros that are compatible with any other language(s) that use REL code. The SuperSoft compiler has a couple of pages worth of compiler switches to aid in debugging with or without SID or DDT. They have a good support group as well. SuperSoft runs around $250.00, available from SuperSoft Associates.

BDS C

BD Software C, aka BDS C, is one of the oldest of the eight bit C's, and this package has been around long enough to have the bugs worked out of it. One of the most

(continued on next page)

remarkable things about BDS C is its one pass compiler system. Once the compiler is invoked, it does all the housekeeping of parsing and optimization without any further operator intervention. The linker provided with the package also is a one pass proposition. No need for submit files, it is the fastest of the C's to get source code up and running. Compile and link time is measured in seconds, not minutes. The bad news, REL file are not produced, the intermediate code is in the form of a CRL file. On the other hand, there is nothing in BDS C that I have ever encountered in the way of a disadvantage for which Leor or one of his many friends and supporters have not written a function or utility to overcome that disadvantage (including float and initialization). CASM.C is a program written specificaly to produce ASM code. Anything that produces ASM code will produce relocatable (REL) code.

Like any of the larger implementations, BDS takes up the biggest part of two distribution disks. The function library is enormous. There is literally something for everyone. Groupings of math functions are available for float, trig, log, etc. It is not "legitimate" C float, so don't use them if you are interested in portability and Unix 7 initialization. Short of that, however, BDS will do about anything. Most code written in BDS is written with portability in mind, and as long as it is integer, it will run on Unix 7 C compilers. If there is a larger function library than BDS C's, I am unaware of it, although a 16 bit implementation, C86, comes close with almost 100 functions. A large number of software packages in today's marketplace like MicroShell, Mince, Scribble, and on and on have been written in BDS C. That in itself is enough to speak for the package. The newest version (1.5) has a nearly all new documentation package that is well written by Leor himself and runs close to 200 pages. Leor's function source code is a part of the distribution package which is handy for both fellow programmers and for an unscrupulous few that have felt free to rip him off. Leor's feeling about this is that he wishes that they would at least give him credit, as they well should. Leor Zolman has probably done more for C than anyone this side of Dennis M. Ritchie and Ken Thompson.

BDS C is available from Lifeboat Associates, the original distributer, from Leor Zolman, and from Dedicated Micro Systems. Updates are available to legitimate owners from CUG, the C Users Group, at $8 for the disk. BDS C runs around $150.

CAVEAT EMPTOR

Caveat emptor, or "let the buyer beware", is just as important a point to consider in choosing software as any other feature, and that is why I'm putting this right in the middle of this comparison. I'm sure all of you are aware of the potential pitfalls of buying software. For one thing, you want to be prepared for a few snags when you buy the very first versions of anything; they tend to have more bugs than a compost heap. There are exceptions, especially with companies who spend a lot of time and money beta testing their products, but a hearty dose of skepticism in evaluating various claims made by software houses can save you a lot of grief.

AZTEC C

Aztec C II is just about as full an eight bit implementation as I have encountered. It has everything from Unix 7 with the exception of bitfields. The system produces 8080 assembly and as such is compatible with the Digital/Microsoft family of programming utilities, M80, L80, MAC, SID, DDT and the like. It has its own assember and linker, and both are easy to use. Producing an ASM file upon linkage, it is a little slow, as are all of this type, but it runs much faster and easier than some. The best news here is, of course, the inclusion of float, long and double. It is an integral part of the compiler, and not one of the function packages. To support its math potential there are nearly two dozen math, log and trig functions. Aztec has a good function library that is well rounded and essentially Unix 7 plus. Aztec C is available for $195 from Manx Software Systems.

TELECON SYSTEMS C

This C is available for 8080, 8086, PDP-11 and 6809. The 8080 version comes both with and without float (although double is not implemented at this time). It is very close to Unix 7 C, supporting just about everything but bitfields. Like many C compilers, it allows assembly to be intermixed with C source code before compilation. The function library is not one of the largest, but is essentially Kernighan & Ritchie with the usual **printf**, **strcat**, and the like.

The compiler produces 8080 assembly, and no linker is furnished.
Like SuperSoft and Q/C, it is assumed you will either own M-80 and L-80 or you will cut and paste using ASM.COM. Unfortunately I did not have the documentation available at the time of this writing and therefore can't list all the features. Dan Roady of Telecon gave me a Sieve run time of 17 seconds on a 4 MHz Z80. A very respectable time, although I suspect the code was optimized. The integer version is $200, float $350, versions other than 8080 run $200 to $500.

WHITESMITHS' C

Whitesmith's C has been around long enough to be a C staple. At $600 and up, it is far from being the most popular of the C's. However, it was the first C offered in eight bit to have a nearly full implementation. In fact, its compiler will support just about every feature of Unix 7 C. However, it uses its own intermediate code called A-Natural. Not only is it not compatible with Microsoft/Digital 8080, but is unbelievably slow to compile and link, not to mention complicated. The library function tends to be a bit different as well. Many of K&R's functions are intact but the remainder of the functions are atypical for the main body of C. Thus, because of the A-natural generated code and some of the atypical though ingenious functions, portability with other C's has been lost. Whitesmiths' C is available from Whitesmiths, Ltd.

TINY C

Tiny C is literally in a class by itself. It is a small C-like language package that has both an interpreter and a compiler. The package is one of the most complete in "C-dom". It has its own text editor (PPS), a C interpreter AND its source code, a C compiler, sample programs, and 384 pages of the most voluminous documentation in the field, over half again as long as the book that defined the language.

Since I have taken the attitude that if it is not within K&R's definition of C, it is not "C", C-like is an apt description of Tiny C. It is very close to a small set of C like Small C, C/80, etc., but with important syntactic differences. Some examples are the use of brackets [] instead of braces to delimit blocks and the absence of the semicolon statement terminator. The function library has C's getchar and putchar, fopen, fread, fwrite and fclose, giving it the very basics of C I/O. The remainder of the functions are atypical to C proper, but still have their C equivalents. For example, **gs** is equivalent to gets, ps to puts, nim to atoi, etc.

On the other hand, it is hard to imagine a friendlier way to get an introduction to C, particularly if the would-be C programmer is new to structured languages, for Tiny C has a user friendly, interactive interpreter and plentiful documentation. Once the Tiny C interpreter has been mastered, then the compiler can take the programmer one step closer to Unix C. A very small version of the Unix shell called "tiny shell" handles redirection by way of the standard < and > shell operators. A few Unix-like tools are also provided. For the serious hobbyist who wants to get into C, the availability of the source code of the interpreter will keep him busy learning and, if so desired, modifying the code. The code is supplied for both 8080 machines and the PDP-11 family. A full set of stack functions are in the package, so system programming is a natural. The price is $100 for interpreter source and object code, $250 for compiler object code, available from Lifeboat Associates.

I have concentrated on 8 bit C's because I feel that the 8 bit user is still in the vast majority today. However, Unix 7 is of 16 bit parentage, and things like type long are extremely difficult on 8 bit. The 16 bit C's offer more, then, of Unix C than 8 bit C's. As a matter of fact, the 16 bit C's that I have looked at so far are about as close to full Unix 7 as one could hope for. Here's but a brief glance at three of them.

COMPUTER INNOVATIONS C86

There's not a lot to say about C86 because there's no bad news. At least none that I've been able to discover so far. All of K&R is supported, including the function library. It is very very close to Unix 7 C. George Eberhart, the author of C86 (and Tiny-C's compiler) is one of those exceptionally gifted and dedicated individuals that has caused the industry to grow at an exponential pace. C86 reflects this ability.

A great deal of pain went into keeping it compatible, not only with Unix 7 C, but with BDS. There is simply not enough in the way of praise and gratitude for software writers who go for deliberate portability and cross use. They make programming a pleasure. A nicety of the C86 package is the use of very long variable names. It goes a long way towards making self documenting code. Something to look for in 16 bit languages in general that are running on the 8086 is support for the 8087 math processor. This extremely powerful little piece of hardware runs with an 80 bit word and will do number crunching that will rival a million dollar mainframe. C86 supports this mini marvel of Intel. C86 costs $395. I have been told (by George) that soon the package will be available for most micro processors starting with the 68000. Even an 8080 version is in the planning. Although the 8080 version may not sound glamorous at first, remember this would be the only full 8 bit C available with the exception of Whitesmiths.

LATTICE (8086/8088)

Another very outstanding 16 bit C is Lattice C which is K&R C just about to the letter. Any deviations from the definition of the language found in Appendix A of Kernighan and Ritchie are so small as to be insignificant. Whereas 8 bit C's tend to be somewhat of a free-for-all when it comes to what will be implemented and what will not, CI-C86, Lattice, and Digital work very hard to maintain "to the letter" C compatibility. The library functions furnished with Lattice are classic C offering full compatibility with the mainstream of C functions. There are 150 total functions if the special functions peculiar to the IBM-PC are included.

The package comes with a well written and organized manual, the manual and the compiler both being the work of Francis Lynch. It is quite technical, and it seems to have left 'no stone unturned'. A second manual whimsically called "The C-Food Smorgasbord" contains a multitude of functions and macros for all sorts of things. Decimal (BCD) arithmetic can be done with a separate package that is part of the "Smorgasbord". The package will do floating point decimal work while maintaining a precision of 16 significant digits (a great help when working on the national debt). The precision of the main package is 6 to 7 figures for float and 15 to 16 for double. Float as handled by this special package is done in the form of a character array, not dissimilar to Leor Zolman's BD Software C. Like BDS C, the functions used with the floating package are peculiar to the package and are not main-stream C. Additional functions and macros furnished with the Smorgasbord are console and list device I/O functions, IBM-PC BIOS interface functions, and a very interesting package to interface the myriad of terminals whose manufacturers seemingly have taken such great delight in making them incompatible with each other. The last package is a potpourri of miscellaneous functions and tools.

Lattice is of mini-computer descent. Originally written for General Automation, it was recoded for the 8086 in both an MS-DOS version and a CP/M-86 version. It supports the 8087 math processor, and like all 8087 language versions will simulate the chip if it is not present. The price of this very complete package is $500 for the MS-DOS compiler and $150 for the Smorgasbord. Available from Lifeboat Associates.

DIGITAL RESEARCH

At the time of this writing, Digital Research has gone into beta testing with their own C. The compiler was written by Michael Lehman, DRI's director of Research and Development, and author of Pascal MT+. If you have ever had the pleasure of working with Pascal MT+, you will know the potential quality of work involved. DRI's C is billed as Unix 7 with no excuses. All code tested on the new implementation was run against Unix 7 C on a DEC PDP 11-70, and the new compiler was modified until identical results were produced. It's a most commendable undertaking. Like SuperSoft, DRI developed C for the purpose of developing their own software on it. Lehman

(continued on next page)

has also used "peep-hole optimization", and unbelievably tight object code is the result. Library functions for the 8087 math processor are part of the package. Bit fields are supported. A very interesting enhancement of this C is record locking, which means it maintains compatibility with MP/M, and that means ensured file security. Like most DRI languages, relocatable code is produced as a part of the source to object sequence. The linker and macro loader are part of the package.

Most of the newer C versions support command line redirection. DRI's C is no exception. Versions are being planned for the 68000 and Z8000. Although no 8080 version is in the planning at this time, it is hard to believe that Digital will pass up the opportunity of the very large 8080 market which they did so much to create. Price is not firm at this time, a nebulous 300 to 500 dollars. There is no debugger at this time, but a Lint-like error routine is built in. Like all DRI language division products, there is no charge for the use of the runtime library.

A Quantitative Approach to the Comparison of C Compilers

This has all admittedly been pretty qualitative in its analysis of today's C's. The following tables offer a more quantitative approach.

A Comparison of C Compilers

name of pkg	float double	typedef	sizeof	static	initializers	param -defines	casts	assembler & linker
C/80	no	no	yes	yes	stat	no	yes	yes
Infosoft	no	no	yes	yes	yes	yes	no	yes
Q/C	no	no	no	yes	yes	no	no	no
BDS	fun	no	yes	no	fun	yes	yes	yes*
SprSft	no	no			no			no
Telecon	float	yes	yes	yes	yes	yes	yes	no
Wsmth	yes	yes	yes	yes	yes	yes	yes	yes**
Aztec II	yes	yes	yes	yes	yes	yes	yes	yes
Lattice	yes	yes	yes	yes	yes	yes	yes	yes
C86	yes	yes	yes	yes	yes	yes	yes	yes
DRI	yes	yes	yes	yes	yes	yes	yes	yes

fun - done by function not part of the std implementation
* bds compiles direct 8080
** a-natural code

name of pkg	bit field	struct unions	nbr of functs	I/O redirct	manual length	$	Comments
C/80	no	yes	13	yes	35	49.95	8080 cpm/hdos
Infosoft	no	yes	24	yes	38	59	
Q/C	no	no**	56	yes	135	95	has source code
DBS	no	yes	96+	yes	181+	150*	debugger, telnet+
SprSft	no	yes	82		81	250	8080,8086,Z8000
Telecon	no	yes		yes	30	350	
Wsmith	yes	yes	95		140+	600+	unix,rsx,isis,etc.
Aztec II		yes	77	yes	96	195	sid support
Lattice	yes	yes	93+73	yes	163+63	500+150	8086 & 8088
C86		yes	95	yes	142	395	8086 plus 8087 spt
DRI	yes	yes	unk	yes	unk		

* sometimes less
** yes on ver 3

Some time ago Jim Gilbreath, the head of Computer Sciences and Simulation Department at the Naval Ocean Systems Center, decided to compare various languages and compilers for operational speed. To do so, he used a program called the Sieve of Eratosthenes as a benchmark program, a program designed to test the capabilities of a given system. Although he probably never intended it to become "the" benchmark, his article which appeared in Byte Magazine, September 1981, has become a "classic". C is one of the fastest languages run, and it also has some of the most compact code. The benchmark program is an ideal test for C, and Jim Gilbreath brought out another article in Byte, January '83, comparing C and Pascal compilers. Some of the data following is from that article, to which I have added my own interpretations.

Not only operation speed is tested by the Sieve program, but the length of the object code, and compilation and load time as well. About the only thing missing from the test is a "bug" test! Since the Sieve quantifies the comparison, it is an ideal medium to complete this article on a comparison of C compilers. Because compact code is one of the reasons that C is chosen as a language, I will list the C compilers tested first by order of memory used in K bytes (compiled length of a program from origin to end of file in K bytes).

CW/C	1.8
C/80	3.1
Q/C	3.3
BDS	3.7
C86	4.1
Telecon	5.7
Aztec	8.5
Infosoft	8.6
Lattice	11.1
Whitesmiths	12.0
SuperSoft	17.7

The offsprings of Small C do pretty well in this department, with CW/C being nothing short of miraculous. Very notable are BDS, a very large C system, and C86, a full set of C with the burden of having to work in a 16 bit environment. (Sixteen bits is fast for numeric execution, but it is not noted for its storage efficiency).

Now for speed of execution. To keep the results consistent, the tests were conducted on a Z80 running at 4 MHz. The exception was C86 which had to be run on a 8Mhz 8086 (and Lattice on an IBM-PC 8088 at 4.7 MHz). Time is in seconds and no attempt was made to optimize performance.

C86	7
Lattice	10 (4.7 MHz on 8088)
C/80	25
Whitesmiths	26
Aztec	33
SuperSoft	34
Telecon	38

BDS C v1.43	40 40*
CW/C	53
Q/C	49
Infosoft	51
CW/C	53

*Newest version 1.5 ran on my wife's CCS 2210, Z80, 4MHz, at 34 seconds.

When it comes to speed of execution, the Small C offspring didn't fare so well, with the notable exception of C/80. Whitesmith's comes in a speedy third. If you double C86's time, it will be about a fair comparison against the others at 4 MHz. Even at an interpolated 14 seconds, it's impressive.

There is a passion among C compiler writers for optimization. Code optimization is a religion for the creators of C packages, and versions producing tighter code seem to be coming out monthly. BDS C, for example, has just undergone a major rewrite going to version 1.5 for primarily that purpose.

Besides code optimization, there is also data allocation optimization. On most C versions, this is available by compiler switch. For most programming applications, the program will be I/O bound by the slowness of the peripherals. In the case of programs with a great deal of calculations, however, the I/O is critical for fast running code. The following is the sieve run time with optimized data allocation.

C86	7 (8 MHz)
BDS C	15
Whitesmiths	16
Aztec	21
C/80	26
Q/C	26
Telecon	28
InfoSoft	28
CW/C	30
SuperSoft	31

The last criterion is compilation and link time. When you are into some heavy development work, continually recompiling after re-editing, seconds can seem like hours. The time in the following table is in seconds.

BDS	21
C/80	37
Q/C	49
Lattice	66
C86	58
CW/C	71
SuperSoft	85
Aztec	86
Infosoft	96
Telecon	201
Whitesmith's	310

BD Systems C two pass system is hard to beat for speed of compilation. Again, C/80 is notable, particularly in the light of its producing intermediate code that is capable of being relocatable. Obviously, there is a close correlation between generated code and compilation time.

Today, more than ever, C is expanding its power, in increasingly refined implementations, and its influence, in the number of installed C systems. It is a major language of this decade. There are few producers of C language packages that are not constantly making new versions with more features, and code optimization has practically become a "religion'. We all can look forward to even better and more powerful C's. We, as users, are fortunate to have such a wide variety of viable options in choosing the packages of our choice.

Change of Address

Please notify us immediately if you move. Use the form below. In the section marked "Old Address," affix your *Lifelines* mailing label—or write out your old address exactly as it appears on the label. This will help the *Lifelines* Circulation Department to expedite your request.

New Address:

NAME ____________________

COMPANY ____________________

STREET ADDRESS ____________________

CITY ____________________ STATE ________

ZIP CODE ____________________

Old Address:

NAME ____________________

COMPANY ____________________

STREET ADDRESS ____________________

CITY ____________________ STATE ________

ZIP CODE ____________________

The Transporter

by Van Court Hare

The Transporter (Workman & Associates, 112 Marion Ave., Pasadena CA 91106, 213-796-4401, $69.50) is a brief collection of programs which, in conjunction with CP/M's PIP and LOAD utilities, permits transmission of any type file from one machine to another. Only one copy of *The Transporter* need be available—at the transmitter.

The Transporter can, in fact, transport itself—or any other program—from sending to receiving location, without the need for any additional communication programs. No special installation or "custom patch" is necessary.

The Transporter's procedures, however, require substantial operator supervision. The transmission code employed is three times slower to transmit (and requires three times more receiving memory) than direct binary communication.

Heavy volume users will prefer dual installation, hand-shaking communications programs, such as MODEM7 or Byrom Software's BSTAM for their automatic features—even though some local installation effort may be needed.

Handshaking is the ability of one machine to "talk back" to another, and so to stop transmission when necessary, to use error checking protocols, to call for retransmission of bad data, and so on. Fully automatic communication and file transfer systems require compatible communications programs at both transmitter and receiver to handle handshaking; in addition, *full duplex*, or simultaneous two-way communication facilities, must be used. *The Transporter* does not provide any form of handshaking, although it does provide for error checking after file transfer.

Some Reference Sources

The latest version of the public domain program MODEM7 is on Volume 84 of the CPM User's Group series, see *Lifelines/The Software Magazine*, September, 1982, p.28. CPMUG is at 1651 Third Ave., New York City 10028, and current duplication cost is $15/volume. No telephone, so write.

Byrom Software's BSTAM, and its cousin BSTMS, are $150 list each, and widely distributed. These products are seldom advertised; they don't need it, they're so good. Early reviews appeared in *Lifelines/The Software Magazine*: BSTAM in November, 1980, p. 2; BSTMS in September, 1980, p.6. Both programs are mature, tested products. Even today you will go far to find their equivalent for ease of use. Their reliability is unquestioned.

An Opinion

The Transporter will undoubtedly be used by many to trade themselves up from one-ended to two-ended operation. For example, *The Transporter* can send MODEM7 to a remote location where it can be installed, then used with its parent to obtain the advanced features just described. Needless to say, commercial programs may be subject to the same fate—in violation of their license agreements. Incidentally, the name *Transporter* is from Star Trek: "Beam me up, Scotty."

The Transporter in Action

The Transporter distribution disk comes with 2K of original code by Anton Pietsch (SEND-HEX.COM) and nine public domain programs: CRCK.COM (from CPMUG Volume 84) to do cyclic redundancy checks on the files sent and received; NULLKILL.COM to eliminate unwanted nulls from text files; and several .DOC files. Jerry Prounelle, BYTE's monthly columnist, wrote the documentation, which includes a general discusion of file communication using PIP, plus specific instructions for use of SEND-HEX.

SEND-HEX, the essential program on the Transporter distribution disk, is placed on the sending machine (TX hereafter), which must be able to transmit through the port assigned to CP/M's PUN: logical device. The receiving machine (RX, hereafter) picks up the message on the port assigned to CP/M's RDR: logical device.

For practical purposes, and ease of hookup, these communications ports will usually be serial, RS-232 types; the characteristics of both communication ports—baud rate, number of data and stop bits, parity if any—must match exactly. If it is not the default condition of your present CP/M installation, CP/M's STAT command can assign your TX serial port to PUN: and your RX serial port to RDR:. The console device must be on a separate physical port on both the TX and RX machines, because the TX console controls transmission and the RX console controls reception. The required setup is aptly described in the SEND-HEX documentation.

A Seven-Step Blastoff

Now the fun begins! Oil up your swivel chair—you now need to operate both TX and RX at once. Or, if you are using a phone-modem hookup to a remote location (more difficult) you need a sturdy helper at the other end and some form of voice communication. You may have to fiddle a bit to adjust the TX operation to RX's capabilities. Here is how it goes:

1. The sender at TX types SEND-HEX[d]Filename nn (where nn is an optional number in the range 1-15 which controls the length of data blocks transmitted), then

<RETURN>. For example,

```
A>SEND-HEX B:Myfile.COM 10
    <RETURN>
```

RX has no way to tell TX to pause when the RX memory fills up and disk storage is initiated. Consequently, long files must be sent in several segments to avoid loss of data. Segmentation of the TX file is the purpose of the SEND-HEX "nn" parameter. The size of RX's working memory determines the value of "nn" to use.

Omission of "nn" places *no* limit on the maximum length block for transmission. (TX files of 10-15K can usually be received completely without segmentation if RX is a 64K machine.) Larger files will require (a one time) experimental adjustment of "nn" (try, say 15, then 14, then 13, and so on) until you find the *largest* segment size which RX can accept in one shot. More on this momentarily. Whatever value of "nn" has been chosen, SEND-HEX will say "Current Segment Number = 01 / Press RETURN when receiving computer is ready.'

2. At RX type PIP [d:] File1.HEX=RDR:, then <RETURN>. This preparation for file reception may be done at any time before the next step.

3. Finally, at TX type a <RETURN>. Transmission starts. SEND-HEX converts the original file at TX into standard Intel Hex format records and transmits them to RX, where PIP receives them (The user need know nothing about Intel Hex files, except that they are about three times larger than the original. However, it is the intermediate Intel HEX code which ties the two CP/M machines together.)

4. Three conditions can now occur at RX:

(a) If the RX working memory is adequately spacious to hold the entire TX Intel Hex segment transmitted, the complete segment passes from TX to the RX working memory with no interruption or chopoff. Upon reaching the end-of-file mark, PIP stores the received segment as a complete file. If TX reports "END OF FILE," you are done sending, and go to Step 5, below.

(b) If TX reports another segment is still available to send, you must set up PIP at RX again with another name; for example, PIP[d:]File2.HEX=RDR:, and the operator at TX then strikes <RETURN>. You can continue in this way, segment-by segment until the entire file is complete and "END OF FILE" appears at TX. Now go to STEP 5.

(c) If the selected TX segment size was too large, the RX memory will be exceeded before end-of-file, transmitted data will be lost, and the received segment will be erroneous. If you are sitting in front of both RX and TX, you will pick up this error condition immediately; the RX disk will start up before the TX console reports END OF FILE or next segment number. If the operator does not catch a memory overrun, the LOAD operation at Step 5 or a final CRC check will. If you do get lost data, go back to Step 1, make "nn" smaller, and try again. (Be sure to remember the largest value of "nn" you were able successfully to use for RX; it is the value to use thereafter for the same RX machine.)

5. When TX finally reports END OF FILE, you will have one or more file segments with suffix .HEX at RX. The objective is to end up with only one .HEX file; consequently, if you have more than one segment, use PIP to put the segments together in correct sequence; for example:

PIP File.HEX=File1.HEX,File2.HEX,
File3.HEX,....

6. To get the RX file back to its original form, type "LOAD File" (no quotes) on the RX machine following CP/M rules. (LOAD assumes a .HEX extension, so don't type it. The name typed for File is the final name you want to use.) CP/M's LOAD command converts an Intel Hex file format into a .COM file and simultaneously checks the .HEX file format for accuracy. (You can Type a .HEX file and look at it if you like; you will see the format includes fixed line lengths and structure.) An invalid Intel Hex record is an error. Thus, the LOAD operation can have two outcomes:

(a) *Successful LOAD*—You will get a get a CP/M prompt, and you are done. If the TX file was originally a .COM file, the received file name is also correctly formed. If the TX file was not a .COM file, use CP/M's REN command to rename it with the appropriate original extension.

(b) *Unsuccessful LOAD*—There was a problem with the received file format (invalid Intel Hex record or file format).

In the latter event, the transmitted segments were probably too large, so try again from Step 1. If a correctly sized segment process results in a bad LOAD, you have a noisy or otherwise faulty communication hookup, so check and try again from Step 1.

7. As an additional check on accuracy, CRCK.COM is provided for computing a "cyclic redundancy check" number for the original and the received files. (You can use SEND-HEX to transport CRCK.COM to RX if necessary.) AT TX type CRCK Filename and <RETURN>. You will get a hexadecimal number on the console. Type the same line at RX. You will again get a hexadecimal number. If the TX and RX values do not agree exactly, there is some discrepancy between the two files, most probably due to line noise. For .COM files, there is usually no option but try-try-again from Step 1.

Because the process described above must be repeated for each separate original file transported, *The Transporter* is a tedious way to transfer entire disks, but is a satisfactory method for passing across one or two critical programs.

BSTAM

For comparison, consider Byron Software's BSTAM (a commercial grade, automatic CP/M to CP/M file communicator) which requires careful installation on both TX and RX.

BSTAM bypasses CP/M's CBIOS and uses machine language "drivers" for direct communication port access. (This avoids erratic timing and CP/M's possible response to any strange control characters.) The installed drivers must be appropriate to the machine at hand. Good vendors will either do this installation for you, or provide a disk full of install options at no extra charge. (For drivers currently supplied, see notes in *Lifelines/The Software Magazine*, June, 1982, p. 82, which also apply to BSTMS.)

Once the drivers are installed, either TX or RX can move full disks in either direction with CP/M-like commands from the prompt level.

For example, after the CP/M prompt "A>", RX types "RECEIVE B:" to set up the desired disk for reception. Then, TX need only type "TRANSMIT B:*.*" to move the entire contents of disk B on TX to an empty disk B on RX.

Data blocks of 128 characters plus

(continued on next page)

one line control character are automatically transmitted, so RX memory is no problem and manual file segmentation is avoided. Pure binary communication can be used for the highest speed transfers, and no secondary operations like LOAD are required. For example, a .COM file FF Hex will be transmitted as *one* byte, not two ASCII F's.

Further,—and nobody does more checking than this!—each character transmitted is checked for framing, overrun, parity, length, and so on; and, each block received is also checked for lost data, exceptions, time out, block CRC, block count, and other error types. Erroneous blocks are retransmitted and checked 200 times (transparently) before BSTAM gives up!

Handshaking, full duplex communication, and the BSTAM program at both ends makes this elegance possible. The result is silky smooth, no-nonsense file conversion without operator labor. Guess which program software houses and distributors use for file conversion?

You pay your money, and take your choice. The total cost of a BSTAM setup is $300, but if your time is worth anything, that's a bargain. Clearly, you will never receive a corrupted file.

BSTMS

The Byrom Software program BSTMS, designed to communicate with non-CP/M mainframes, is also of interest to many. If properly used, BSTMS can also communicate with other CP/M machines. Like *The Transporter*, BSTMS is a one-ended package. It resides on the CP/M machine and permits your microcomputer to emulate a TTY terminal for time-share service.

BSTMS employs the ESC key to flip between your machine and the mainframe (or other foreign computer). Normally, with BSTMS running, what you type goes to the mainframe in usual terminal fashion. In this case, you won't even know BSTMS is there. You can get back to your CP/M unit at any time without loss of mainframe connection by typing the ESC key and then the C key. That is, you can operate two machines at once if you need to. You can also get a complete help menu if you type ESC H, turn your CP/M printer on and off with ESC P, and so on with other ESC choices. BSTMS also lets you up-load and down-load text files from the mainframe. ESC R and ESC T are used for this purpose.

To receive a file, type whatever instructions the mainframe needs to LIST a file, then type LIST (or the equivalent), *but do not type* <RETURN> yet. Then on your machine Type ESCR.BSTMS will request the name you want for the received file. You name the disk and filename desired, [d:]Filename.TXT, and strike <RETURN>. BSTMS sets up your disk, then sends <RETURN> to the mainframe. Transmission starts.

BSTMS will limit received file size to what your working memory can hold and chop the excess during attempted disk operations—unless you install handshaking with the mainframe. X-ON/X-OFF protocol can be initiated automatically from the ESC menu, and should be used if your mainframe can recognize it. This protocol will permit automatic reception of long files with no operator intervention. Without handshaking, very long files must be split using a mainframe program.

Files transmitted to the mainframe by BSTMS are not so restricted and may be of any length. Mainframes usually delay the line-end <RETURN> echo if a delay is needed there, and BSTMS will hold up the next transmitted line until that echo is received. Alternatively, the X-ON/X-OFF protocol can be used.

Text files (ASCII) can be sent and received directly, but to handle binary (.COM) files, BSTMS (like SEND-HEX) requires an intermediate code conversion. Utility programs named DECOMPRS and COMPRES do this for you at your end using normal Hex instead of Intel Hex format as the code. (Most mainframes have Binary/ASCII conversion programs available at that end; if not, you must write one in a mainframe language if you want to up-load binary programs.) The DECOMPRES.COM expansion for binary files is exactly two (instead of three) bytes for one—a Hex FF is sent as two ASCII F's—so BSTMS is faster than SEND-HEX. Clearly, however, it is easier to have programs in listable form when you use BSTMS in either direction.

BSTMS error detection (like that of BSTAM) is compulsively complete; and, although BSTMS cannot demand retransmission in the event of erroneous data (the operator has to try again) you won't get errors in your stored file. We have never in three years of use seen a false stored byte using this product.

If two CP/M machines have BSTMS installed, they can communicate with each other with X-ON/X-OFF protocol. Whole disks cannot be transmitted automatically, but long single files may be passed across without user intervention.

Conclusion

Each of the three programs described has its own virtues:

BSTAM is the best for automatic file-to-file communication and disk conversion tasks, particularly for whole disks of intermixed file types. However, BSTAM cannot be used for terminal emulation.

BSTMS is an excellent terminal emulator and *text* file communicator, but lacks the fully automatic file transfer features of BSTAM and also requires intermediate steps for handling binary files.

The Transporter cannot compete with BSTAM or BSTMS in their own specialities—but it can do something neither can accomplish: directly reproduce a program, including itself, at a remote CP/M location *without* having any special additional programs at the destination and without any special installation at the transmitter.

For those reasons, we are glad we have all *three* programs on hand. ■

Feature

How About Some POWER

by Bob Kowitt

Power is a program that will do a great deal toward eliminating the errors and confusion arising when a novice uses CP/M. Actually, there is very little in POWER that has not been available to members of CPMUG, the CP/M User's Group. However, one would have to dig for it. Then, after finding it, the program to perform the function would have to be loaded and run.

POWER combines many of the intrinsic functions within CP/M, the most useful of the functions within the transient programs provided with CP/M and several enhancements that I have been looking for, some of which I found and others I didn't.

Pavel Breder, the author of POWER, has combined these functions into one program that calls sub-routines of a menu-driven package that, in my opinion, is well worth the $150 being charged by COMPUTING, the San Francisco distributor.

The general format of POWER permits menu-driven operations for most CP/M operations including copying, typing, renaming, erasing, and reclaiming erased files which helps to eliminate typing errors during these procedures and source/destination errors that have caused so many miscopied files when using PIP. In addition, there are enhanced direct disc accessing routines and enhancements to memory modification abilities that have been possible with DDT or SID.

I would like to go down the list of some of the commands to demontrate the added strength POWER gives the user of CP/M.

Most of the operations are driven by a screen directory. The command may be used with the CP/M asterisk format with one additional feature: the period may be left out. Therefore you can type ERA ** and all files will be handled.

COPY	This command by itself yields a directory of the disc with a unique number for each entry. To copy to another disc, you have the option of entering the file's temporary number, several numbers or an including form such as 5-12, meaning all files numbered 5 thru 12.
ERA	Erases files in the same manner as COPY copies but lets you know as each is being erased. This permits unerasing at once should you have made an error. The program can be modified (again, menu driven) to ask before each erasure, if you want to erase the particular file listed, just in case you had meant to type DIR or something else and type ERA by mistake.
REN	Renames files, again with the directory format. After selecting your numbers, it will step through your list, one by one, allowing you to rename your selected files. The operation is made easier by allowing you to enter an asterisk to retain either the existing program name or type.
RUN	Enter the file's temporary menu number and the appropriate .COM will be run. At present, this takes you out of POWER, but Clyde Steiner of Computing told me that they are working on a new version which will set up a submit file permitting POWER to reload upon completion of the COM file operation.
TYPE	This runs just like CP/M's type function with the exception that a queue can be established in the preceding manner to display a series of files. At any time during the display, the space bar will halt the display and single step by line until any other key is used. Paging can be disabled or enabled using the LOG function.
DUMP	Comes in four flavors: ASCII, formatted ASCII, hex and combined hex/ASCII.

So far, minor stuff!! However, let's get away from the CP/M standard operations now.

CHECK	Read a disc file and do a CHECKsum on it. Yes, CPMUG does have CRCK to do the job but it must be on the disc, loaded and run.
STAT	This does only one function of the STAT we all know and love. It lists the free and used space on the drives. However, we have other functions within POWER to do most of the others.
SETDIR,	SETSYS, SETRO and SETWR will set the attributes in the file called for to the directory, system, read only and read/write status as desired.
SPEED	Ever want to slow up the display on your terminal to reading speed? SPEED allows you to set the display rate from a level of 0 to 9. At level 9, you can go out for lunch while your screen displays.
SEARCH	is a goodie I have been trying to find for several years. It permits search-

(continued on next page)

ing thru memory for a group of bytes. Not only that, but you can use a wild card, '?', to find a group and display the space to either side. Suppose you want all locations within POWER itself that have the characters "SET". Type:

SEARCH 100 3200 "???SET???".

This will yield the locations in the POWER command table for SETDIR, SETSYS, SETRO and SETWR, and display them with the three bytes before and the three bytes after, should you want to modify the commands themselves.

DS	This is a modification of the DDT command 'S' used to substitute memory bytes. DS, however, not only displays the hex values at the memory address, but also the value in ASCII and binary. To alter the value, you don't need to know the hex for the ASCII characters to modify memory in the former example. Just type '.A' and you can enter your changes in ASCII being certain to allow a space between each entry.
GO	(address) will load a file to the address and run it at that address.
LOAD	(file) (address) will load the file but not run it.
SAVE	(file) (address) (sectors) will save a file at any memory address. First run SIZE (filename) to get the size in sectors. If the file is already on the disc and the size has not been altered, the sector size need not be entered. POWER will read the size from the existing file. Note that the file can be saved to disc from any memory location, not just 100H as with the CP/M intrinsic SAVE command.

Back in my days, BC, that is, before CP/M, I had a North Star system with the North Star DOS. I valued the ability to go to any physical sector of the disc and read or write directly to the disc. Admittedly, it is a dangerous procedure for the novice but, in the hands of a knowing technician, it is a valuable and powerful tool. Now, thanks to POWER, I can do this again and even better in CP/M.

I can now:

READ WRITE	to and from any track and sector into memory location, the number of sectors I want to manipulate.
READGR WRITEGR	any CP/M group to and from any memory location.

Ever want to transfer a group of files from one user area to another? First you must use DDT to move PIP to the new user area. Then invoke the user area and pip from the former area using the [Gn] parameter. POWER has the command:

XUSER	Now when you COPY, you will copy directly into the new area.
LOG	Would you like to set POWER to display your directory in two, three, four or whatever columns? Set it to ask you for verification before every move, verify all transfers, create a backup file if a file exists during a copy procedure. Suppose you are copying a long list of files and during transfer of a long file, you run out of disc space. POWER can cancel that transfer and try to transfer the next smaller file. All of these can be done from a menu by invoking this command.
TEST	does an extensive test of the disc read and write, saving any bad sectors found in a reserved SYSTEM file. After this is done, CP/M will not attempt to write to these sectors. The disc has effectively been reclaimed for further use.

This article does not attempt to cover all the facilities of POWER but enough to show that it is a powerful tool. Should you want to provide POWER to a client and modify it so he cannot do irretrievable damage (with READGR, for example), using the DS command to change the first character of any command in the table to '.', will prevent the use of the command and also its display by the '?' which lists all available commands.

During any procedure, hitting the ESCAPE key or Control-C will stop the procedure while a Control-K goes to the next operation in the procedure. In addition, there are four user definable functions available and jumps available to your own routines. There are three pages of customization notes and a READ.ME file on disc.

So much for the good news. Now the bad news. . . I found none; only one bug but several things I would like to have modified. First the bug: in the SEARCH mode, all locations are displayed plus two, 3115H and 3128H, which had no connection with what I was seeking. This came up on all searches so I learned to ignore it. Not a disabling bug.

Changes:

1. Re: RUN — This one is already in study. Come back to POWER after running another COM file.

2. Re: RECLAIM — I have been running FINDBAD from CPMUG. I modified it to display a mark at each group read. When using the TEST command, POWER looks at every sector within the group and consequently takes much longer than necessary to test the disc since it will continue to examine all 16 sectors in the group even if the first one is found to be unusable. These bad sectors are stored as a file on the disc named, "========.===" and therefore removed from access by CP/M on future writes. There are 243 groups on my disc. Since CP/M files everything in groups, finding a bad sector within a group should disallow the entire group. After the test, you are asked whether you want

to repair the disc. Inasmuch as I did not know how or what, I did not do so. . . nothing in the manual about it.

3. Re: READxx — When I use Ward Christensen's DU to look at a disc, it will stall when finding some bad sectors but after 10 reads, will let me know and display the sector for possible repair. I was unable to do this using POWER's READGR or READ function. Many of the problems from a BDOS error can be repaired but not within POWER. This ability is necessary for more advanced use of the CP/M system.

4. Re: RECLAIM — Unerasing previously erased file can present a problem if you are not aware of the operation of CP/M. When the RECLAIM procedure is called, ALL previously ERAsed files on the disc are displayed. By this time, some of the groups previously allocated to these old files could and probably have been assigned unless there were no disc writes since they were ERAsed. RECLAIM lists them one at a time and asks whether you want to UNERASE them. BE CAREFUL!! Only files erased since the last disc write may be UNERAsed safely.

5. Re: ALL FUNCTIONS — I would like the screen directory to print in alphabetic order. Finding the various files to operate on would be much easier. This one lack, while not serious enough to preclude buying POWER, has forced my use of PIP after running my SD (sorted directory program). I realize that there could be some additional complications in the coding since the numbers next to each entry probably refer to the files' location on the CP/M directory track. An index must be set up in memory to call for the program to be acted upon. Nevertheless, it is a necessary enhancement that should be provided at the earliest opportunity.

Qualitative Factors

	RATING
Documentation	
organization for learning	7
organization for reference	7
readability	7
includes all needed information	5
Ease of Use	
initial start-up	7
running other COM files	6
on board help	3
Error recovery	
ordinary mis-typed commands	7
catastrophic mis-typed commands	7

Ratings in this table are 1-7:

1 = clearly unacceptable for normal use

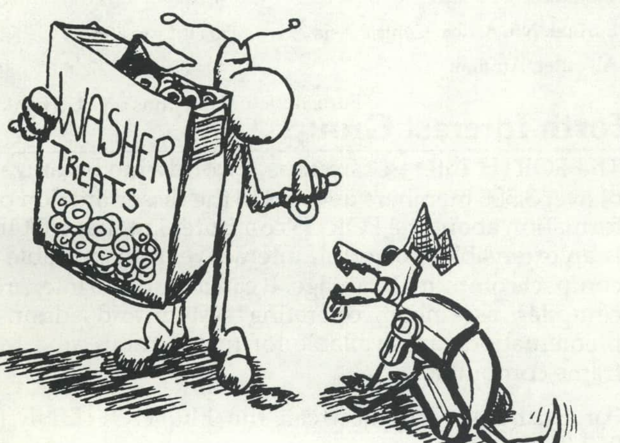
4 = good enough for most purposes

7 = excellent, powerful, or very easy depending upon the category.

POWER is available for $150 from:

COMPUTING
2519 Greenwich St.
San Francisco, CA 94123

and they offer a money-back guarantee. If you don't like it, return it. While I have seen this offer associated with a combined working and demo disc, this is something unheard of with working programs alone. ■



FORTH-79

Version 2 For Z-80, CP/M (1.4 & 2.x), & NorthStar DOS Users

The complete professional software system, that meets ALL provisions of the FORTH—79 Standard (adopted Oct. 1980). Compare the many advanced features of FORTH—79 with the FORTH you are now using, or plan to buy!

FEATURES	OURS	OTHERS
79-Standard system gives source portability.	YES	
Professionally written tutorial & user manual.	200 PG.	
Screen editor with user-definable controls.	YES	
Macro-assembler with local labels.	YES	
Virtual memory.	YES	
BDOS, BIOS & console control functions (CP/M).	YES	
FORTH screen files use standard resident file format.	YES	
Double-number Standard & String extensions.	YES	
Upper/lower case keyboard input.	YES	
APPLE II/II+ version also available.	YES	
Affordable!	$99.95	
Low cost enhancement options; Floating-point mathematics	YES	
Tutorial reference manual		
50 functions (AM9511 compatible format)		
Hi-Res turtle-graphics (NoStar Adv. only)	YES	

FORTH-79 V.2 $99.95
ENHANCEMENT PACKAGE FOR V.2:
Floating point $ 49.95
COMBINATION PACKAGE (Base & Floating point) $139.95
(advantage users add $49.95 for Hi-Res)
(CA. res. add 6% tax; COD & dealer inquiries welcome)

MicroMotion
12077 Wilshire Blvd. # 506
L.A., CA 90025 (213) 821-4340
Specify APPLE, CP/M or Northstar
Dealer inquiries invited.

Users Group Corner

We are starting a new columns on User Groups. If you belong to a Users Group or Computer Club, please send us information on your club and we will publish it in our Users Group column. Please include your groups address, dues, and what your club is interested in.

Pascal/MT + Users Group

The Pascal/MT + family of compilers has been adapted for use on different operating systems, such as CP/M, MP/M, CDOS, MDOS, M/OS or RMX-86, as well as a large number of different computers. AT present, the processors i8080/i8085, Z80 and i8086/i8088 are supported. The compiler for MC-68000 based systems (16-bit) will become available during 1983. Current indications are that Pascal/MT + and the associated programming package, SPP (Speed Programming Package) will soon become the system of choice for business, technical and scientific applications, especially since REALS can be represented in both floating point and BCD numbers.

The Pascal/MT + Users Group, MTPUG, was formed for the purpose of encouraging the use of Pascal and in providing the users of this rather special program product with the opportunity to communicate woth others who speak their language. All communications will be published in a Newsletter which will include reviews and articles on software packages, data base languages, programs submitted by the members, and graphics software. Attention will be directed to programs and products in which Pascal/MT + is the "host" language.

MTPUG is a non-profit organization and is not associated with Digital Research, Inc. Nevertheless, Digital Research provides the editor with information on bugs and forthcoming events for inclusion in the Newsletter. Since its beginning, they have supported and encouraged the Editor of MTPUG in many ways.

The MTPUG Newsletter is published four times a year. Each issue consists of about 20 pages. It contains experiences reported by the members, tutorials, reviews of new products or articles and books, small Pascal routines (procedures, functions and programs), letters from manufacturers and members, bug reports, fixes and news about persons and products. The MTPUG Newsletter is composed of material submitted by the members and by the News Editors. The cost of the Newsletter is included in the annual Membership fee which is presented below.

The MTPUG Newsletter only publishes Pascal programs if they are short, since we have never found anyone who was willing to type long ones from printed material. Therefore, the exchange of programs between members of MTPUG will be on "MTPUG Program Disks." The index of each new disk will be published in the Newsletter. The index of all available MTPUG program disks (7 disks at present) can be requested from your Editor. A $1 donation and self-addressed stamped envelope is appreciated. The program disks are availabe on standard 8″ (soft-sectored SS/SD IBM 3740 format) from your Editors. Consult the most recent Newsletter for the availability of 5.25″ floppies in a variety of formats. The material on a single 8″ diskette may be contained on 1, 2 or 3- 5.25″ diskettes. The cost for the standard 8″ program disk is $10 for orders from the US, Canada and Mexico and $13 (air mail) elsewhere. Members who submit Pacal programs or articles for the Newsletter on diskette will have their disk returned with either a copy of the most recent program disk or the one of their choice. Since postage is a major factor in our budget, it is essential that all persons who send disks include return postage.

All applications for membership must include your name and full address (including postal code) in exactly the same way you wish it to appear on your MTPUG Newsletter. if at all possible, limit the number of lines in the address to no more than four (4) lines. If you reside in Europe, North Africa, Western or Central Asia then write to Guenter Musstopf at the address below. Otherwise contact henry Lucas.

MTPUG	MTPUG Europe
Pascal/MT Users Group	Pascal/MT Users Group
Henry Lucas	Guenter Musstopf
P.O. Bos 192	Schimmelmannstr, 37a
Westmont, IL 60559	D-2070 Ahrensburg
(312) 986-1550	West Germany
After 7PM and NOT (Tues. or Thurs.)	Phone 04102/56629

Please indicate if NEW or RENEWAL subscription (Circle one).

Dues:	**1-year**	**2-year**	**3-year**
United States	$ 7.00	$13.00	$20.00
Canada and Mexico	$ 8.00	$15.50	$23.00
Europe, No. Africa, Central Asia	30 DM	59 DM	88 DM
All other, Air mail	$14.00	$27.50	$41.00

Eurocheck, or US Funds on a US bank only.

Forth Interest Group

The FORTH Interest Group is a worldwide organization of over 3,500 members devoted to the dissemination of information about the FORTH computer language. FORTH is an extensible, powerful, interactive, transportable and compact computer language. It can include an interpreter, compiler, assembler, operating system and editor. Implementations are available for micro, mini, amd mainframe computers.

For further information call the FIG HOT LINE (415) 962-8653.

CBasic Users Group

PO Box 2365
Sherman, TX 75090

The philosophy of CBASIC ASSOCIATES is to offer any economically feasible services of interest to its members. The services available and the resources allocated to provide them will be given the following priorities:

1) Business Applications. (Including personal finance)
2) Business Education.

(continued on page 36)

Letter To The Editor

Dear Editor:

Regarding Robert P. VanNatta's article of the March 1983 issue of *Lifelines* entitled, "Get a Better Performance Out of CB-80," I found a bug in his sample program as well as some rather elementary mistakes.

The bug appears when he translated his old CBasic program statement (Figure 3A, page 36) which checked to see if the variable "CHARACTER%" was "less than or equal to 65." This caused trouble with the capitalization of the letter "A" in names. This fix, as shown in page 1 of the enclosed listings, is to change the statement to check to see if the variable "CHARACTER%" is simply "less than 65."

Since the article was about better performance, and hence to some degree about optimization of both CB80 code and of the resulting 8080 code, the following three modifications can make quite a difference in a completed program.

The first sample shows how easy it was to remove a variable which was not needed. This resulted in a 9.7% reduction in code size and in a whopping 30.0% reduction in the data area size. This is what his article was talking about on page 5 under the subheading "Use Complex Statements."

The next example, found in the enclosed listings, shows no large savings in either the code or data area but is a practice that even beginning programmers should follow. That is the use of CB80's built in handling of true and false instead of using 1 and 0 to do the same thing.

If we go to the last listing you can see what the use of CB80's true and false can really do. Instead of having a complex "IF" statement, we can reduce it to just one simple statement. It will assign CB80 true (−1) if it is equal and CB80 false (0) if it isn't equal. The result of this change is a 3.8% drop in the code size.

If we put all of these changes together we have not only eliminated needless lines of code, but have reduced the code size by 13.6% and the data area size 30.0%! That is quite a savings for only 23 lines of original CB80 code!

Your magazine is very informative and enjoyable. I look forward to reading it every month.

Randy Kimbro
Software Development
Star Software

```
Rem     -- Changed:
Rem        Character% <=65
Rem
Rem     -- To:
Rem        Character% <65

Def Lowcase$ (x2$)

        String x2$, Lowcase$, Scratch$, Segment$
        Integer Last.character%, Character%,
          Pointer%, Flag%

        Scratch$=Left$(x2$, Pointer%, 1)
        Last.character%=Len(x2$)
        Flag%=0

        For Pointer%=2 to Last.character%

          Segment$=Mid$(x2$, Pointer%, 1)
          Character%=Asc (Segment$)

          If Character% <65 then \
            If Character%=32 then \
              Flag%=1 \
            Else \
              Flag%=0 \
          Else \
            If Flag%=1 then \
              Flag%=0 \
            Else \
              If Character% <=90 then \
                Segment$=Chr$(Character%+32)

          Scratch$=Scratch$+Segment$

        Next Pointer%

        Lowcase$=Scratch$

Fend

Rem     -- Modification Number 1

Rem     -- Removed variable 'SEGMENT$'

Def Lowcase$ (x2$)

        String x2$, Lowcase$, Scratch$
        Integer Last.character%, Character%,
          Pointer%, Flag%

        Scratch$=Left$(x2$, 1)
        Last.character%=Len(x2$)
        Flag%=0

        For Pointer%=2 to Last.character%

          Character%=Asc( Mid$( x2$, Pointer%,1 ) )

          If Character% <65 then \
            If Character%=32 then \
              Flag%=1 \
            Else \
              Flag%=0 \
          Else \
            If Flag%=1 then \
              Flag%=0 \
            Else \
              If Character% <=90 then \
```

(continued on next page)

```
            Character% = Character% + 32

         Scratch$ = Scratch$ + Chr$(Character%)

      Next Pointer%

      Lowcase$ = Scratch$

Fend

Rem     Modification Number 2

Rem     -- Change FLAG% to use CB80 false (0) and
           true (-) instead of 0 and 1
Rem     -- Changed:
Rem        Flag% = 1
Rem
Rem     -- To:
Rem        Flag% = -1
Rem
Rem     -- And changed:
Rem        If Flag% = 1 then \
Rem
Rem     -- To:
Rem        Fi Flag% then \

Def Lowcase$ (x2$)

      String x2$, Lowcase$, Scratch$
      Integer Last.character%, Character%,
        Pointer%, Flag%
      Scratch$ = Left$(x2$, 1)
      Last.character% = Len(x2$)
      Flag% = 0

      For Pointer% = 2 to Last.character%

         Character% = Asc( Mid$( x2$, Pointer%, 1 ) )

         If Character% <65 then \
            If Character% = 32 then \
               Flag% = -1 \
            Else \
               Flag% = 0 \
         Else \
            If Flag% then \
               Flag% = 0 \
            Else \
               If Character% < = 90 then \
                  Character% = Character% +32

         Scratch$ = Scratch$ = + Chr$(Character%)

      Next Pointer%

      Lowcase$ = Scratch$

Fend


Rem     -- Modification Number 3

Rem     -- Changed:
Rem        If Character% = 32 then \
Rem           Flag% = 1 \
Rem        Else \
Rem           Flag% = 0
Rem
Rem     --To:
Rem        Flag% = (Character% = 32)
```

```
Def Lowcase$ (x2$)

      String x2$, Lowcase$, Scratch$
      Integer Last.character%, Character%, Pointer%,
        Flag%

      Scratch$ = Left$(x2$, 1)
      Last.character% = Len(x2$)
      Flag% = 0

      For Pointer% = 2 to Last.character%

         Character% = Asc( Mid$( x2$, Pointer%, 1 ) )

         If Character% <65 then \
            % = (Character% = 32) \
         Else \
            If Flag% then \
               Flag% = 0 \
            Else \
               If Character% < = 90 then \
               Character% = Character% = + 32

         Scratch$ = Scratch$ + Chr$(Character%)

      Next Pointer%

      Lowcase$ = Scratch$

Fend
```

AUTHOR'S RESPONSE

This response confirms the main point of my article which was intended to suggest that careful optimization of any given code segment can usually result and attend a 30% reduction in the amount of code.

The suggestions of Mr. Kimbro are ones which for the most part I made in the article and failed to follow through in the example, although, modification #3 is a wrinkle which I had never thought of.

Robert P. VanNatta

Attention Subscribers

To *Lifelines* Magazine, its staff, and all its subscribers:

It has come to my attention that issues of *Lifelines* Magazine have been mailed out improperly. This resulted in a total misunderstanding on our part. In no way should *Lifelines* Magazine be held accountable for this; we accept complete responsibility. You have my personal guarantee this matter will be watched very closely to insure it will not happen again.

I would like to thank *Lifelines* Magazine for being so understanding in this matter. Thank you.

Al Basile
President
Complete Direct Mail Service

Software Notes

New Versions

MicroSpell for IBM-PC
Lifeboat Associates
1651 Third Ave.
New York, NY 10028

Version—Interim before 5V

This version includes the following: A reverse video patch has been incorporated into the context printing routing. (Thanks go to Clyde Washburn of Earth Terminals for this patch.) A warning is now issued whenever the correction or replacement of a word results in its size changing. This is an indication that a soft hyphen might have to be replaced, or the text reformatted. A bug in V4.5 which prevented the auto-replace feature for the "R" and 'C" commands from working has been fixed. The disk I/O routines have been rewritten. This greatly speeds up the dictionary loading process.

INFORMA X
Abacus Data Inc.
1920 San Marco Blvd.
Jacksonville, FL 32207

This information management system features many new features including: cross-file data sharing, multiple file reporting, multiple screens per record, far broader application capabilities, increased speed and security and greater data file capacity. INFORMA can now run on any 8- or 16-bit systems with Z-80, 8085, 8086 and 8088 processors, operating CP/M-80 or CP/M-86.

OTHER NEW VERSIONS

Lattice C (CP/M-86) 1.03/1.21
C-Food Smorgasbord 1.2
ASCOM-80 2.24
PRO-MAN 5.02
WordStar & Mail Merge (CP/M-86) 2.24
dBASE-II/86 & PC-3.0
BDS C 1.50a
The CP/M Workshop 1.02
UNICALC/PC—3.0
MULTIPLAN—1.05
TIM-III 3.22 (CP/M-80)
QUICKCODE(IBM PC)
BSTAM-86 VICTOR (CP/M-86, MS-DOS) 4.6
PAS-3 Medical—1.72

New Products

FOOT ↑CONTROL

Digital Servo Sytems
PO Box 1248
San Luis Obispo, CA 93406

This conversion for use with word processing programs kit gives the operator an additonal control key that is located on the floor and operated by foot. Generating a control character is a simple matter of pressing the foot switch and then typing the desired character key. FOOT CONTROL provides improved editing speed, greater editing accuracy, and highly reduced operator stress and strain. It also can be used as a duplicate ESCape key for software requiring large amounts of ESC sequences.

Price: $39.50

FilePlan

Chang Labs
5300 Stevens Creek Blvd. Suite 200
San Jose, CA 95129

This electronic filing system has been designed for ease of data entry by the end-user. The exclusive worksheet format allows multiple records to be viewed simultaneously and existing records to provide examples for data entry. Also, FilePlan has special help commands, an on-screen menu, data validity (attribute verification) and user prompts. Records can contain between 128 and 2048 characters with up to 32 variable-length fields with up to 99 characters each. Binary-Tree indexing keeps records in the proper sequence, automatically updating the sequence as records are added, deleted or modified.

Requirements: CP/M system 8- or 16-bit versions, minimum of 64K, and two drives with at least 150K storage per disk.

Price: $295

BIBLIOGRAPHY

Software Digital Marketing
2670 Cherry Lane
Walnut Creek, CA 94596

This program compares citations in a manuscript with the entries in a card catalog and constructs a bibliography of all entries cited. Entries are added to the catalog using a text-editor. Each catalog entry has a key name (for example, author and year of publication), followed by biblio graphic information such as the author's name, the title,

(continued on next page)

journal, publisher and annotations. The entries may be of any length and format. BIBLIOGRAPHY can also copy entries from the catalog to footnotes in the manuscript, or replace citations in the manuscript with numbers corresponding to the order in which the works appear in the bibliography. It also constructs a bibliography of all works excited—alphabetized or numbered, with annotations included or excluded.

Requirements: CP/M-80 or CP/M86 or IBM-PC DOS, most wordprocessors (WordStar, Spellbinder, PeachText and SuperWriter.

Price: $125

TOTAL MATERIALS

TCS Software Inc.
3209 Fondren Rd.
Houston, TX 77063

This materials/parts explosion and tracking package for OEM assemblers and manufacturers works together with an enhanced version of TCS TOTAL INVENTORY to provide a complete inventory system with the flexibility of a built-in data base manager. It produces a Bill of Materials for every product, prints production scheduling, mate rials planning, and production cost analysis reports, and lists reference numbers for engineering drawings and parts lists numbers.

Requirements: CP/M-80 2.2, 48K TPA, 100K of Disk Storage.

Price: N/A

TRANSFORM

MasterComputing Inc.
PO Box 17442
Greenville, SC 29606

This structured program translator allows you to write your programs in a structured Microsoft BASIC, then automatically translates those programs into "real" MBASIC code. You can even emulate PROCEDURE calls with an incredibly flexible INCLUDE facility. You can create generic subroutines that can be merged into your program during translation. Variable names are changed automatically. You can write structured programs without line numbers, with labels, and INCLUDE your library programs.

Requirements: CP/M-80

Price: $29.95

DPATCH

Systems Plus Inc.
1120 San Antonio Rd.
Palo Alto, CA 94303

This utility recovers data from damaged hard or floppy disk and flags I/O errors without destroying user data. DPATCH also recovers files accidentally erased, displays (and alters) any sector on a disk, and can provide a printed log of each session. Designed with the business computer user in mind, DPATCH operates in full screen mode.

Requirements: CP/M or MP/M

Price: $195.00

8086 FORTRAN Compiler

SuperSoft
PO Box 1628
Champaign, IL 61820

This FORTRAN compiler is full ANSI 66 standard with extensions. Hardware floating point support for the 8087 coprocessor is available as an option, as is a RATFOR preprocessor. It is also available for Z80 based microcomputers running the CP/M-80 operating system.

Requirements: CP/M-86 or MSDOS or IBM PC DOS.

Price: N/A

RXWRITER

Hall Design
250 Maple St.
Wilmette, IL 60091

This prescription writing program for physicians permits preparing prescriptions six at a time using a physician's individual list of drugs. Prescriptions are printed in duplicate, one for the patient another for the clinical record. A disk file is created which contains the name, date, diagnosis, and prescription abbreviations. This can be searched to find, for example, all the patients who were given a specific drug. Included in the system are utilities for adding, deleting, or modifying drugs in the drug file and a help routine which looks up the information in the physician's list of drugs.

Requirements: CP/M-80, 48K

Available for: 8" SD, Apple II, Kaypro, Xerox 800, H-89-Z89, Osborne

Price: $50.00

Bugs

RUN/Z CP/M 8080/Z80 version only
Rev. -1.02
System/Z Inc.
PO Box 11
Richton Park, IL 60471

Two minor bugs are present in RUN/Z. They are: 1. minor inaccuracies in trig, log, and exponentiation. 2. slow execution of certain exponentiation functions.

They may be easily corrected with the PATCH utility program provided on your master disk.
Please follow this procedure precisely:
1. As PATCH is a compiled BASIC/Z program, execute it by typing the command: RUN/Z PATCH
2. Respond with A through P to indicate the name of the drive on which the program to be "patched" resides.
3. PATCH will display a line number, enclosed in angle brackets (e.g. <001>). Enter the appropriate four numbers, each followed by pressing <RETURN>, as detailed below:

```
<001>   154 002 034 017
<002>   154 002 035 088
<003>   154 002 093 067
```

4. Finally, press just <RETURN>. PATCH will allow for correction of any entry errors prior to the final write to your disk.

Macro of the Month

Todd Katz

One of the nicest things about PMATE macros is that it is usually easier to customize an already existing PMATE macro or *even* improve one, than it is to write one from scratch.

Thus the first macro this month is an enhancement of a SEARCH macro developed by Mike Olfe to allow a search-and-replace routine which gives you the option of implementing the change or not. With PMATE's standard C*n*↑*nx* command you are presented with an accomplished fact.

This macro has now been "enhanced" to give a little more screen friendliness by clearing out four or five lines at the top of the screen while the *string to be changed* and the *string to change to* are displayed. The routine should also give inspiration if you are interested in building menus that are longer than the 80 columns on the command line. The STRING macro, also listed, must be stored in PMATE's permanent macro area in order for SEARCH to work.

The second macro, TYPEWRIT.MAC enhances one of Mike Aaronson's sample macros in the current PMATE manual. As the name implies, TYPEWRIT.MAC turns your computer into an overly-expensive typewriter. The enhancement portion of the macro permits you to edit the line you are typing as long as you have not hit a return, and you can also use the DEL key to back-up and erase through the line before the RETURN sends the entire line to the printer. This feature duplicates the "magic line" feature available on $1,700 Xerox electronic typewriters, among others.

HIGHBIT.MAC will display any special graphics or other characters at your terminal and the decimal number of the character. Although the current version of PMATE will not allow you to save these high bit characters *per se*, they can be utilized with the insert command *ni*. I've found these characters useful in CP/M-80 for building more attractive displays, menus, prompts and flags for my text editing macros.

On the development front, the boys at Phoenix Software are engaged in a massive revision of PMATE. Most of the efforts are being directed at the 16-bit version but a trickle down effect for the 8-bit world can be expected. In addition to the revision, PMATE seems to be having children. One version, not yet available, is reportedly VERY friendly, with menus no less. Another is nothing less than a full blown typesetting machine. No release date for either product has been announced.

It would not do not to take note of the departure of Mike Olfe from this column. Mike has moved down to the Washington, D.C. area to help the world's first electric company master the art of microcomputers. Mike will be missed but he has promised to offer us sage advice and barbed criticism and — hopefully — a macro now and then.

```
;search macro
4[i
$]                        ; insert 4 blank lines
-2l                       ; move back two lines
t                         ; tag (remember) current position
.iSearch String: $        ; insert prompt
–                         ; catch characters between tag and
                          ; end of search string
b9c                       ; copy search string to buffer 9
-l                        ; move back one line
t.iReplace String: $–b8c ; move replace string to buffer 8
-l4k                      ; move back one line and delete
                          ; search/replace window
                          ; now the search string is loaded
                          ; into buffer 9 and the replace string
                          ; into buffer 8
a                         ; move to top of file
[                         ; begin iteration level No. 1
e                         ; suppress error message if string
                          ; not found
s@9$                      ; search for the string of characters
                          ; stored in buffer 9
@e{%}                     ; if the error flag is set
                          ; (string not found) exit macro
g<space> to replace, any other key to skip $
                          ; insert prompt in command line
@k=32[                    ; if a space (32 decimal) in typed
-c@9$@8$]                 ; change string that matches the
                          ; contents of buffer 9 to the string
                          ; contained in buffer 8
                          ; end iteration No. 1
78ff$0qv$$

;high bit tester
80f                       ; 80 column format
126v0                     ; set variable 0 to 126
[                         ; begin iteration
@0=255{%}                 ; if variable 0 equals 255 exit
va0                       ; increment variable 0
@0i                       ; insert character equal to
                          ; variable on the terminal screen
@0\                       ; insert value of variable 0 next
32i                       ; insert a space
qr                        ; redraw the screen
]                         ; end iteration

[                         ; begin iteration No. 1
gI'm an overgrown typewriter$
                          ; command line prompt
@k=127{                   ; if key struck is a DELete
-md↑}                     ; move back, delete key just
                          entered
                          ; go to beginning of iteration
@k=13[                    ; IF key struck is a return begin
                          ;iteration
```

(continued on page 34)

Estate Tax Plan

For those of you planning to die rich, Aardvark Software has just the program for you. It is called ESTATE TAX PLAN. The version reviewed here is version 1.3. The program is not a tax return calculation program, but rather a planning tool designed to assist estate planning professionals project potential Federal estate tax liability.

The interest in this program is not that it does something new or unique. It doesn't. Its role is strictly utilitarian. You, the estate planner, collect the usual asset and liability information together with the necessary biographical information. Then, instead of spending half a day with a scratch pad and a pocket calculator, you simply enter this information, wait a minute or two, and either display the results on the screen or drive them to the printer.

Features

Estate Tax Plan takes into account the revisions made in the federal tax law by the Economic Recovery Tax Act of 1981. At risk of boring everyone, it should also be noted that this program also makes the side calculations necessary to project the tax benefits available under IRC Sec. 6166 and Sec. 303.

The version reviewed was delivered in the 8-inch CP/M format and occupied 240K of disk space (Apple and IBM version are reportedly available). It consists of three files. There is a 100K data file, a 138K program overlay, and a 2K menu. No disk workspace is required, and if these files can somehow be fitted on your drives, the program will work. (They need not be on the same drive). If you are wondering how a 138K program overlay works on a 64K Z-80, system you will have to ask someone other than this writer. But that is the way it is.

Actually, all available evidence suggests that the ESTATE TAX PLAN is written in PASCAL and that the magic is accomplished by loading portions of the overlay as needed.

As far as installation goes, there should be no problem if your terminal is 24 x 80. The terminal installation menu is longer than average, and the appendix contains an additional two-page list of emulations. There is also provision for the "Other" brand of terminal.

The Estate Tax Plan is completely menu driven. In fact, it consists of over 89 submenus and several main menus. The manual is a pleasantly bound three-ring binder with tab dividers. It shouldn't be regarded as a model for clarity, but it is certainly adequate for the task. The manual does provide some examples, and the program itself is so straight froward that this writer was satisfied that he had mastered its use after no more than a couple hours of fiddling with it.

The error handling routines in this program are the best that I have ever seen anywhere. This writer is of the opinion that that good quality programs simply don't crash—no matter what the user does. Within the limits imposed by CP/M 2.2, Estate Tax Plan meets this test. The error trapping is so strong that it will generally regain control and reload the main menu even if the user resorts to such devious behavior as removing a necessary disk midway through the program execution. Every keystroke of keyboard input is validated a key at a time. If it is illegal, the terminal bell will ring and the keystroke will be rejected. If I were to criticize the error trapping at all, it would be to suggest that some provision be made to communicate the error other than by use of the bell. There are quite a few people around who have terminals that ignore bell command (CHR$(7)). The only real bug that I could find in the program was in the date input routine. It expected four numbers consisting of the month and the year such as 0683. Unfortunately it would accept 1983 for an answer. This error would eventually cause the calculation procedure to abort.

Summary

This program is not a substitute for a lawyer or an accountant, but rather a tool intended for use by such a professional to remove the drudgery of the extensive amount of number crunching required to prepare an estate plan. This limited task is accomplished very credibly and the Estate Tax Plan is not suitable for actually preparing a Federal Estate Tax return, rather it is strictly a planning tool. Similarly it contains no provision for dealing with state death tax liabilities. With Federal estate tax exemptions soon destined to exceed $500,000, it is doubtful if this program will ever attain the popularity of Pac-Man. However, the overall quality of this program from the standpoint of structure, organization and design is simply outstanding. ■

```
13i-l          ; insert return, move back one line
1xt            ; print one line
][             ; ELSE
@ki            ; insert key into text
]              ; end iteration No. 2
]              ; end iteration No. 1
I              ; macro I
 qa            ; limit number of string arguments
               ; to one
[              ; begin interation No. 1
ga$            ; get string from keyboard ??
@k=127[-d↑]    ; if key hit is a DEL delete
                 character
               ; previously
               ; entered and return to beginning
               ; of iteration No. 1
[              ; begin iteration No. 2
@k=13[%]       ; if keyboard character equals
               ; carriage return
               ; exit this macro
@ki            ; (else) insert character
]              ; end second iteration
]              ; end first iteration
```

Software Notes

Three recent *Lifelines* articles came together fro me to form the nucleus of this sofware note. I had been reading Ward Christensen's 8080 Assembler Tutorial series and was looking for a good assembly language programming project. Until now, I had found MBASIC to be adequate for the application programs I moonlight to the small business sector. Besides, since I didn't have a relocatable macro assembler, I couldn't easily link an MBASIC program to even the simplest assembler subroutine. But Gregory Knott's Pseudo-Relocatable Subroutines article (*Lifelines*, 6/82) proved to the link I needed. Now I could link my BASIC programs to assembler subroutines to achieve what, for me, was the best of both worlds—the speed of the BASIC language for program development and the nitty-gritty of machine language for execution speed and functionality in certain areas not obtainable n BASIC. Michal Karas's CP/M BDOS series provided the assembler link to CP/M.

With the three articles in hand, I jumped into my first MBASIC program containing a subroutine CALL. This subroutine would give me direct control of what the user keyed as input to my BASIC program. Too often, the input screen templates of my application programs were "cracked" by a control-C or long input that over-typed an adjacent field before the ENTER was hit. Now I had the tools to implement true protected/unprotected fields on a dumb terminal because I could capture each input code as it was typed. What characters were echoed would also be controlled by this subroutine.

the first step was to recreate the LOADSUB program described in Gregory Knott's article. This program executes in the Transient Program Area (TPA) and relocates part of itself to igh memory (A000H). It then relocates my subroutine (DIRCTIO.COM) into higher memory (B100H) and finally loads and executes MBASIC in the TPA. MBASIC has options that 1) reserve memory from B100H to FDOS and 2) load and execute my MBASIC program (DIRCTIO.BAS). 41-42 substituting my subroutine name (DIRCTIO.COM) for PRINTHI.COM and my MBASIC program name (DIRCTIO) for PRINTEST. The COM file, LOADSUB.COM, was created with Digital Research's ASM and LOAD.

Now I was ready to write the assembler subroutine, figure 1. It would be CALLed for every input char of a user response to the BASIC application program prompt. It would return a printable character to be appended to the response string, a flag indicating a certain function (i.e. backspace or ENTER) or nothing in the case of unwanted, unprintable characters. I chose to use the CP/M direct user interface to console (BDOS function 6) rather than input from console keyboard (BDOS function1). With function 6, I could control the echo to the keyboard. Therefore I setup a BDOS call with the E register equal OFFH (indicating input). This input character is returned in A. If no character has been entered, A equals zero. (A timer could be added to this wait loop which would result in a time-out indication being returned to the BASIC application program if a character was not struck within a pre-set interval.) With the user response in the A register, I now check to see if it is a printable character (20H<=A<=7EH). If so, I'm ready to return it to the application in the string pointed to by CALL argument 1. If not, CALL argument 2's string is setup—1 to indicate ENTER was keyed, 2 for backspace, and N for any other non-printable character. While MBASIC provides the bridge to the assembly subroutine with its CALL statement (parameters appearing in registers when control is received by the subroutine), we must build out own interface to return the operator keyed character or flag indicator to MBASIC. We have the address of the string we want to setup in either CALL argument 1 or 2. Argument a (register pair HL) points to the string where a printable character will be stored while argument 2 (register pair DE) points to the non-printable character indicator string area. A map of the internal MBASIC string storage is in order here.

If string AB$ = '1' then:

Address	HEX Content	Comment
N-3	42	
N-2	41	String name (AB)
N-1	00	
N	01	Length of string
N+1	low	Address of 1st byte of string
N+2	high	

and CALL subroutine-name (AB$) will cause the address N to be placed in register pair HL. So to return the desired string to MBASIC, the assembler subroutine bumps the corresponding register pair (HL or DE) by one and stores the data (printable char or indicator flag) at the address contained in the next two bytes. The RET instruction gets us back to the MBASIC statement following the CALL. ASM and LOAD are used to create DIRCTIO,COM.

Finally, a portion of the MBASIC program, figure 2, to illustrate use of the assembler subroutine. Line 310 defines a function for cursor positioning. Line 350 defines the location of the assembler subroutine. Now we want to solicit NAME from the user with "Name:______________________" as a prompt. Line 410 shows the prompt starts at position 5 of line 10. The cursor is positioned to column 11 of that line to await input. The WHILE/WEND structure shows a CALL to our assembler subroutine, DIRCTIO. Upon return, either a printable character will be found in CONSOLE.CHR$ or a flag indicator will be set in NON.PRINT.FLAG$. In line 465 a check is made for backspace. If found and it wasn't entered at the first position of the input area (column 11)

(continued on next page)

we decrement the column indicator (CURRENT.COLUMN%), drop the last character of our built-up input string (REPLY$) and replace that character on the screen with our original prompt character, the underscore. (A backspace entered at the first position of the input area is ignored but could be programmed to reposition to the previous input field.) Line 470 causes a printable character to be added to the built-up input string and screen display. The current column indicator is also incremented. Line 480 WENDs us back to line 440. The loop is repeated until either the ENTER key is struck (NON.PRINT.FLAG$=REPLY.END%+1), which in this example is 12 characters. At line 495, the user input is safely tucked away in REPLY$.

As Greg-s article shows, the program is executed by:

A>LOADSUB

requiring the following files on drive A:

MBASIC.COM
DIRCTIO.COM
DIRCTIO.BAS
LOADSUB.COM

BASIC and assembler, coexisting and communicating—each doing what it does best. Thanks Ward, Greg and Mike.

```
;        DIRCTIO.ASM
;
;        Assembly language subroutine to get 1 char at a time
;        from the console in DIRECT I/O mode. Returns the character
;        to the calling MBASIC program by storing it in the 1 char
;        string pointed to be incoming CALL parameter 1, or stores a
;        flag indicator in the string pointed to by CALL parameter 2 if
;        non-printable char was typed.
;
bdos         equ 005h        ; CP/M bdos entry vector
direct$io    equ 0006h       ; bdos direct I/O function number
input$fla-
g            equ 0FFh        ; direct I/O input indicator
;
start        org    0100h    ; CP/M TPA base address
loc          equ    0b100h   ; subroutine starting address
z            equ    loc-$    ; TPA to LOC relocation constant
;
             push   h        ; save MBASIC CALL parameter 1
                             ; (printable char string addr)
             push   d        ; save MBASIC CALL parameter 2
                             ; (nonprintable flag string addr)
wait:                        ;
             mvi    e,input$flag ; set direct I/O function for input
             mvi    c,direct$io  ; bdos function number
             call   bdos     ; obtain 1 char or status in A reg
             cpi    0        ; if (A)=0 no char was ready
             jz     wait+z   ; continue waiting
                             ; 20h<=char<=7Eh?
             cpi    07Fh     ;
             jnc    no$prt+z ; no
             cpi    020h     ;
             jc     no$prt+  ; no
                             ; yes, a printable char
             inx    sp       ; CALL parameter 2 not needed
             inx    sp
             pop    h        ; restore MBASIC CALL parameter 1
             inx    h        ; point to string addr (LO)
             mov    c,m
             inx    h        ; point to string addr (HI)
             stax   b        ; store input char in MBASIC   program
             ret             return to MBASIC program
;
no$prt:                      ;
             cpi    0dh      ; char=CR?
             jz     char$cr+z ; yes
                             ; no
             cpi    08h      ; char= BS?
             jz     char$bs+z ; yes
                             ; no
             mvi    a,'N'    ; non-printable char, bypass
             jmp    return+z ;
;
char$cr:                     ;
             mvi    a,'1'    ; load CR indicator flag
             jmp    return+z
char$bs:
             mvi    a,'2'    ; load BS indicator flag
             jmp    return+z
;
return:
             pop    h        ; restore MBASIC CALL parameter 2
             inx    sp       ; CALL parameter 1 not needed
             inx    sp
             inx    h        ; point to string addr (LO)
             mov    c,m
             inx    h        ; point to string addr (HI)
             mv     b,m
             stax   b        ; store flag indicator in MBASIC prog
             ret             ; return to MBASIC program
             end    start
```

```
300 REM    DIRCTIO.BAS
303 REM
305 REM    MBASIC program linked to assembler subroutine
306        DIRCTIO.COM to filter user keyed input
307 REM
310 DEF FN XY$(X,Y)=CHR$(27)+CHR$(89)+CHR$(Y+31)
        +CHR$(X+31):
        REM cursor positioning function for UNIVAC UTS-42
350 DIRCTIO%=&HB100: REM addr of assembly subroutine to get
        1 char
400 REM solicit NAME
410 ROW%=10: PROMTP%=5:
        PROMPT$="Name:____________":
        REPLY.START%=11: REPLY.END%=22: REPLY$=SPACES
        (0):
        CURRENT.COLUMN%=REPLY.START%:
        NON.PRINT.FLAG=SPACES$(1)
430 PRINT FN XY$ (PROMPT%,ROW%) PROMPTS FN XY$
        (REPLY.START%,ROW%):
440 WHILE NON.PRINT.FLAG$<>"1" AND
        CURRENT.COLUMN%<REPLY.END%+1
445     CONSOLE.CHAR$=SPACE$(1):
        NON.PRINT.FLAG$=SPACE$(1)
455     CALL DIRCTIO%
        (CONSOLE.CHAR$,NON.PRINT.FLAG$):
          REM assembly subroutine to get 1 char
465     IF NON.PRINT.FLAG$="2" THEN IF
        CURRENT.COLUMN%>REPLY.START% THEN
        CURRENT.COLUMN%-1:
        REPLY$=LEFT$(REPLY$,LEN(REPLY$)-1):
          PRINT FN XY$(CURRENT.COLUMN%,ROW%) "___" FN
        XY$(CURRENT.COLUMN%,ROW%);
470     IF NON.PRINT.FLAG$=SPACE$(1) THEN
        REPLY$=REPLY$+CONSOLE.CHAR$:
        CURRENT.COLUMN%=CURRENT.COLUMN%+1: PRINT CON-
        SOLE.CHAR$;
480 WEND
495 REM REPLY$ now contains NAME response
```

(continued from page 28)

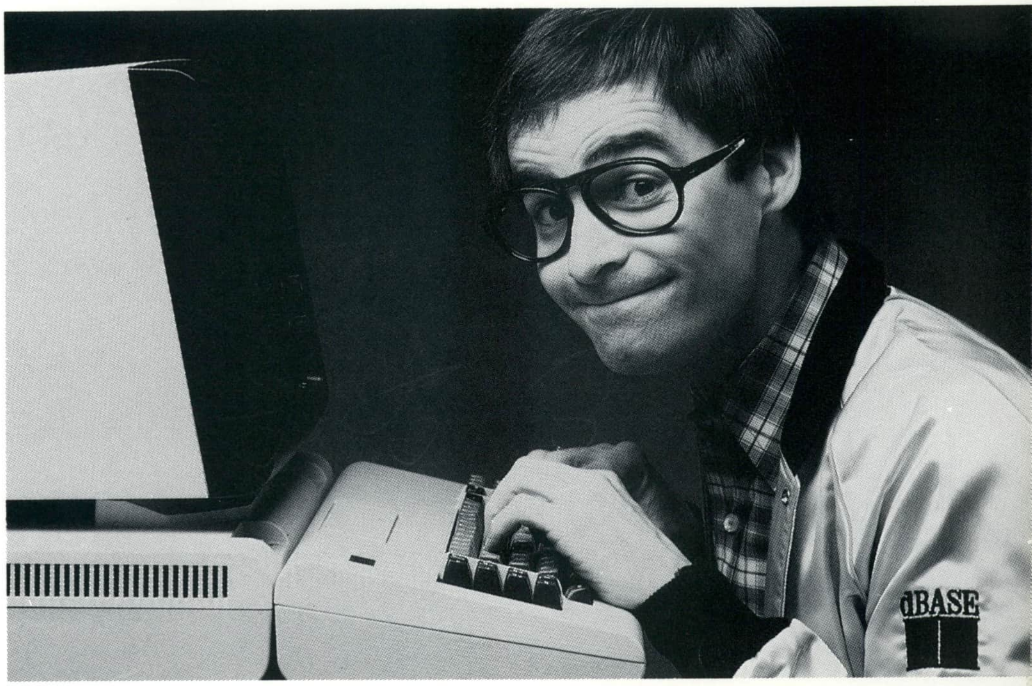
3) Home organization.
4) Personal Education. (Including the children)
5) All other.

SIG/86 Users Group
for MS-DOS (PC-DOS) Users
47-4 Sheridan Drive
Shrewsbury, MA 01545
617-845-1074

Membership Information:
Rates: $18.00/year
10.00/6 months

Bulletin Board:
Free and available to anyone.
300 baud access: 617-842-1435
1200 baud (212A) access:
617-842-1712
11 PM to 8 AM daily, other hours on notice both on the system and through out newsletters.

All you dBASE II™ hotshots are about to get what you deserve.



You've written all those slick dBASE II programs.

Business and personal programs. Scientific and educational applications. Packages for just about every conceivable information handling need.

And everybody who sees them loves them because they're so powerful, friendly and easy to use.

But that's just not good enough.

Uh-uh.

Because now you can get the gold and the glory that you really deserve.

Here's how.

We've just released our dBASE II RunTime™ application development module.

And it can turn you into an instant software publisher.

The RunTime module condenses and encodes your source files, protecting your special insights and techniques, so you can sell your code without giving the show away.

RunTime also protects your margins and improves your price position in the marketplace. If your client has dBASE II, all he needs is your encoded application. If not, all you need to install your application is the much less expensive RunTime module.

We'll tell the world.

With your license for the dBASE II RunTime module, we provide labels that identify your program as a dBASE II application, and you get the benefit of all the dBASE II marketing efforts.

We'll also provide additional "how to" information to get you off and running as a software publisher sooner.

And we'll make your products part of our Marketing Referral Service. Besides putting you on our referral hotline, we'll publish your program descriptions and contact information in *dBASE II Applied,* a directory now in computer stores world-wide.

Go for it.

But we can't do any of this until we hear from you.

For details, write RunTime Applications Development, Ashton-Tate, 10150 West Jefferson Boulevard, Culver City, CA 90230.

Or better yet, just call (213) 204-5570. And get what you deserve today.

dBASE™ II

ASHTON·TATE

Lifelines™/ **The Software Magazine**
1651 Third Ave., New York, New York 10028

NEW YORK N.Y. JUL 6'83 U.S. POSTAGE 71

FIRST CLASS

[redacted]