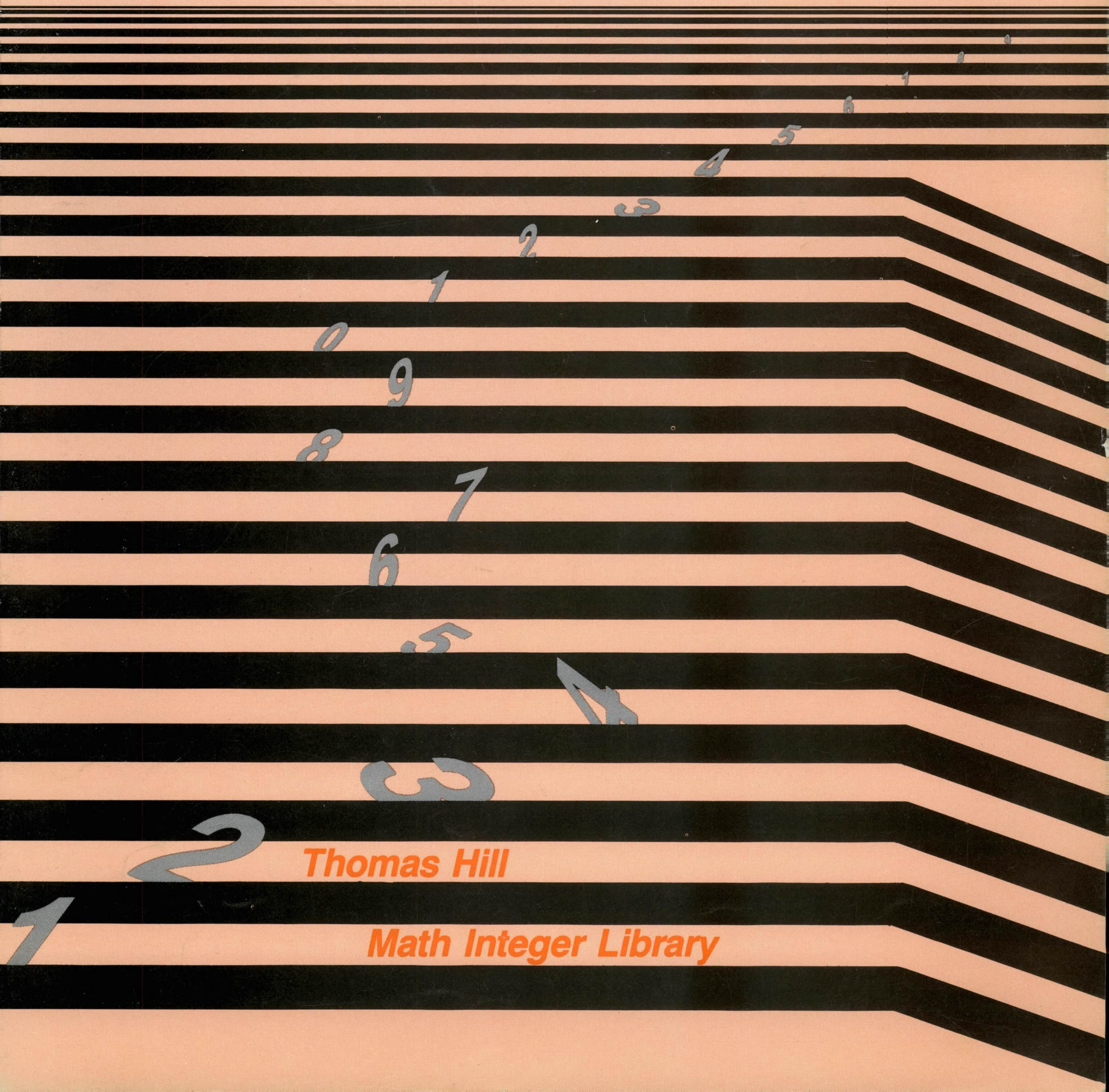
LIFELINES™

The Software Magazine™

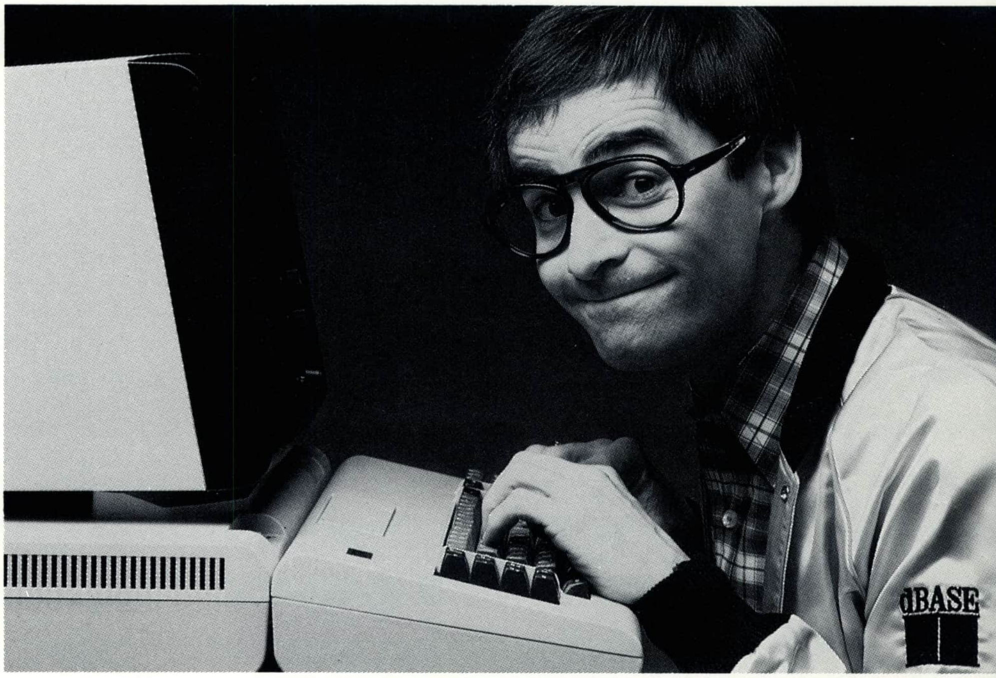
$3.00 July 1983 Volume IV, No. 1 (SSN 0279-2575,USPS 597-830)



Thomas Hill

Math Integer Library

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

Publisher: Edward H. Currie
Editor in Chief: Susan Sawyer
Production Manager: Kate Gartner
Technical Editor: Al Bloch
Art Director: Kate Gartner
Typographers: Paul Blockhaus, Rosalee Feibish
New Versions Editor: Lee Ramos
Advertising Manager: Carolann Abrams
Circulation Manager: Trina McDonald
Printing Consultant: Sid Robkoff/E&S Graphics
Dealer/Customer Service Manager: Crescent R. Varrone
Cover: Kate Gartner

Opinion

Features

Software Notes

Product Status Reports

Miscellaneous

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. Canada, 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.

Opinion

by Edward H. Currie

Pogo Was Right. . . .

Last month we discussed a number of "orbiter dictums" that had been promulgated by the "experts" among us. This month we will begin a review of what happened to each of these pontifications.

The first was the belief that multiuser operating systems were important for micros. When Intel designed the 8080 it was assumed that the advantage of "micro" technology was to provide high powered computer technology at an extremely low price. This meant that a system designer could afford to sprinkle microprocessors liberally throughout the system. Keyboards, floppy disk subsystems, printers, etc., were all to have their own micro. Unfortunately, once someone realized that the CPU was spending a great deal of its time in neutral waiting for the painfully slow peripherals it was decided that something must be done. Programmers set to work to develop a multiuser environment. The end result of their labors was a primitive multiuser/multitasking environment in which many exisiting applications didn't work. Puzzled authors tried to determine what applications they wanted to write which would allow numerous users of the same three dollar microprocessor to each execute several tasks contemporaneously.

Needless to say nothing much was written and today with the emphasis upon networking it seems unlikely that it ever will. After all networking was the obvious choice. Multitasking is important for the typical user only because it allows him to continue to use his machine when it is printing or engaged in file transfer via telephone lines, or other low level background tasks.

Another popular myth was that end users would write the bulk of their own applications software in BASIC. While BASIC has proven to be a powerful language in the micro environment and continues to be the leading language in terms of language sales, few of you are writing programs in it. The reason is simple. Few people are writing programs in any language and with the advent of products such as the Lattice C compiler, BASIC/Pascal have taken a back seat.

Pascal continues to enjoy a great deal of popularity but has been largely "undone" because of extensions which have had a serious impact upon the portability of Pascal source programs. C, a serious competitor, remains a pure language and C compilers for all machines are readily available. The most important criterion for determining the success of a language is the number of applications written in the language. So get busy, all you UCSD experts, and produce a library of good applications.

As to program generators, they are touted by the diehards who are convinced that complex programming tasks can all be reduced to trivia if one uses a program generator. The fact that program generators are having to struggle to get a foothold while authors continue to write in Pascal and C is sufficient evidence that such applications are not to dominate.

'Eight bit is dead" is still widely held, but only by those who think that sixteen bit hardware is superior to eight bit systems. This is hopefully to be the case but cannot be until and unless the same vast library of software is available for sixteen bit machines.

Computer hardware is becoming cheaper and cheaper but the important thing to realize here is that a computer system has a software component also. Software and more importantly, software support show little sign of becoming anything except more expensive as time goes on, and end users continue to demand more and more sophistication, better documentation, increased support.

It has long been held that computer dealers were the only significant distribution channel for software. The advent of software stores, availability of software in bookstores and departement stores, bundling of software with hardware, etc., indicates that all known distribution channels are to utilized.

The use of micros for accounting purposes has existed since Peachtree released their first application package. However the average micro user is not an accountant and the average small business can find more appropriate uses for microcomputers. Floppy disk based systems may not be appropriate for accounting functions even in the smallest of businesses.

Hardware manufacturers continuously relearn certain inescapable facts one of which is that an unhappy customer means trouble. Whether the customer is unhappy with the hardware he purchased from either the manufacturer or the manufacturer's representative or unhappy with the software, the result is the same, trouble. The end user is unlikely to complain as much to the dealer who sold him a two hundred dollar package as to the person who sold him the five thousand dollar system which has become as useful as a block of granite because of some software problem.

While it is perhaps true that any bits are better than no bits, a more profound truth is the observation that a computer is only as good as the software available for it. Thirty two bit machines will have a difficult time replacing their sixteen bit counterparts until a good software library is available.

The Japanese have so far not been successful in finding mass distribution channels for their products in the United States. Distribution methods used for TV's, stereos, etc., are just not appropriate for the bulk of the Japanese computer offerings.

(continued on page 34)

Feature

Learning Not to Swear at Your DELETE Key

by Mark R. Gardner

ONE FEATURE OF CP/M IS LEFT OVER FROM THE primitive teletype days: the echo of characters "rubbed out" with the DELETE key. The already printed character on the teletype roll couldn't be effaced, so the clever designers made the DELETE (or RUBOUT) key echo, and gave us control-R to replot the line so we could see what we were doing. I'm not the world's best typist, nor worst, but I make enough mistakes that I get frustrated at the clobbered lines that result from using DELETE instead of BACKSPACE (this last is control-H, and works the way CRT's deserve — the character being deleted is overwritten with a space, and the cursor parked appropriately). On the keyboards I've used, the DELETE key is often handier than the BACKSPACE key (DataMedia DT80/1, DEC VT100, Intel MDS, my own Toshiba T100), and anyway, its silly to be stuck with this atavistic feature that assumes my seven color (plus black) display is on a roll of paper!

I've solved the problem. I found the BDOS instruction that traps the BACKSPACE key, and made it trap the DELETE key as well. (Fortunately, whatever code previously processed the DELETE key executes after my patch, and hence is never executed.) My version of CP/M is 2.2, and has no trouble working with my patch. I've also installed it in an Intel MDS under CP/M, and it behaves just as well (better, in my opinion, since the DELETE key works correctly). I believe it will work in any version 2.2, but I can't test every computer, alas, that runs it. I'll leave that up to you.

Where the patch goes

My Toshiba T100 has a BDOS based at 0D000H. The compare instruction for the backspace key is at 0D202H, and there are some unused bytes at 0DDF0H (the end of the BDOS just before the start of the BIOS). The compare instruction and the conditional jump just following it are changed to be just a jump to the patch area at 0DDF0H. The patch area is then filled with the original compare and jump corresponding to the BACKSPACE key, and also a compare and jump to correspond to the DELETE key. The two sections of memory are shown in Fig. 1, before and after the patch.

Naturally, not every BDOS is based at 0D000H, but at locations depending on the amount of memory. If you want to install the patch by hand, you can, but you'll have to find where your BDOS starts, and DDT is a dandy tool for doing this, and also for installing the patch. In DDT, dump from 0 and note the contents of locations 6 and 7. These are NOT the base of the BDOS, although they WERE until DDT was loaded and changed them. However, all is not lost. Now dump from the location given by those locations (remember that the least significant byte is first). The first three bytes are now ANOTHER jump (sigh, be patient, we're getting there). Dump from the location specified by this jump, and *voila!* you've got it. The seventh and eighth bytes in the dump give the address of the base of the BDOS. Just subtract six. Easy, right? Fig. 2 shows the process I've just described.

How to put the patch in

To continue, it's time to put the patch in. You can use the H command to calculate the addresses, but you can probably do it in your head. The new program goes at the BDOS base + 0DF0H, and a jump to it replaces the 5 bytes at the BDOS base + 0202H. Fig. 3 shows the calculations of the two patch addresses, and continues with the use of the A command to enter the new code and the overwrite of the old code. Notice the oops! at my attempt to enter a NOP instruction at 0D202H. I did NOT press control-C, I pressed an 'N', the first letter of NOP. Unfortunately, the previous entry of NOP had changed code in progress, and the program ran through the NOP smack into the following 08, which is an EX instruction on my Z80, undefined on 8080's. Anyhow, it broke the program. It would be better to continue as shown in Fig. 4, by writing a little program somewhere to make all the changes at once. Completed, the patch works immediately.

Here's a better way

If you'd like something a little easier (after all, the patch goes away when you do a cold boot, as at power-on), all you have to do is enter and assemble the program I have sent along with this little article (presented in Fig. 5). It automatically determines the BDOS base, installs the patch, and returns to CP/M. After boot, just run it once, and everything is cool. It beats the DDT operation by a mile (or at least, 75 seconds), and is less prone to failure. You don't need to type in all the comments, but they might be handy to you later. The code is commented fairly well, so I won't say much here. I will point out that the code contains the patches, with their contained jumps labelled. This is so the PUTPAT portion of the program can calculate the appropriate REAL jump addresses and stash them within itself before moving the patch portions to the two respective places in memory where they reside. This is self-modifying code, generally considered dangerous, so if you try it in your own work, be careful, or you too may see control-C appear when you only type an 'N'.

Could patch the system tracks

I plan to install this patch in the image of CP/M on the system tracks on my disk. There are various ways to do this, from SYSGEN to various "zapper" programs. I plan to use a bugger program that I am currently writing, but

(continued on next page)

it's not quite ready. In any case, each of you will find a favorite way to get it onto your disk, if running the program at each boot is not sufficient. In the meantime, if you discover how to make the BACKSPACE key work like the DELETE key, I don't want to know.

```
BEFORE:
  ■ DD202,D20F
  mem       2  3  4  5  6  7  8  9  A  B  C  D  E  F  ↑0 ↑1
  D202      FE 08 C2 16 D2 78 B7 CA EF D1 05 3A 0C D3 32 0A .....x.....:..2.
  ■DDDFO
  mem       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
  DDFO      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...............
AFTER:
  ■ DD02,D205
  mem       2  3  4  5  6  7  8  9  A  B  C  D  E  F  ↑0 ↑1
  D202      00 00 C3 F0 DD 78 B7 CA EF D1 05 3A 0C D3 32 0A .....x.....:..2.
  ■DDDFO
  mem       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
  DDFO      FE 08 CA 07 D2 FE 7F C2 16 D2 C3 07 D2 00 00 00 ...............
```

FIG. 1. Toshiba T100 memory before and after DELPATCH, provides operation of DELETE key identical to BACKSPACE key, i.e., erase, not echo.

```
A>DDT
DDT VERS 2.2
—D0,F
0000      C3 03 DE 80 00 C3 00 CO 00 00 00 00 00 00 00 00 ...............
                            ↑  ↑
—DC000,C00f
C000      C3 A2 C6 00 00 00 C3 4F C3 C3 24 C5 00 01 1E EB .......0..$.....
             ↑  ↑
—DC6A2,C6AF
C6A2      E3 22 4A CF E3 C3 06 DO 2A 06 00 22 AB C6 .■J   .....*..■..
                            ↑  ↑
```

FIG. 2. Procedure in DDT to find base of BDOS for your computer. Numbers here are for my Toshiba T100. Note final result is 0D006H.

```
—HD006,6
D00C D000
—HD000,202
D202 CDFE
—HD000,DF0
DDF0 C210
—DD202,D20F
D202      FE 08 C2 16 D2 78 B7 CA EF D1 05 3A 0C D3       .....x.......:.
—DDDF0,DDFF
DDFO      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ...............
—ADDF0
DDFO          CPI  8
DDF2          JZ   D207
DDF5          CPI  7F
DDF7          JNZ  D216
DDFA          JMP  D207
DDFD
—AD202
D202          NOP
D203          ↑C
```

FIG. 3. Continuing with hand entry of patch. Note calculations with H command, and dumps to verify contents before proceeding. Also note the oops! at 0D203H.

```
—A2000
2000          LXI  H,0
2003          SHLD D202
2006          MVI  A,C3
2008          STA  D204
200E          SHLD D205
2011          RST  7
2012
```

```
—G2000
*2011
—↑C
A>
```

FIG. 4. The better way to change location 0D203H. Short program entered at 2000H allows all 5 bytes to be changed (safely) at the same time. The RST 7 instruction returns control to DDT after the change, and the patch will work.

```
; DELPATCH.ASM 4/4/83 MARK R. GARDNER
; Copyright 1983 MRGENT Consulting

; Program to patch the BDOS so that the DELETE key functions (at the CP/M command line) like the BACKSPACE key, i.e., it backspaces and overwrites
;   with a space, rather than echoing the deleted character.

; Program in 8080/8085 assembly code. Assemble with ASM, convert to .COM file with LOAD, run with DELPATCH. Program requires no arguments on
;   control line, and will return to CP/M when finished. If it finds that the first instruction at PT1LOC is not a CPI 08, it assumes the installed CP/M is not
;   patchable, and aborts. Note that this program will not work properly under DDT, since DDT changes the contents of locations 6 and 7 that this
;   program uses to find where the patch belongs.

        ORG     100H
        JMP     PUTPAT
BDOSST  EQU     6               ; LOCATION OF ADDRESS OF BDOS (PART OF THE JUMP BDOS AT LOCATION 5)

; The following locations have 6 subtracted to offset the fact that the BDOS jump at location 5 is to BDOS base + 6.

PT1LOC  EQU     202H-6          ; INSERT LOCATION PATCH 1
PT2LOC  EQU     0DF0H-6         ; INSERT LOCATION PATCH 2
JPBAK1  EQU     PT1LOC+5        ; REENTRY FROM PATCH 2
JPBAK2  EQU     216H-6          ; REENTRY FROM PATCH 2

PATCH1:         ; PATCH 1, REPLACES BDOS FROM PT1LOC TO PT1LOC+4 (5 BYTES)
        NOP
        NOP
PT1JMP:
        JMP     PT2LOC          ; THIS JUMP LOCATION IS A DUMMY—IT IS REPLACED (SEE AT PUTPAT, BELOW) BEFORE THE
                                ; PATCH IS MOVED TO THE BDOS. (THIS IS TRUE FOR THE THREE JUMPS IN PATCH2 AS WELL.)

PATCH2:         ; PATCH2, NEW CODE IN BDOS AT PT2LOC
        CPI     8               ; SEE IF IS BACKSPACE
JZBK1:
        JZ      JPBAK1          ; IF SO, DON'T GO TO JPBAK2
        CPI     7FH             ; SEE IF IS DELETE KEY
JNZBK2:
        JNZ     JPBAK2          ; IF SO, DON'T GO TO JPBAK2
JMPBK1:
        JMP     JPBAK1          ; (SEE NOTE ABOUT JUMPS IN PATCH1.)

PUTPAT:         ; PUT THE PATCH IN MEMORY
        LHLD    BDOSST          ; CALCULATE THE REAL JUMPS FOR THE PATCHES
        LXI     D,PT2LOC
        DAD     D               ; HL NOW CONTAINS REAL PT2LOC
        SHLD    PT1JMP+1        ; SO FIX PATCH1

        LHLD    BDOSST
        LXI     D,JPBAK1
        DAD     D               ; COMMENTS ETC. AS JUST ABOVE
        SHLD    JZBK1+1
        SHLD    JMPBK1+1
        LHLD    BDOSST
        LXI     D,JPBAK2
        DAD     D
        SHLD    JNZBK2+1

CHKBD$:         ; CHECK THE BDOS FOR CPI 8
        LHLD    BDOSST          ; CALCULATE THE LOCATION TO INSTALL PATCH
        LXI     D,PT1LOC
        DAD     D               ; HL POINTS TO PATCH AREA

        MOV     A,M             ; SEE IF IS CORRECT INSTRUCTION
        CPI     0FEH
        JNZ     BADBDS          ; IF NOT, REFUSE TO PATCH IT
        INX     H
        MOV     A,M
        CPI     8
        JN2     BADBDS
        DCX     H
```

(continued on next page)

```
                                  ; HL STILL POINTS TO PATCH AREA
         LXI     D,PATCH1         ; DE POINTS TO THE PATCH
         MVI     B,PATCH2-PATCH1  ; SET THE PATCH LENGTH
         CALL    MOVEIT           ; AND PUT IT IN PLACE

         LHLD    BDOSST
         LXI     D,PT2LOC
         DAD     D                ; CALCULATE LOCATION TO INSTALL PATCH2
         LXI     D,PATCH2         ; POINT TO THE PATCH
         MVI     B,PUTPAT-PATCH2  ; SET THE PATCH LENGTH
         CALL    MOVEIT           ; AND PUT IT IN PLACE
         JMP     LEVPAT

BDBSMG:          ; BAD BDOS MESSAGE
         DB      13,10,'THIS PATCH DOES NOT FIT YOUR BDOS.',13,10
         DB      '(SEE THE SOURCE PROGRAM FOR NOTES)',13,10,'$'

BADBDS:          ; BAD BDOS
         LXI     D,BDBSMG
         MVI     C,9              ; BDOS PRINT STRING
         CALL    5                ; TO TELL USER PATCH NOT DONE

LEVPAT:          ; LEAVE THE PATCH PROGRAM
         MVI     C,0
         CALL    5                ; RETURN TO CPM WITH INSTALLED PATCH

MOVEIT:          ; PERFORM THE SPECIFIED MOVE
         LDAX    D
         MOV     M,A              ; PICK AND PLOP
         DCR     B
         RZ
         INX     D
         INX     H
         JMP     MOVEIT           ; CONTINUE UNTIL ALL MOVED
         END                      ; THAT'S ALL
```

FIG. 5. DELPATCH.ASM, an assembly program to perform the patch (if appropriate) and return control to CP/M. This is recommended, unless you can easily patch the system tracks on your disk, to permanently preserve the patch at each boot.

A Call for Manuscripts

Perhaps you've done some writing before. Or maybe you've always wanted to write. It could be that reading *Lifelines/The Software Magazine* has given you some ideas on what you have to contribute. We're interested in hearing what you have learned, and so are other readers. Whatever *serious* CP/M-80 compatible software you've been using, we'd like you to write for us. We like to publish both long essays and those short gems which can hold so much important information.

Send us a brief resume of your software experience, and samples of your previous writing, if you have any. (Don't be shy if you're *not* an experienced writer.) Then we can talk about your work and about payment for your efforts. Write or call: Editorial Dept., Lifelines Publishing Corp., 1651 Third Ave., New York, N.Y. 10028. Telephone: (212) 722-1700.

SpellStar Revisited

by Robert P. VanNatta

IN THE APRIL, 1982 ISSUE OF *Lifelines*, I reviewed SpellStar version 1.0 from MicroPro. This is, of course, the dictionary overlay for WordStar. In that article I observed that I liked the program just fine except that it 1) worked too slowly, 2) crashed frequently, 3) and would not work with my Radio Shack video board (meaning that it was unusable with Pickles and Trout CP/M and only marginally useful under Lifeboat CP/M).

A more recent issue of *Lifelines* (September 1982) carried a response from MicroPro claiming that the problems had been cured with a new version of SpellStar (version 1.2).

Only because of that response did I learn that a new version had been released. The update is a "no charge" update, but if you don't know to ask for it you won't get it. (I can't help but contrast this with the Digital Research policy of mailing postpaid updates to registered users who buy early non-working versions of their products.)

Anyhow, I have now tested SpellStar version 1.2 (and also an even later version called 1.21). It is much revised. Physically the size of SPELLSTAR.OVR has been reduced from 30k to 18k. The unexplained departures to the operating system no longer occur while checking large files. The speed is now competitive. For example, on a small file of 66 words SpellStar version 1.0 required a full minute; while a competitive spelling program, Spellguard, required 27 seconds. Version 1.2 requires 14 seconds.

On a moderate file of 422 words Version 1.2 and Spellguard were of equal speed. On a very large file, 12,000 words, Spellguard is still faster (1:12 minutes compared to 2:12 minutes for the checking routines), but SpellStar is in the ballpark.

I will not be completely happy with any dictionary routine until it works with the speed and convenience comparable to that of the paragraph reform key (Control B). Unfortunately, I all too well understand the constraints of a 64k memory board, and must recognize that an "on line" spelling checking routine is not feasible in the 8080-Z80 environment.

Among the gripes that I have with SpellStar version 1.2 is that it crashes during the correction routine when you are checking large files. SpellStar contains a small buffer which is supposed to hold the last few words that you have ignored during the correction routine. The idea is that if you come upon a correctly spelled word that is not in the dictionary, after you bypass it the first time, the word will be remembered and automatically bypassed in the future. This buffer holds 10 to 20 words and is supposed to dump itself out the back end if more words are accumulated than will fit.

Unfortunately, on my Model 16, under both Pickles and Trout CP/M and Lifeboat CP/M, SpellStar crashes on the 15th word during the correction routine. The error message is an "Internal Error I18 (memory Shortage)." The only good thing about it is that the crash landing is soft. The error is trapped and a push of the escape key will land you in WordStar in the edit mode. SpellStar can then be restarted with ↑L, and the correction routine can be completed without any further problems. The other problems include the failure of SpellStar to recognize the clear screen command of Pickles and Trout CP/M. This does not interfere with the usefulness of the program but does render it a cosmetic disaster area with that installation. In a similar vein, the SOROC 120 installation routine that is used by the ATON variant of CP/M for the TRS80 Model II/16 did not correctly identify the inverse video codes. SpellStar displays correctly without highlighting, but it is a shock to see it in that mode, after being dazzled by the extensive highlighting that appears under the Pickles and Trout and Lifeboat variants of CP/M.

Several months of use have convinced me that the dictionary maintenance routines have a potentially serious limitation. You are freely permitted to add words to the dictionary at your whim. Unfortunately, you are not told anywhere of either the limit of additional words permitted or the penalty for exceeding the limit. I have discovered that you can reliably tell when the dictionary will not accept additional words by merely watching the "tube" during the dictionary update. When SpellStar mumbles about an internal error and executes a warm boot, you know that the dictionary is full. Never mind that it also destroyed your master dictionary file at the same time! It was too full anyway, and you, of course, have a backup copy for future use.

A final irritation that has come to my attention is that all words that are designated for addition to the dictionary are stored in a temporary file called the ADD file. This is fair enough, except that each time you enter the edit mode the file is destroyed and recreated. This means that if you work through a long file and periodically execute a ↑KS, to save and reedit, you will find that upon completion your ADD file will only contain the words flagged for the dictionary since the last SAVE.

In summary, I still find SpellStar pleasant and easy to use. Version 1.2 gets to the point where it crashes much more quickly than did version 1.0, and unlike the fatal crashes of the earlier version the check-time crash is recoverable. This has got to be a great improvement. I now avoid the main dictionary explosions by putting my extra words in a supplemental dictionary. Believe it or not, I continue to use this "gobbler" in preference to several other dictionary programs that are gathering dust within my grasp. My reason for this is that I like the program design. I merely wish that it worked. It may be that by the time you see this in print MicroPro will have a still later version on the market. I surely hope so, but since I am merely a paying customer

(continued on next page)

of MicroPro and a registered user of SpellStar, don't expect me to know about it.

After I wrote most of this article and before I sent it for publication I made one last check to see if version 1.2 was the current version. I found out it wasn't. Somewhere in the last six months version 1.21 had appeared. My dealer claimed that it fixed numerous bugs that were present in version 1.2. Anyhow, I got version 1.21 which was another "no charge" update. Amid high expectations I checked for the same bugs that plagued version 1.2. Of the three bugs described in this article it is one for three. The screen display now works properly under ATON CP/M for the Radio Shack II/12/16. It still doesn't work right for Pickles and Trout CP/M. Also, SpellStar still crashes on large files with an internal error.

By the time you read this there will be still another version of SpellStar out, as a new version of WordStar has been announced for June 1983. This version of WordStar will require different overlays for both MailMerge and SpellStar. For my own two cents worth, my frustration level with SpellStar has reached the point that I don't much care whether they ever get a version that works or not because I am going to be using something else.

A hot prospect for a substitute for SpellStar is THE WORD PLUS by Oasis. It is a stand-alone system that will work with almost any word processing program. It is competitively priced, faster than SpellStar, and has the ability to suggest corrections for suspect words. THE WORD PLUS can be configured so that it uses a ↑@ (00h) for an error flag. If this is done you can actually use the convenient SpellStar correction routines (triggered by a ↑QL) that are built into WordStar even though you don't own SpellStar.

Actually the correction routines provided by THE WORD PLUS are even more convenient than those of SpellStar, but it is useful to use both, as THE WORD PLUS will leave a flag where spelling corrections change word length or by any word that you choose to flag rather than correct.

The convenience of this is that you can make a quick pass through the corrected file using the WordStar ↑QL command and do any housekeeping or reformng that may be indicated.

Conclusions

This writer keeps his head in the sand most of the time, so I have probably missed something, but if there is something that SpellStar does better (besides crash) than THE WORD PLUS, I simply haven't found it. One possible issue, however, would be disk space. With a disk change SpellStar will run on a system with at least two 108k floppies. The dictionary for THE WORD PLUS consumes 138k so don't expect those single density 5¼-inch floppies to handle THE WORD PLUS. It is possible that someone who has less than the 2.5 megabytes of floppy disk storage, characteristic of my TRS80 Model 16, might be more concerned about this requirement but from my viewpoint THE WORD PLUS is well worth the extra space. ■

Tips and Techniques

In the February Z80 tutorial, Kim DeWindt expresses interest in the possible uses of the RLD and RRD instructions. These operations are paricularly useful for BCD arithmetic. In fact, a single RLD or RRD instruction can often replace several lines of 8080 code.

For example, suppose a packed BCD number (2 digits per byte) is located in memory at addresses PACKNUM thru PACKNUM + PACKLEN − 1. Then, the following code shifts the entire number right by one nibble and inserts zero in the high order nibblez.

```
        XOR  A          ; Clear accumulator.
        LD   HL,PACKNUM
        LD   B,PACKLEN
BCRIGHT RRD             ; Nibble from prior byte to (HL).
                        ; Low nibble of (hl) to A.
        INC  HL         ; Point to next byte.
        DJNZ NCRIGHT
```

This routine could be used to align operands for BCD floating point arithmetic and is much faster than code using single bit shifts.

As another example, we might need to print a BCD number. First we must change it to ASCII format. The following code performs the required conversion for PACKNUM and stores the result at ASCNUM:

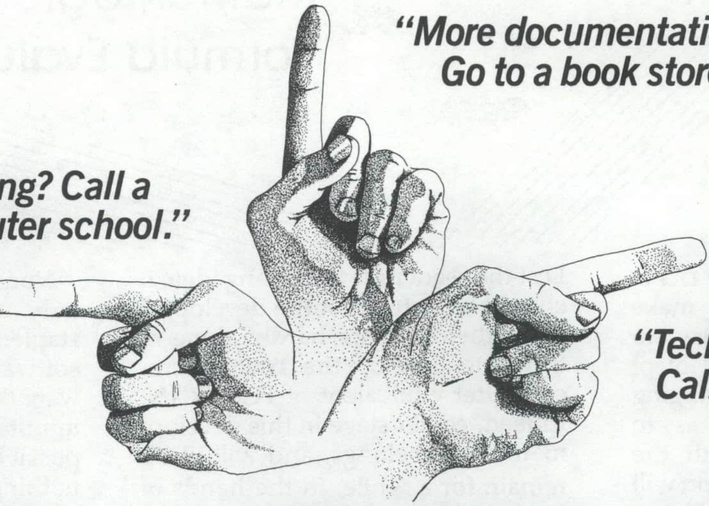
```
        LD   HL,PACKNUM ;
        LD   DE,ASCNUM
        LD   B,PACKLEN
        LD   A,30H      ; High nibble for ASCII format
UNPACK  RLD             ; High nibble of (HL) TO Accum
        LD   (DE),A     ; Store ASCII byte.
        INC  DE         ; Point next to ASCII location.
        RLD             ; Low nibble of (HL) to Accum
        LD   (DE),A     ; Store ASCII byte.
        INC  DE         ; Point next ASCII location.
        RLD             ; Restore original packed byte
        INC  HL         ; Point next packed byte.
        DJNZ UNPACK
```

The Z80 instruction set contains several features which enhance its ability to do BCD arithmetic. Would *Lifelines* be interested in an article on this subject which would include implementations of all the standard operations?

Sincerely,
Robert Pirko

"More documentation? Go to a book store."

"Training? Call a computer school."



"Technical support? Call the publisher."

Interested in dBASE II™ or 1-2-3™? Beware The Dreaded Finger Pointers!

Sound familiar? Does your dealer turn into a "finger pointer" when you need help?

At SoftwareBanc we offer a complete system that doesn't stop when your software is delivered.

Careful Product Selection

Do you get bewildered by the endless lists of software you find in most ads? Let us be your quality control department.

We only sell the best programs on the market. After a thorough evaluation we chose dBASE II™ for data processing, and 1-2-3™ for financial management.

Our complete line of add-on products help you to continue to get the most from your software.

Expert Technical Support

When you buy software from us, you can rest assured that help is only a phone call away. Just call us at (617) 641-1235 for all the free support you need.

Free dBASE II™ User's Guide

Order dBASE II™ from us, and you'll receive a free copy of our dBASE II™ User's Guide. You can also buy the User's Guide first for only $29, and then receive a full credit when you buy dBASE II.™

French Translation
La Commande Electronique
5 Villa Des Entrepreneurs
75015 Paris, France

Japanese Translation
JSE Int'l
9F Toyo Bldg. 6-12-20 Jingmae
Shibuya-ku Tokyo, Japan 150

1-2-3™ & dBASE II™ Classes

Want more in-depth information about dBASE II™ or 1-2-3™? Attend a SoftwareBanc Seminar near you. Each session runs from 9 to 5, and costs $175. Seminars are in lecture format with a custom sound & video system which is used to display taped interviews with prominent software personalities and sessions with various software programs.

Los Angeles
July 18-22

Anchorage
August 11-12

Washington, D.C.
Aug. 29-Sept. 2

New York City
September 19-23

Prices You Can Afford

†1-2-3™	$399
†dBASE II™	$479
†ABSTAT™	$379
dBASE II™ User's Guide	$29
DBPlus™	$95
dGRAPH™	$199
dUTIL™	$69
dNAMES™	$109
QUICKCODE™	$199
TEXTRA™	$60*

†No-risk 60 day money back guarantee
*Only available for IBM PC with MS-DOS.

Free Catalog

If you want to learn more about SoftwareBanc, call or write for our free product catalog.

SoftwareBanc

661 Massachusetts Avenue
Arlington, Mass. 02174
For technical support call:
(617) 641-1235

Dealer Inquiries Invited.

™ Manufacturer's trademark
Payment may be made by: MasterCard, Visa, check, C.O.D., money order. Mass. residents please add 5% sales tax. Add $5.00 for shipping and handling. Prices subject to change.

SoftwareBanc
Order Toll Free
1-800-451-2502
(617) 641-1241 in Mass.

SoftwareBanc
661 Massachusetts Avenue
Arlington, MA 02174

MatheMagic and The Art of Formula Evaluation

by Davis A. Foulger

WHAT WOULD YOU DO IF you wanted to make BASIC really easy to use? There are a lot of answers to that question ranging from "BASIC is already really easy to use" to "Learn LOGO." In truth, the answer you give to that question will depend on the amount of experience you have with BASIC, the other programming languages you know well, and the kinds of applications you are interested in.

What is clear, regardless of how easy you think BASIC is, and BASIC is pretty easy, are the following:

1) BASIC takes a while to learn. No matter what the intentions of the language designers, BASIC does not really emulate the way people talk or think.

2) BASIC is cluttered with its past. Some statements (PRINT, for instance) are anachronistic and unnecessary in many applications.

3) BASIC rarely results in clear code, especially in longer programs. Even highly experienced BASIC programmers often have trouble figuring out how a BASIC program works.

MatheMagic is an applications program for mathematicians and others who work with complex mathematical formulas. Its developers describe it as an "ultimate calculator," but that description really does violence, at least in an age of desktop computing, to both MatheMagic and calculators. It is not as simple or portable as a calculator. It is, on the other hand, much easier to use than any calculator, especially when complex formulas are being evaluated.

MatheMagic is probably better described as a "formula translator," a description that should immediately bring at least one programming language, FORTRAN, to mind. The comparison is apt, as MatheMagic is really attempting to do, in the 1980s, what FORTRAN attempted to do in the 1950s and BASIC attempted to do in the 1960s—e.g., *to make it easy for people to (1) put formulas and data into a computer and (2) get answers back out.*

Lest the reader get the wrong idea, it should be noted that the developers of MatheMagic are hardly alone in trying to develop the 1980s microcomputer equivalent of FORTRAN. Indeed, center stage in this development effort belongs, and will likely remain for a while, in the hands of spreadsheet programs like VisiCalc, T/MAKER, SuperCalc and CalcStar, to name a few. Ultimately, any review of MatheMagic will have to be compared with both those spreadsheets and the programming languages, like FORTRAN and BASIC, that MatheMagic attempts to simplify.

MatheMagic versus BASIC

We started this review by asking what might be done to make BASIC really easy to use. This was obviously an important question to the people at International Software Marketing (ISM) who turned MatheMagic into a program. Their answers are constrained, to a large extent, by the limitations of the application they were creating. If those answers refuse to break new ground in the history of programming languages, they do, by and large, succeed.

Their most important answers are the following:

1) Make formula and data entry as simple as possible. Where the user must enter code directly—the entry of formulas and data— allow entry in a form that requires minimal training, that the user will be able to readily recognize and change. Where direct coding is unnecessary, use menus.

2) Restrict the user's options to really needed choices. A lot of BASIC language options are unnecessary in many applications.

3) Keep user coding as clean as possible by treating formulas as objects and permitting modular programming in which the result of one formula can be incorporated, without regard to placement, within another.

None of these answers is particularly new. Menus have become a staple ingredient of user friendly software, and finding menuless software that people can learn within 30 minutes to an hour is close to impossible. Restricted options are nothing new either; menus almost inevitably reduce a user's options. Critics of software have, moreover, increasingly favored languages that allow a modular approach to programming. Whether championed as structured programming (Pascal) or object-oriented programming (SMALL-TALK and LOGO), modularity is generally seen as the one feature whose inclusion in BASIC would be most beneficial over the long term.

Formula entry

From a practical standpoint, MatheMagic might be described as BASIC reduced to a series of LET statements. Formulas are entered in almost exactly the same way they are typically entered into the BASIC program. Indeed, a simple formula can probably be evaluated in the MicroSoft BASIC Interpreter on the IBM Personal Computer (the machine I'm testing Mathemagic on) with about the same number of keystrokes that would be needed in MatheMagic.

A typical LET statement of the form "LET A=B+C" would typically be entered into the IBM-PC's MicroSoft BASIC Interpreter in one of two ways. First, it might be evaluated directly, with the user writing "B=1: C=2: PRINT B+C" or "PRINT 1+2." Second, it might be incorporated into a larger program, in which case the values of B and C would probably be determined before the statement, the print operation would probably be taken care of in a later statement, and the actual formula would be entered looking something like this: "235 A=B+C."

The short unnumbered programs are evaluated in the IBM-PC Microsoft BASIC with a single stroke of the enter (return) key. The longer program is evaluated by writing RUN and then stroking the enter key. Having this range of options is one of the nice things about working in Microsoft BASIC. But those options do make BASIC harder to learn and use than it might be.

The form of a MatheMagic formula is identical to the form of the BASIC language LET statement in almost every respect, but the manner in which that formula is entered and evaluated is quite different. One must first move into FORMULA mode by typing "F." One must then implement the CREATE/CLEAR option by typing "C." Both of these keystrokes are suggested by menus and are rather easy to figure out without reading the MatheMagic documentation. Then you can write out your formula.

There are very few differences between the MatheMagic formula and the BASIC language LET statement. Indeed, we can, at this stage, enter:

"A=B+C."

The formula is perfectly valid in MatheMagic. Also perfectly valid is the formula:

"GROSSINCOME=COSTS+NETINCOME"

a statement that can be made in the IBM-PC Microsoft BASIC, but which is not possible in many BASIC implementations.

MatheMagic really isn't that picky, however. It will also calculate "B+C" or "COSTS+NETINCOME" without any problems. You get the same answer with or without the "A=." A stroke of the return key enters the formula into MatheMagic. A stroke of the "/" key evaluates the formula.

If values for B and C (COSTS and NETINCOME) have already been entered into MatheMagic, the formula will be evaluated as using those values. If values have not been entered, MatheMagic will ask for them. Clearly, the procedure involved in calculating a formula in Mathemagic is very simple. The process is largely menu-driven. The form of formula entry is reasonably intuitive. The program asks for things when it doesn't have them. Formulas are rather easy to read, change, and understand, especially when full words are used.

Program control

When you load MatheMagic up on your computer, you are confronted with three windows which are labeled, from top to bottom, the "COMMAND AREA," the "DISPLAY AREA," and the "ENTRY AREA." These screens and their implied division are a constant in MatheMagic. No matter what you do in the program, menus and messages will be displayed in the upper window, formulas, variables and answers will be displayed in the middle window, and formulas and variables will be entered and edited in the lower window.

The entry and use of formulas and variables are controlled in MatheMagic entirely through a fairly extensive menu-structure which guides the user through a limited array of choices. A Main menu allows the user to move into "Formulas" entry, calculation and editing, "Variables" entry and editing, and to perform "Printing," get "Help," or "Set" parameters for the system. Each of these choices, in turn, permits a similarly limited range of choices, ultimately allowing the user to choose between roughly twenty different actions.

This limited range of options seems more than adequate, however, for most applications. Indeed, it is quite flexible. Calculations can be performed in three different ways, with stepwise and repeating options added to the normal simple calculation command. Stepwise calculation allows the user to evaluate the formula one step at a time, an option which can be useful if you want to look at intermediate steps. Repeated calculations invoke the MatheMagic equivalent of a FOR...NEXT loop, with the proviso that MatheMagic saves the result at each loop as an array.

Few users are likely to have much use for that array within MatheMagic, but the ability to compute and save the array allows the user to use it in other programs, including ISM's forthcoming GraphMagic. Arrays are not, however, a strength of MatheMagic, as will be seen when MatheMagic is compared to Spreadsheet programs a little later in this article.

Object-oriented programming

However similar the form of MatheMagic formulas may be to the form of BASIC language formulas, there are differences which, in general, enhance the flexibility and ease of use of MatheMagic. The most trivial of these differences is the character used to separate different calculations from one another. To put more than one formula on a single line in BASIC, a colon is used (A=A+B: D=A/100). MatheMagic prefers a semicolon (A=A+B; D=A/100).

A more important difference is found in MatheMagic's use of the question mark. The appearance of a question mark at the end of a variable name within a MatheMagic formula specifies that variable as an "ask" variable. This specification tells the program to disregard any values it may have stored for that variable name and prompt the user to enter a new value. This is a particularly convenient feature, especially when a task demands the recalculation of a formula several times with changes in only one or two values.

In BASIC, the full effect of this question mark would require several lines of code, including INPUT, STOP and GOTO commands (the GOTO might be accommodated by a WHILE...WEND or IF...THEN...ELSE loop). Thus MatheMagic's question mark is clearly a rather powerful feature.

The question mark is almost insignificant in importance, however, when compared with Mathemagic's stored formula feature. Without the stored formula feature, MatheMagic would be nothing more than a rather interesting applications program. With it, MatheMagic becomes a user-friendly programming language which a person learns without ever knowing that he or she is actually programming.

MatheMagic allows the user to save formulas and variable sets for later use. This is an important feature, if only because its gives MatheMagic the ability to perform calculating tasks that would be difficult or impossible on all but the most powerful Hewlett-Packard and Texas Instruments Magnetic Card

(continued on next page)

Programmable Calculators. It allows those tasks to be performed, moreever, more easily, and at much greater speed, than would be possible even on such powerhouses as the Texas Instruments SR-52/TI-59 calculators or the Hewlett-Packard HP-67/HP-41 calculators.

Formulas, once saved, can be recalled at will, enabling the user to return to frequently used but complex formulas rather quickly. Recall of formulas has been set up so flexibly, however, that one formula can actually be recalled, and evaluated, by another formula. This recall is initiated when MatheMagic sees an ampersand (&) at the beginning of a "variable name" in a formula. The ampersand tells MatheMagic to go find the formula with the name following the ampersand, evaluate it, and use the answer obtained as the value required at this point in the formula.

Here we find the kernel of an object-oriented programming language where formulas become objects to be manipulated as one might manipulate index cards on a desk. As long as the object exists, it can be used, and used anywhere, in any relation to other objects, as the user prefers. Now I don't want to say that you can't do this in BASIC. The truth is that MatheMagic is written in BASIC, but BASIC does not allow a user the ease and flexibility assumed in MatheMagic.

Let us take as an example the formula pair "A=B+C; D=A/100" that was used earlier. BASIC requires me to include both of these formulas in any new program that needs them. It also requires that they appear in a particular order in the program, with the LET, INPUT, or READ statements that establish the values of the A and B formula appearing *before* the formula that determines D.

Such restrictions don't apply in MatheMagic. Saving the formula A=B+C creates an object that can be called by any other formula in MatheMagic, so long as the disc that holds that formula is in the machine when the formula is called. If I save it under variable name @A (which is certainly descriptive), I need only use @A in a formula for it to be called. I am not, moreover, restricted in making such calls. I can call the same formula several times in the same formula and in several different formulas.

The Formula As Program

This feature makes it convenient to think of formulas as modular programs in which one program has the capability of calling another program to help out. This kind of modularity is, of course, the very essence of object-oriented programming languages like SMALL-TALK and LOGO, and users might find MatheMagic a valuable addition to their inventory if only to experience what it is like to write object-oriented programs.

Returning to our example, BASIC's "D=A/100" becomes "@A/100" in MatheMagic. This formula can also be saved (perhaps under the name "@D,") and used in still other formulas. It is possible to build rather deep stacks of formulas using this feature, which is both a convenience and a danger. The convenience comes in the ease with which highly complex programs can be written once the underlying objects are built; in the ease with which an entire complex of formulas can be changed and debugged. The danger comes in the areas of memory size (deep stacks of formulas may threaten the memory space of a microcomputer) and recursion. Recursion is perhaps the larger danger, as it is possible to write and save a formula that calls itself.

The most exciting features of Mathemagic are stored formulas from the standpoint of ease of use and software development. The object-oriented nature of MatheMagic makes it very easy to master and maintain. That is, of course, the most important impact of the decision to make MatheMagic object-oriented. Ease of use should help MatheMagic to market success.

But the object-oriented features are also something of a breakthrough for the BASIC language itself. MatheMagic is written in BASIC, and if the principles that allowed MatheMagic to achieve modularity could be somehow extended to BASIC, BASIC would become a much more satisfying language to use.

Room for improvement

Clearly, MatheMagic has some nice features, but there is room for improvement in several areas. The first is the display, which operates on my IBM-PC in a 40-column mode that I find rather bothersome and unnecessary. Eighty columns would make it easier to enter long formulas and would be somewhat more eye pleasing on a display that almost never runs in anything but 80-column mode.

The program could also be somewhat smoother in its calculations. The user sees entirely too much of MatheMagic's operations when the program is evaluating a formula. Such displays are, of course, nice when operations are stepwise, but only slow down the program under most circumstances. Smoothness is particularly lacking when "ask" variables are buried inside stored formulas that get called by other formulas. It takes some practice to get to a final answer here.

It would also be nice if the program had provisions for using a wider variety of names for formulas and variables, particularly in the implementation on the IBM-PC. The IBM-PC contains a complete Greek alphabet in its "upper" 128 characters. Many established statistical and engineering formulas are these characters. It would be nice to be able to use them directly inside MatheMagic. Support of lower case letters, moreover, would make multiword variable names more readable.

These are not, however, deficiencies in MatheMagic so much as they are deficiencies in the IBM-PC's MicroSoft BASIC Interpreter. Once the program is compiled, ISM will be free to make other changes to the program, perhaps including the addition of some character options for naming. That compilation will also speed the program up considerably.

Beyond these deficiencies, however, objections to MatheMagic are largely a matter of taste. The program is written to serve a particular set of needs. It serves them rather well. If you have that need, MatheMagic will be a valuable addition to your software library. If you don't have the need, then no amount of complaining will suit the program to your needs.

This brings me to my last complaint, which is a matter of taste. Merchandising MatheMagic as the ultimate calculator seems to be something of a mistake. Every time I describe MatheMagic to a microcomputer user as software that turns a microcomputer into a calculator, I get laughter. "Why," I am asked, "would anybody spend $5,000 on a computer to turn it into a $20 (or even a $200) calculator." They are, of course, right. I wouldn't do it either. The problem is that MatheMagic is more than a calculator. It's really a "Formula Evaluator." ISM needs, I think, to refine the vocabulary with which they describe their product if it is to reach the success of which it is capable.

MatheMagic versus the spreadsheets

It is important that a review of MatheMagic compare the program with programming languages like BASIC because it is, in some sense, an easy-to-use mathematical applications programming language. As was noted early in this article, however, MatheMagic is hardly unique in this respect. To succeed, MatheMagic is going to have to compete with, and differentiate itself from, VisiCalc and the many other VisiClones that lead the microcomputer software market.

Despite the fact that MatheMagic and the spreadsheets are basically doing the same thing, e.g., performing calculations on numbers, this differentiation really isn't very difficult. Spreadsheets excel in tasks that involve working with arrays. They are at their best when a large number of related numbers vary according to a limited number of assumptions. They cannot, as a rule, do anything with arrays that one might like. Indeed they are generally rather weak as matrix manipulators, but for almost any kind of manipulation or modeling task that involves large numbers of related numbers, they are hard to beat.

It is this characteristic of spreadsheets that have made them particularly popular in business. VisiCalc is a great tool for predicting the long-term impact of a small drop in stock prices. SuperCalc is a fantastic program for manipulating a budget until it works. But whatever the talents of spreadsheets in dealing with tables of numbers, they are not at their best when it comes to calculating complex formulas that involve the manipulation of large groups of numbers toward a particular end result.

Electronic spreadsheets are predicated on the metaphor of the accountant's spreadsheet. The central unit is the cell and it is the contents of cells that are manipulated within the spreadsheet program. To write a complex formula on a spreadsheet, one must think in terms of cells rather than the variable name of which the formula is actually composed. Variable names must be thought of in terms of cells; translated into cell names. The potential for error increases as a result.

As with BASIC, moreover, placement is important in spreadsheets. If one calculation depends on another, the misplacement of the antecedent formula on the spreadsheet will result in erroneous results that can only be sorted out, if it can be sorted out at all, by the recalculation of the entire spreadsheet.

Complex formulas and recursion are, on the other hand, old stuff for MatheMagic, which takes the language of formulas as its metaphor. Cells are the object in a spreadsheet, with one cell capable of calling the contents of another. The object in MatheMagic, on the other hand, is formulas which are represented by whatever words the user chooses. This allows formulas to be entered into MatheMagic with little or no translation. Mistakes are harder to make, and the fixing of mistakes is much easier.

MatheMagic is not a wonder in the art of working on arrays. It can work with arrays, but it does so with somewhat more difficulty. The user must be skilled to get it right. The user will also find the display of MatheMagic, which is designed for formulas, somewhat less satisfying than that of a spreadsheet for working with arrays. A spreadsheet displays the array in a way that allows the user to see what is going on. MatheMagic does not.

A comparison of MatheMagic to spreadsheet programs reveals two strongly contrasting approaches to calculation, each of which is useful in different contexts. Indeed, their respective strengths and weaknesses are strongly complementary. Each is strong where the other is weak. Just as I would not sell BASIC expecting MatheMagic to replace it, I would not sell my copy of VisiCalc expecting MatheMagic to replace it.

The value of MatheMagic is not in its replacement value. It is in its ability to evaluate complex formulas and use those formulas in a modular way. If you need to write quick and dirty programs to translate things from one format to another, BASIC is invaluable. If you need to work with arrays of numbers, a spreadsheet is invaluable. If you need to work with clusters of complex formulas, MatheMagic is invaluable. If you need all three, you should own all three. Each does its respective job far better than any of the others. I put myself in this last group. I have found MatheMagic a valuable component of my software library. ■

New Versions

MAIL80 1.1

Pegasus-Basis, Inc.
670 International Pkwy Suite 100
Richardson, TX 75081

This new release will run on virtually all CP/M* (and MP/M*) microcomputers including the HP-125 and Vector 2600.
New features include:
1) Telephone extension and a 28 character comment line have been added to the customer/client record. 2) The user can specify which drive or drives the data files are on. 3) In addition to mailing labels, the system now prints file and Rolodex cards.

C-Food Smorgasbord 1.4

Lifeboat Associates
1651 Third Avenue
New York, NY 10028

The TIP utility program has been improved to accept hexadecimal values when you are typing control sequences; tiptest.c and testl.bat are added as a small example of how to use the tip package.

Lattice "C" 1.04

Lifeboat Associates
1651 Third Avenue
New York, NY 10028

1) Includes a new utility program, Object Module Disassembler; for programmers who wish to debug C modules at the machine code level, the OMD program provides a listing of the machines language instructions, generated for a particular C source program. 2) The -d compile time option for LC1 has been implemented in this version of the compiler, and differs from its description in the manual in only minor details. The -i option is completely new. The -id reads all # include files from drive "d", where "d" is a single alphabetic character, either upper or lower case, specifying a disk drive ("a" for A:, etc.).

Feature

Accessing the MP/M Operating System From Within dBASE

by Dr. Howard Vigorita

SINCE ACQUIRING BOTH dBASE AND MP/M ABOUT one year ago, I have been stymied in my quest to access the wealth of MP/M facilities from within the menu driven applications package I have developed with dBASE. Such features as system clock access, process scheduling, password protection, etc., have been totally inaccessible because of differences between the CP/M and MP/M implementation of SUBMIT files which render the dBASE 'QUIT TO' mechanism inoperative. However, this may have been a blessing in disguise. An article appearing in the July, 1982 issue of *Lifelines* has suggested an approach vastly superior to any SUBMIT-based technique.

My technique involves a two-stop process. First a short program loading interface is installed into the dBASE sorting area (anywhere between address A400 hex and BDOS). The actual installation of the loader can be done using DDT or with the forthcoming dBASE version 2.4's LOAD command at run time. Second a dBASE command file is used to poke to the loader's command line buffer after which the loader routine is called as a subroutine.

The loading interface is quite trivial under MP/M II. The operating system provides an XDOS function #150 which sends a command line to the MP/M Command Line Interpreter (CLI, nee CCP under CP/M). The routine, complete with housekeeping calls, can be found in Digital Research's MP/M II, Application Note #6. I merely ORG'd it to begin at 42900d with the command line at 43000d, well within the dBASE sort buffer at easy to remember round addresses. I chose a location well above A400h (=39360d) so as to leave a small area for poking shorter sequences such as the list detach calls which I use at the end of print sequences.

A companion dBASE command file provides a menu of common commands together with a run time input command line for more knowledgeable users. It converts the requested command, one character at a time, into ASCII decimal, while poking the result into the loader's command line. Note that in order to provide the full character set of which my terminal is capable, the ASCII string table had to be formulated in two concatenated parts with each part containing only one of the two quote characters so that the other quote character could serve as the string delimiter.

The advantage inherent in this loader technique is that almost any command line capable of input when the operating system prompt is displayed can be run from within dBASE without having to reload dBASE on return. An even greater advantage is that almost any COM or PRL file can be run without having to reassemble it to a new load location. I say "almost" because a SUBMIT command doesn't work and changing the default user or drive can only be done by poking the first character of the command line structure.

The major limitation of this technique is that the CLI must be able to find a free memory area in which to load the program requested by the command line. In the case of a PRL (page relocatable) file, the CLI will find the smallest free memory segment that the PRL file will fit into, automatically relocate it there, and then execute it. Since all of the usual CP/M built in commands and MP/M utilities are supplied as PRL files with MP/M, no limitation will be experienced on most MP/M systems. To run a COM file, however, the CLI needs an available absolute memory segment large enough to hold the program. On my 4 bank 2 console Altos system, I can only execute WordStar from within dBASE on one console if the other console is inactive.

Now if I could just get my hands on a dBASE native code compiler. . . .

```
;  Send CLI.asm
;  Assembly language fragment to send a command to the MP/M II
;  Command Line Interpreter
;
           ORG     000h
Base       EQU     $
BDos       EQU     Base + 0005h
;
;  XDos function equate table:
SetPriority        EQU          145
AttachConsole      EQU          146
AssignConsole      EQU          149
SendCliCommand     EQU          150
GetConsoleNum      EQU          143
;
;  Program body:
           ORG     42900              ; set up an entry point
                                      ; @42900
           JMP     42923
           ORG     42923              ; assemble so command
                                      ; line at 43000
           LXI     h,0000h            ; save the old stack
                                      ; pointer
           DAD     sp
           SHLD    OldSp
           LXI     sp,Stack + 0016h   ; set up a new stack
                                      ; pointer
           MVI     e,190
           MVI     c,SetPriority      ; raise console priority
           CALL    BDos
           MVI     c,GetConsoleNum    ; get & fill in console #
           CALL    BDos
           STA     AssignPB
           STA     CliCommand + 1
           LXI     d,AssignPB         ;
           MVI     c,AssignConsole    ; assign console to CLI
           CALL    BDos
           INR     a
           JZ      Finish             ; exit if assignment fails
           LXI     d,CliCommand       ; otherwise,
           MVI     c,SendCliCommand   ; execute command
           CALL    BDos
           MVI     c,AttachConsole    ; reclaim the console
```

(continued on next page)

```
            CALL    BDos
            MVI     e,200
            MVI     c,SetPriority        ; restore default priority
            CALL    BDos
Finish      LHLD    OldSp                ; restore old stack pointer
            SPHL                         ; then return
            RET
;
; Data and storage areas
;
AssignPB:
            DB      $—$                  ; console number
            DB      'cli       '         ; Command Line
                                         ; Interpreter name
            DB      0                    ; null end of name marker
;
CliCommand:
            DB      0                    ; default disk and user
            DB      $—$                  ; console number
                                         ; 50 byte command line:
            DB      '
            DB      0                    ; terminate with a null
Stack       DS      016h
OldSp       DS      02h
            END
```

```
*
* SendCLI.cmd
* sub menu to send commands to the MP/M II operating system via its
* XDOS function #150 (Send CLI) facility. This routine expects to find an
* assembly language routine installed in dBASE at location 42900d with
* the command buffer at 43000d.
*
* Note: dBASE version prior to 2.3C may require a dummy argument
* with CALL statement
CLEAR
*
* ASCII character set following space character
STORE '!"#$%&' +;
" '( )*+,-./0123456789:;<=>?@ABCDEFGHIJKLM
NOPQRSTUVWXYZ[ ]" + ;
"↑<'abcdefghijklmnopqrstuvwxyz{:}" TO ASCII
*
STORE 42900 TO Loader
STORE 43000 TO cliBuffer
STORE '                                   ' ;
      TO MCommand
STORE 0 TO Choice
STORE F TO Done
DO WHILE .not.Done
  @ 0,25 SAY ' Operating System Command Menu '
  @ 3,15 SAY ' 0. Return to the main menu'
  @ 5,15 SAY ' 1. Display short directory — all files'
  @ 6,15 SAY ' 2.                          — database files'
  @ 7,15 SAY ' 3.                          — command files'
  @ 8,15 SAY ' 4. Display extended directory — all files'
  @ 9,15 SAY ' 5.                             — database files'
  @10,15 SAY ' 6.                             — command files'
  @12,15 SAY ' 7. Report of disk drive free space'
  @14,15 SAY ' 8. Erase — any files of type "BAK" '
  @15,15 SAY ' 9.       — any files beginning with "Temp" '
  @17,15 SAY ' 10. Turn password protection off'
  @18,15 SAY ' 11. Turn password protection on'
  @20,15 SAY ' 12. Compose your own custom tailored command'
  @23,28 SAY 'Enter your choice ' GET Choice PICTURE '##'
READ
*
ERASE
DO CASE
CASE Choice = 1
  STORE 'DIR *. *[SYS]' TO Command
CASE Choice = 2
  STORE 'DIR * . DBF[SYS]' TO Command
CASE Choice = 3
  STORE 'DIR * . CMD[SYS]' TO Command
CASE Choice = 4
  STORE 'SDIR' TO Command
CASE Choice = 5
  STORE 'SDIR * . DBF' TO Command
CASE Choice = 6
  STORE 'SDIR * . CMD' TO Command
CASE Choice = 7
  STORE 'STAT' TO Command
CASE Choice = 8
  STORE 'ERAQ * . BAK' TO Command
CASE Choice = 9
  STORE 'ERAQ TEMP* * ' TO Command
CASE Choice = 10
  STORE 'SET [PROTECT = OFF]' TO Command
CASE Choice = 11
  STORE 'SET [PROTECT = ON]' TO Command
CASE Choice = 12
  @ 10,2 SAY 'Enter systems command ' GET MCommand
  READ
  STORE MCommand TO Command
  ERASE
OTHERWISE
  STORE T TO Done
  loop
ENDCASE
*
* poke command one character at a time into command buffer
STORE 0 TO Cnt
STORE LEN(TRIM(Command) ) TO Length
DO WHILE Cnt < Length
  * note that default will be space character
  POKE cliBuffer + Cnt, 32 + @( $(Command,Cnt + 1,1),ASCII )
  STORE Cnt + 1 TO Cnt
ENDDO WHILE Cnt < Length
*
* terminate the command with a NULL and call as subroutine
POKE cliBuffer + Length, 0
SET CALL TO Loader
CALL
*
* subroutine returns here and continues
  WAIT
  ERASE
ENDDO WHILE .not.Done
RETURN
```

Renew

We're looking forward to hearing from any of you August subscribers who haven't called or written. If your subscription started with the August '82 issue you should have received a letter from us, urging you to renew. You can see that Lifelines/The Software Magazine has given you value this past year and we're expecting your support again. Don't delay! Send your check right away or get out your VISA or MasterCard and call Lifelines/The Software Magazine Subscription Dept. at (212) 722-1700. The address is 1651 Third Avenue, New York, NY 10028.

Stockvue Reviewed

by Robert P. VanNatta

IF YOU ARE TRYING TO MAKE a buck in the stock or stock options market, Star Value Software of 12218 Scribe Dr., Austin, Texas 78759 has released a stock and option analysis program that might be of some interest.

If you are trying to make a buck in the stock or stock options market, Star Value Software of 12218 Scribe Dr., Austin, Texas 78759 has released a stock and option analysis program that might be of some interest.

Stockvue, as it is billed, is effectively a dedicated spreadsheet suitable for analyzing stock and stock option trades. The user can enter such information as contemplated trade dates, interest rates, sticking price and the like; Stockvue will compute the percentage gain or loss, the gain or loss in dollars, and the effective return on your money.

Versions are advertised as being available for the TRS-80 Models I and III(TRSDOS), the IBM-PC(PCDOS), and CP/M. The version examined for this review was the CP/M variant. It is compiled in Microsoft BASIC. The program file is 40K in size and is said to require at least a 56K system to run (and perhaps more). Under CP/M, a 24×80 terminal is required. The program only needs 11 keys, most of which are user definable, so it can be described as reasonably hardware independent. A terminal capable of highlighting or generating inverse video makes for a little more pleasant display but is not required.

The terminal codes are stored in a disk file, and suggested installation files are provided for the ADM3a, Adds Viewpoint, Heath H19 and ACT-IV terminals. If your terminal is not one of those four (and whose is?), you must struggle through an awkward but usable terminal installation program and enter your terminal codes one at a time. This writer brought the program up on a TRS-80 Model 16 using ATON CP/M (which emulates a SOROC 120), Lifeboat CP/M (which emulates both an ADM3a and an ADM 31) and Pickles and Trout CP/M (which doesn't emulate anything). No incompatibilities were observed. Although I must grudgingly admit that the installation routines worked, there was an unmistakable hacking noise emanating from my computer throughout the installation process. The installation program is 39K of compiled BASIC, which has me wondering whether it should have an ease of use rating higher or lower than one might apply to DDT.COM.

The documentation is 47 pages long and is excellent. The price is advertised at $189.00 postpaid.

Audience

The usefulness of this program is probably limited to those who trade stock options and who, in addition, are convinced that they can never learn to run Visicalc (or a Calc-clone).

Functionally, Stockvue is a spreadsheet dedicated to evaluating stocks and stock options. Use of the program consists of moving the cursor around the screen in a spreadsheet fashion and entering appropriate values. Recalculations are automatic. As such, it does nothing that cannot be constructed on a Calc-clone. The trade-off is simple. You use a spreadsheet and build your own model, or you buy Stockvue and use their model. The advantage of rolling your own is, of course, the flexibility in being able to change it.

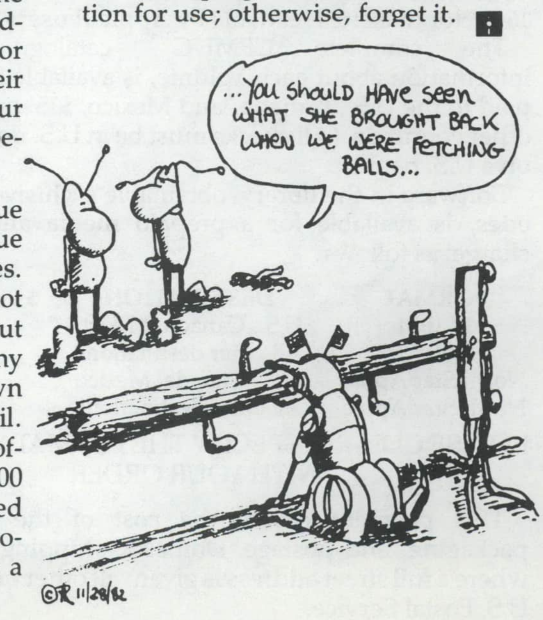
My biggest criticisms of Stockvue relate to its non-features. Stockvue has neither file nor printer routines. This effectively means you cannot store or recall any information about any of your calculations in any fashion, except by copying it down on an old envelope with a pencil. This writer for one, finds it a bit offensive to sit down behind a $5000 computer which in turn is plugged into a $2000 printer, and find a program that is incapable of making a permanent record of my work.

My immediate impression upon first loading this program was that it was something that had been downloaded from a cassette driven Radio shack Model I. Subsequent correspondence with the authors confirmed my suspicions (only to the extent that they acknowledge that it was written on TRSDOS based Microsoft BASIC and lately downloaded to CP/M and compiled with the Microsoft compiler).

I am told that the authors are considering a new version which may have disk or printer capabilities (or both), so if this is a relevant consideration to a prospective user, it might pay to check for a new version.

Conclusions

I was unable to uncover any glaring bugs in the program. It appears to perform exactly as documented; however, this writer has rarely seen more code (40K) that did less. The limitation of Stockvue is in its very narrow audience. If you: 1) are a stock option trader (it won't handle commodities); 2) are unwilling or unable to build your own model on a spreadsheet; and 3) have an adequate supply of pencils and used envelopes for recording your calculations, this program merits consideration for use; otherwise, forget it. ■



Users Group Corner

New York Amateur Computer Club
P.O. Box 106 Church St. Station
New York, NY 10008

The NYACC was founded in 1976 and is the largest computer club in New York City supporting all types of microcomputers and many users groups. It is a nonprofit group that has been extensively involved in the cataloging and distributing of public domain software and sponsors the SIG/M along with the Amateur Computer Group of New Jersey (ACGNJ). The club has a hot line with a recorded message announcing the date and location of all of the meetings in the New York City area. The NYACC is editing and distributing PC/Blue: The Public Domain Software Library for PC-DOS, along with publishing a catalog. It also publishes catalogs (seven, currently) for CPMUG and SIG/M software. The NYACC publishes a newsletter in which it announces new public domain software. Dues are $15 per year.

The catalogs are available by mail from the NYACC. Each catalog is $10 with shipping to North America and $15 with overseas airmail. All orders must be prepaid in U.S. funds.

The diskettes are available from many local computer clubs. They are also available by mail. The CP/MUG volumes are available from the CP/M User's Group (see below). The SIG/M volumes are distributed on 8″ by the SIG/M at Box 97, Iselin, N.J. 08830.

CPMUG
The CP/M Users Group
1651 Third Avenue
New York, New York 10028

As of March, 1983, the Library contained nearly 100 Volumes of software ranging from language interpreters, editors, and assemblers in full source code, to games and pictures. This software is available on 8″ single density CP/M-80 and SB-80™ diskettes, on North Star diskettes readable by users of double-density CP/M-80 1.4, double-density CP/M-80 2.2, or quad capacity CP/M-80 2.2, or on 16-sector diskettes readable by Apple II users.

The complete CPMUG™ catalog, providing information about each Volume, is available for $10. prepaid to the U.S., Canada, and Mexico, $15. prepaid to all other countries. (All checks must be in U.S. dollars drawn on a U.S. bank).

Software in the library, obtainable exclusively on diskettes, is available for a prepaid media and handling charge, as follows:

FORMAT	DESTINATION	$ PER VOLUME
8″ IBM	U.S., Canada, Mexico	$13.
8″ IBM	All other destinations	$17.
North Star/Apple	U.S., Canada, Mexico	$18.
North Star/Apple	All other destinations	$21.

PLEASE CLEARLY SPECIFY THE FORMAT YOU WANT WITH YOUR ORDER

This payment covers the cost of the diskette(s), packaging, and postage. Domestic shipping is via UPS where a full street address is given; all other orders are via U.S. Postal Service.

Please note: CPMUG has no telephone facilities; all correspondence must be via mail.

Future additions to the Catalogs, as well as abstracts of all new CPMUG releases will be published in *Lifelines™ The Software Magazine™*.

Please place requests for Library software, orders for *Lifelines*, and comments on any other matter, on separate notes in order to expedite replies. (Checks must be in U.S. currency drawn on a U.S. bank). Although *Lifelines* and CPMUG are in the same building, they are not part of the company.

Members receiving the material are reminded that software contributions are necessary if the exchange program is to prosper. Software contributions are gladly received for inclusion into the Library with the understanding that the contributor is authorized to make the material available to others for their individual noncommercial use.

Software should be accompanied by sufficient documentation in the form of internal comments or accompanying *.DOC file to permit the material to be applied and/or modified. Where appropriate, the documentation should describe any supporting software (interpreter, memory, clock, etc.) necessary to utilize the routine. For your convenience, a comprehensive submittal form is included on most distributed diskettes. Contributors are invited to request any Library Volume in exchange for the one submitted.

Are you ***missing*** something?

Look inside, and find out how you can keep in touch with software changes. With update analyses, bug reports, patches, tips and in-depth reviews, we provide timely news on your software and how you can get the most out of it.

Thunderclock Routine

by David W. Walker

This Program is for the Thunderclock routine written up in June 1983 Lifelines p.31

```
;                       CLOCK
;
;Program to read a Thunderclock card, in an Apple II with
;a Z80 Softcard, and print the date and time to the
;console, in the format
;                 THU NOV 11 8:35:24 AM
;
;                by D. W. Walker 11 Nov 1982
;
F3DE =      ZCARD      EQU   0F3DEH     ; address of
                                        ; Z80 card
F3D0 =      VEC65      EQU   0F3D0H     ; pass 6502
                                        ; subroutine
                                        ; address
F045 =      ACC65      EQU   0F045H     ; pass 6502
                                        ; accumulator
;
F200 =      BUFFER     EQU   0F200H     ; date/time
                                        ; string buffer
0009 =      PUTSTR     EQU   9          ; CP/M "print
                                        ; string" code
0005 =      BDOS       EQU   5          ; CP/M BDOS
                                        ; jump
;
0100                   ORG   100H       ; vector
                                        address
;
;           Find card: check first three bytes of each slot
;           until Thunderclock found or all slots checked
;
0100 25E8              MVI   H,0E8H     ; first slot will
                                        ; be 7
0102 2E02   NXTSLT:    MVI   L,02H      ; address =
                                        ; En02
0104 25                DCR   H
0105 7C                MOV   A,H        ; check card
                                        ; high byte
0106 FEE0              CPI   0E0H       ; slot 0, no
                                        ; clock found
0108 C8                RZ               ; so return
;
0109 7E                MOV   A,M        ; get byte at
                                        ; En02
010A FE28              CPI   28H        ; third byte of
                                        ; clock
                                        ; firmware
010C C20201            JNZ   NXTSLT     ; no match, try
                                        ; next slot
010F 2D                DCR   L          ; point to
                                        ; En01
0110 7E                MOV   A,M        ; get byte
0111 FE78              CPI   78H        ; second byte
                                        ; of clock
0113 C20201            JNZ   NXTSLT
0116 2D                DCR   L          ; point to
                                        ; En00
0117 7E                MOV   A,M
0118 FE08              CPI   08H        ; first byte of
                                        ; clock
011A C20201            JNZ   NXTSLT
;
;           Here, clock has been found, at the slot
;           addressed by H,L
;
011D 7C                MOV   A,H
011E D620              SUI   20H        ; change En
                                        ; to Cn
0120 67                MOV   H,A
;
0121 3E25              MVI   A,'%'      ; control char
                                        ; for clock
0123 3245F0            STA   ACC65      ; pass to 6502
                                        ; accumulator
0126 2E0B              MVI   L,0BH      ; point H,L to
                                        ; clock write
                                        ; routine
128 22DOF3             SHLD  VEC65      ; pass
                                        ; subroutine
                                        ; address
012B E5                PUSH  H          ; save clock
                                        ; address
012C 2ADEF3            LHLD  ZCARD      ; get address
                                        ; of Z80 card
012F 77                MOV   M,A        ; write to it
;
;           Control passes to 6502, to execute clock write
;           routine at $Cn0B, with '%' in accumulator.
;           That sets up the clock to return the date and
;           time in the specified format
;
0130 E1                POP   H          ; recover
                                        ; clock
                                        ; address
0131 2E08              MVI   L,08H      ; point to
                                        ; clock read
                                        ; routine
0133 22DOF3            SHLD  VEC65      ; pass
                                        ; address to
                                        ; 6502
0136 2ADEF3            LHLD  ZCARD      ; get address
                                        ; of Z80 card
0139 77                MOV   M,A        ; write to it
;
;           Control passes to 6502, to execute clock read
;           routine at $Cn08. That routine leaves the time
;           string in the buffer at $200 (F200H).
;           Now, send that string to the console.
;
013A 3E24              MVI   A,'$'      ; string
                                        ; terminator
013C 2117F2            LXI   H,BUFFER+17H ; end of string
013F 77                MOV   M,A        ; stuff
                                        ; terminator
0140 1101F2            LXI   D,BUFFER+1 ; point to start
                                        ; of string
0143 0E09              MVI   C,PUTSTR   ; "print
                                        ; string" code
0145 CD0500            CALL  BDOS       ; print the
                                        ; string
0148 C9                RET              ; and return to
                                        ; CCP
0149                   END
```

Feature

Demonstrating the High Precision Integer Math Library: Some Interesting Math Programs

by Thomas Hill

IN A PREVIOUS ARTICLE I PRESENTED A LIBRARY OF routines designed to implement high precision integer math functions. In this article I will show how to use the library to create some dedicated math programs to calculate prime numbers, greatest common divisors, least common multiples, and a pseudo-random number generator.

Review

In the last article I presented the source code for the simple mid-level arithmetic functions of ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION, MODULUS, and SQUARE ROOT. By using these modules in a structured fashion, we may develop further applications. Please refer the the accompanying program listings as I discuss each of the following applications.

Generating prime numbers

Mankind has had a fascination with prime numbers for centuries. The Greek mathematician Eratosthenes developed his famous "sieve" (known to all computer progammers due to its popular use as a 'benchmark') before 200 B.C. To review, a prime number is an integer which has only two trivial divisors: the number one (1) and itself. Thus two, three, five, and seven are all prime numbers. (One, which meets all the requirements of being a prime number, is not included in the accepted list of primes, oddly enough.) A program to generate prime numbers by 'brute force' may be written using the following logic:

1. Since we know two (2) and three (3) are prime, print them.
2. Set our test value T equal to three.
3. Add two (2) to the test value, T. (We add 2 here to keep the value of T odd, since we need not be concerned with even values, which are obviously not prime, being divisible by two.)
4. Set our test divisor equal to three. (The smallest non-even number.)
5. Divide T by D.
6. If the remainder of the division in step 5 is equal to zero, then the value T cannot be prime, since it was divided by a number other than itself. We therefore return to step 3, selecting a new value.
7. If the remainder of the division in step 5 is not zero, and the value of D is not equal to T, then we add 2 to D and return to step 5.
8. If our test divisor equals T then T is prime. (Why?)
9. Print the value of T and return to step 3.

(Excuse the question in step 8 above. This was extracted from a test in programming for a class I once taught.) Listing 1 is the source for a program designed around the logic presented. Note the use of external references to access the High Precision (H.P.) library. This removes much of the programming load, allowing us to concentrate on the task at hand. If you will study the listing, you will see that I have included code to allow the user to select the starting point for the prime number generation. This allows us to look for primes beginning at (say) 123,456,789. Checks are included at the input of a starting value to detect even input values and to make the input odd by adding one to it before starting the prime search. This will prevent starting something doomed to failure. A check is also made after printing each prime value for a CONTROL-C abort at the keyboard. This provides an escape route back to CP/M besides the RESET button.

Included in the library listing in previously published in *Lifelines* (June, 1983) is an improved version of this algorithm, adapted from Knuth's "Art of Computer Programming." If we were to use this 'built-in' function, rather than write one of our own, it would result in a program which accepts the input, passes an input pointer to the PRIME? module, and receives a YES/NO answer from the module. If the answer is yes, then we print the value.

Least common multiple and greatest common divisor

Listings 2 and 3 are programs to find the Least Common Multiple (LCM) and the Greatest Common Divisor (GCD) respectively of two input values. These two results are of use in certain areas of number theory, and also (in an extended form) in cryptographic theory. The GCD algorithm used is extracted from Knuth, Volume 2, and is presented herein:

1. Let A,B be two integers.
2. Let R=A−B * INT(A / B), where the INT function returns the INTeger portion of the division.
3. If R=0 then the GCD=B, terminate the program.
4. Let A <-- B
5. Let B <-- R
6. Goto step 2.

The INT function used in the algorithm is easily handled by using the truncating form of the DIV module. The general form of the GCD program follows that of the prime number generator: Accept the input values, find the GCD, and loop for more input. In this case we terminate when a CTRL-C is entered as an input value.

The LCM program uses the following algorithm (from Knuth again):

1. Let A,B be two integers.
2. $LCM(A,B) = \frac{A * B}{GCD(A,B)}$

Notice that this program takes advantage of the GCD program. This means that we must rewrite the GCD program to operate as a callable subroutine, with values passed as pointers and the result returned as a pointer. After rewriting the GCD in this fashion, it can be added to our library and treated as any of the other modules in future programs. The LCM program is also rewritten in a like manner and added to the library. Listing 3 presents the LCM program, using the GCD module included in the HP math library.

Permutations and combinations

These two programs make use of the library module NFACT, which produces the factorial of its integer input. The formulas involved are:

Permutations: $P(n,m) = \frac{n!}{(n-m)!}$

Combinations: $C(n,m) = \frac{n!}{n!(n-m)!} = \frac{P(n,m)}{m!}$

where "n!" and "m!" are the factorials of "n" and "m" respectively. The factorial of a number may be computed by forming the product:

n! = (n) * (n-1) * (n-2) * . . . . * (1)

Listings 4 and 5 are the sources for the Permutation and Combination programs. Because of the use of the library functions, they are extremely short, and are very easy to debug since we know that the library routines have all been checked for proper operation.

Random number generator

The final library module we will discuss is a random number generator. This particular generator is adapted from Knuth, Volume 2. (Any competent programmer should have a set of Knuth's "Art of Computer Programming" on hand. There is a gold mine of material and techniques contained therein.) The generator is called a "linear congruential generator." Its general form is:

X(n+1) = (a * X(n) + c) MOD m, (n >= 0)

The modulus "m" should be relatively prime to the parameter "a." In the case of the generator used here, I have selected "m" to be:

m = (2↑127) − 1

This selection assures us of the maximum period before repetition begins. Further constraints governing the selection of "a" and "c" are:

1. "c" must be relatively prime to m,
2. (a-1) must be a multiple of "p," for every prime "p" dividing "m"
3. (a-1) is a multiple of 4, if "m" is a multiple of 4.

For this version I have chosen the following values for "a" and "c":

a = c = 2↑16+1

This results in the following values:

m = 170,141,183,460,469,231,731,687,303,715,884,105,728
a = c = 65,537

The theoretical period of this generator is then the value (m−1).

Before we go further, I should define some of the terms used in the discussion above; "relatively prime" implies that the Greatest Common Divisor of the two arguments is 1. Note that this DOES NOT say that the values are prime, although they may be. It merely indicates that they have no common divisors. For example, 25 and 4 are relatively prime, but neither is prime. The "period" of a random number generator is the number of values which may be produced before the sequence of numbers is repeated. Thus in the sequence:

2,4,5,7,8,3,2,4,5,7,8,3,2

the period is 6, because at the seventh value the sequence begins to repeat. Our random number generator has a theoretical period of (2↑127)−1, which is a rather large number (although it is only a fraction of the maximum value which the HP math library will handle.) Note that this figure for m was chosen somewhat arbitrarily, and could be increased greatly if desired.

Listing 6 is the module designed to be added to the HP library. Note how short it is. The "randomness" of the values produced are greatly dependent upon the selection of "m," "a" and "c". The module allows the user to "seed" the generator by passing a non-zero address in the DE register. If the contents of DE are not zero, then the value pointed to by DE is used as the starting value for the term "X(n)" in the generator formula. If the DE register is zero, then the last computed value for the seed is used to generate the next value. This makes "randomizing" the generator easy. One need merely pass an arbitrary address at the first call to the generator to seed it. One method of generating this arbitary address could utilize two readings of the refresh register of the Z80 microprocessor to build a two byte value. Alternatively, one could use two characters input by the user to build a value in the DE register.

What guarantee do we have that the module in listing 6 actually produces random numbers? Well, since they are generated by computer, they are not actually RANDOM, because we may re-generate them by starting over with an identical seed. But, by subjecting the values produced to various statistical tests, we may state certain things about the projected randomness of the method. These tests are many and varied, and are not suitable for inclusion here. Suffice it to say that the generator shown here has been subjected to several of the more important tests outline by Knuth, including the "spectral" test (which seems to be the best of the lot), and it has passed with flying colors. The numbers produced by this generator may be taken to be random, in all but the most exacting situations.

Listing 7 is a program which uses the random number generator to print a list of random values on the console. By using the CTRL-P printer toggle when executing the program, you may get a hardcopy listing to examine at your leisure.

Final words

Well, I hope you found the material presented here interesting. Next time, I will present a program which incorporates much of the HP math library into an 8-function calculator which will accept algebraic expressions using up to ten parentheses. It will provide the functions of addition, subtraction, division, multiplication, raise to a power, square root, factorial, and modulus. Included with it will be a parser module which accepts a parenthesized numeric expression and converts it into a Polish operand/operator stack structure before computing the result.

(continued on next page)

Listing 1

PRIME GENERATOR

```
; This program will generate prime numbers, starting at the
; value input by the operator. Program execution is stopped by the
; detection of a CTRL-C at the console after the output of a value.
;
; The program utilizes the following modules from the HPMATH
; library:

        EXTRN   DIVM        ; Modulus routine
        EXTRN   DIV         ; general division
                            ; routine
        EXTRN   HPINPUT     ; input routine
        EXTRN   HPOUT2      ; output routine,
                            ; version 2
        EXTRN   MOOV        ; multiple byte move
                            ; routine
        EXTRN   AD1         ; general purpose
                            ; addition
        EXTRN   PARE        ; compare two
                            ; multibyte values

; the following equates are used:

CPM     EQU     0
BDOS    EQU     CPM+5
CONST   EQU     11          ; status check at console
CONOUT  EQU     02          ; console output
PRTBUF  EQU     09          ; print string function

CR      EQU     0DH         ; return
LF      EQU     0AH         ; line feed
BELL    EQU     07H         ; terminal bell

; begin the program in the code segment
        CSEG

PRIME1: LXI     D,SIGNON
        MVI     C,PRTBUF
        CALL    BDOS        ; tell world we are here
        LXI     D,IN$VALUE
        CALL    HPINPUT     ; get the starting value

; move the input value to internal storage
        LXI     H,T
        LXI     D,IN$VALUE
        CALL    MOOV

; check value for evenness.
        LXI     H,TWO
        LXI     D,T
        CALL    DIVM        ; use modulo and . .
        LDA     T
        ORA     A           ; check remainder for
                            ; zero result
        JNZ     LOOP1       ; not even, start
                            ; computing

; input value is even, add one to it for odd

        LXI     H,ONE
        LXI     D,IN$VALUE
        CALL    AD1

; begin prime generation loop here.
; Continue until aborted from keyboard.

LOOP1:  LXI     D,THREE
        LXI     H,DV        ; divisor set to three
                            ; for start
        CALL    MOOV

; form T modulo DV
LOOP2:  LXI     H,T
        LXI     D,IN$VALUE  ; refresh value in
                            ; working storage
        CALL    MOOV
        LXI     D,T
        LXI     H,DV
        CALL    DIVM

; check remainder
        LDA     T
        ORA     A           ; is it zero?
        JZ      NEXT1       ; yep, T is not prime,
                            ; get next value

; if remainder is non-zero, increment divisor and check for
; equal to value. Note that divisor is also incremented by 2.
        LXI     H,TWO
        LXI     D,DV
        CALL    AD1

; check for DV equal to current value
        LXI     H,DV
        LXI     D,IN$VALUE
        CALL    PARE
        JNZ     LOOP2       ; not done yet,
                            ; try another divisor

; DV and T are equal, T must be prime.
; print it
        LXI     H,T
        LXI     D,IN$VALUE
        CALL    MOOV        ; update value in
                            ; working storage

        MVI     E,BELL
        MVI     C,CONOUT
        CALL    BDOS        ; ring the bell
        LXI     D,T
        CALL    HPOUT2      ; print the value.

; Note that HPOUT2 destroys the value being printed. So it must be
; saved before outputting. This is why we keep a copy of the input
; value (and successive values) in the integer IN$VALUE.
        MVI     C,CONST
        CALL    BDOS        check for abort
        ORA     A
        JZ      NEXT1       ; no abort, next value
        JMP     CPM         ; kill it.

NEXT1:  LXI     H,TWO
        LXI     D,IN$VALUE
        CALL    AD1         ; increment last value
                            ; by two
        JMP     LOOP1       ; continue the task.

; here is the data segment
        DSEG

SIGNON: DB      CR,LF,'Prime Number Generator Program.',
                CR,LF
        DB      'Version 1.0 -- Brute Force Method',CR,LF,LF
        DB      'Enter starting value, terminated with equals sign
                (=)'
        DB      CR,LF,'->$'

; integer constants
ONE:    DB      1,1
TWO:    DB      1,2
THREE:  DB      1,3

; storage
IN$VALUE:
        DS      128
T:      DS      128
DV:     DS      128

        END
```

Listing 2

LEAST COMMON MULTIPLE

```
; This program computes the Least Common Multiple (LCM) of two
; integers. The algorithm used is:
;
; Let A,B be integers.
;
;                   A * B
; LCM(A,B) = --------------
;                 GCD(A,B)
;
; External modules from HP math library:

        EXTRN   MULT            ; multiplication
        EXTRN   DIV             ; truncated division
        EXTRN   GCD             ; greatest common divisor
        EXTRN   MOOV            ; multibyte move
        EXTRN   HPINPUT         ; input routine
        EXTRN   HPOUT2          ; output routine, version 2

; CP/M equates

CPM     EQU     0
BDOS    EQU     CPM+5
CONOUT  EQU     02
PRTBUF  EQU     09

CR      EQU     0DH
LF      EQU     0AH

; program starts

        CSEG

LCM:    LXI     D,SIGNON
        MVI     C,PRTBUF
        CALL    BDOS            ; signon message
LCM0:   LXI     D,VAL1
        CALL    HPINPUT         ; get first integer (A)
        CALL    CRLF
        MVI     E,'>'
        MVI     C,CONOUT
        CALL    BDOS
        LXI     D,VAL2
        CALL    HPINPUT         ; second value (B)

; check for both zero to abort program

        LDA     VAL1
        MOV     B,A
        LDA     VAL2
        ORA     B
        JZ      CPM             ; abort to CP/M

; take absolute values

        LDA     VAL1
        ANI     7FH
        STA     VAL1
        LDA     VAL2
        ANI     7FH
        STA     VAL2            ; strip sign bit from
                                ; length indicators

; begin LCM computation
; The GCD module accepts value pointers in the HL and DE registers.
; The result is returned in the buffer at (DE), destroying the original value.
; The contents of (HL) are undisturbed.

        LXI     H,TMP
        LXI     D,VAL1
        CALL    MOOV            ; save A for later
        LXI     D,TMP
        LXI     H,VAL2
        CALL    MULT            ; form A * B in the
                                ; TMP buffer
        LXI     H,VAL2
        LXI     D,VAL1
        CALL    GCD             ; get GCD(A,B)
                                ; in VAL1
        LXI     D,TMP
        LXI     H,VAL1
        CALL    DIV             ; VAL1 buffer = LCM(A,B)

; TMP now contains the LCM, so print it.

        LXI     D,LCMMSG
        MVI     C,PRTBUF
        CALL    BDOS
        LXI     D,TMP
        CALL    HPOUT2
        LXI     D,ASK
        MVI     C,PRTBUF
        CALL    BDOS
        JMP     LCM0

; utility routine; send CR,LF to console

CRLF:   MVI     E,CR
        MVI     C,CONOUT
        CALL    BDOS
        MVI     E,LF
        MVI     C,CONOUT
        JMP     BDOS

; data areas

        DSEG

SIGNON: DB      'Least Common Multiple Test Program',CR,LF
        DB      'Version 1.0 -- September 22, 1982',CR,LF,LF
ASK:    DB      'Enter two integers, terminated by an equals sign
                ; ( = )'
        DB      CR,LF,'>$'
LCMMSG: DB      'LCM is $'

; storage

VAL1:   DS      128
VAL2:   DS      128
TMP:    DS      128

        END
```

Listing 3

GREATEST COMMON DIVISOR

```
; This program accepts two multibyte integers and outputs the
; Greatest Common Divisor (GCD) using the following algorithm:
;
; 1. Let A,B be the input integers
; 2. Let R = A - B * INT(A / B)
; 3. If R = 0 then GCD = B, terminate
; 4. Let A <-- B
; 5. Let B <-- R
; 6. Goto step 2.
;
; External modules from the HP math library:

        EXTRN   SB1             ; general subract
        EXTRN   MULT            ; multiplication
        EXTRN   DIV             ; truncated division (INT)
        EXTRN   MOOV            ; multibyte move
        EXTRN   HPINPUT         ; input routine
        EXTRN   HPOUT2          ; output, version 2

; CP/M equates

CPM     EQU     0
BDOS    EQU     CPM+5
CONOUT  EQU     02
PRTBUF  EQU     9

; character equates
```

(continued on next page)

```
CR          EQU     0DH
LF          EQU     0AH
BELL        EQU     07H

; program begins

            CSEG

GCD:        LXI     D,SIGNON
            MVI     C,PRTBUF
            CALL    BDOS            ; announce our presence
GCD0:       LXI     D,VAL1
            CALL    HPINPUT         ; get the first input value
            CALL    CRLF
            MVI     E,'>'
            MVI     C,CONOUT
            CALL    BDOS
            LXI     D,VAL2
            CALL    HPINPUT         ; second value

; now have both input values.
; If both are zero, abort program

            LDA     VAL1
            MOV     B,A
            LDA     VAL2
            ORA     B
            JZ      CPM             ; both inputs zero,
                                    ; finished with program

; take the absolute value of both inputs
; by stripping sign bit from length indicator

            LDA     VAL1
            ANI     7FH
            STA     VAL1
            LDA     VAL2
            ANI     7FH
            STA     VAL2

; compute R

GCD1:       LXI     H,R
            LXI     D,VAL1
            CALL    MOOV            ; R <-- A
            LXI     D,VAL1
            LXI     H,VAL2
            CALL    DIV             ; A = INT(A / B)
            LXI     D,VAL1
            LXI     H,VAL2
            CALL    MULT            ; A = B * INT(A / B)
            LXI     D,R
            LXI     H,VAL1
            CALL    SB1             ; R = A - B * INT(A / B)
            LDA     R
            ORA     A               ; is R = 0?
            JZ      DONE            ; yes, print result

; R not zero, move things around

            LXI     D,VAL2
            LXI     H,VAL1
            CALL    MOOV            ; A <-- B
            LXI     D,R
            LXI     H,VAL2
            CALL    MOOV            ; B <-- R
            JMP     GCD1            ; do computation again

; get here when R = 0

DONE:       LXI     D,GCDMSG
            MVI     C,PRTBUF
            CALL    BDOS
            LXI     D,VAL2          ; B = GCD(a,b)
            CALL    HPOUT2
            LXI     D,ASK
            MVI     C,PRTBUF
            CALL    BDOS
            JMP     GCD0            ; do it again

; utility subroutine, send CR,LF to console

CRLF:       MVI     E,CR
            MVI     C,CONOUT
            CALL    BDOS
            MVI     E,LF
            MVI     C,CONOUT
            JMP     BDOS

; data

            DSEG

SIGNON:     DB      'Greatest Common Divisor Program',CR,LF
            DB      'Version 1.0 -- September 19, 1982',CR,LF,LF
ASK:        DB      'Enter two integer values for GCD
                    ; computations:',CR,LF
            DB      '>$'
GCDMSG:     DB      CR,LF,'GCD is $'

; data storage

VAL1:       DS      128
VAL2:       DS      128
R:          DS      128

            END
```

Listing 4

PERMUTATIONS

```
; This program generates the number of permutations possible for
; "n" items taken "m" at a time, using the following formula:
;
;                 n!
; P)n,m) = ------------
;              (n - m)!
;
; The program uses the following EXTERNALs from the HP math library:

            EXTRN   NFACT           ; factorial
            EXTRN   DIV             ; division
            EXTRN   SB1             ; subract routine
            EXTRN   HPINPUT         ; input routine
            EXTRN   HPOUT2          ; output routine, version 2
            EXTRN   MOOV            ; multibyte move

; CP/M equates

CPM         EQU     0
BDOS        EQU     CPM + 5
PRTBUF      EQU     09
CONOUT      EQU     02

CR          EQU     0DH
LF          EQU     0AH

; program begins

            CSEG

PERMUTE:
            LXI     D,SIGNON
            MVI     C,PRTBUF
            CALL    BDOS
PER0:       MVI     E,'>'
            MVI     C,CONOUT
            CALL    BDOS            ; prompt for first number
            LXI     D,VAL1
            CALL    HPINPUT         ; and accept it
            CALL    CRLF
            MVI     E,'>'
            MVI     C,CONOUT
            CALL    BDOS            ; now the second value
            LXI     D,VAL2
            CALL    HPINPUT         ; and get it.
```

```
; check for both inputs equal zero
          LDA       VAL1
          MOV       B,A
          LDA       VAL2
          ORA       B
          JZ        CPM              ; if both = 0 then abort
; set up area TMP
          LXI       H,TMP
          LXI       D,VAL1
          CALL      MOOV             ; put n --> TMP
          LXI       D,TMP
          LXI       H,VAL2           ; form TMP = (n-m)
          CALL      SB1
          LXI       D,TMP
          CALL      NFACT            ; form (n-m)!, stored in
                                     TMP
          LXI       D,VAL1
          CALL      NFACT            ; form n!
          LXI       D,VAL1
          LXI       H,TMP
          CALL      DIV              ; now form n! / (n-m)!
          LXI       D,VAL1
          CALL      HPOUT2           ; print the answer
; finished with the computations, get some more
          JMP       PER0
;send CR,LF to console
CRLF:     MVI       E,CR
          MVI       C,CONOUT
          CALL      BDOS
          MVI       E,LF
          MVI       C,CONOUT
          JMP       BDOS
SIGNON:   DB        'Permutation Test Program -- Version 1.0',
                    CR,LF,LF
          DB        'Enter two integers, terminated by an equals
                    sign.',CR,LF,'$'
; data areas
VAL1:     DS        128
VAL2:     DS        128
TMP:      DS        128
          END
```

Listing 5

COMBINATIONS

```
; This program computes the value of "n" objects combined "m"at a time.
; The formula used is:
;
;                      n!
; C(n,m) =        ------------
;                  m!(n – m)!
;
; Externals used:
          EXTRN     SB1              ; subtraction
          EXTRN     NFACT            ; factorial
          EXTRN     MULT             ; multiplication
          EXTRN     DIV              ; division
          EXTRN     MOOV             ; multibyte move
          EXTRN     HPINPUT          ; input routine
          EXTRN     HPOUT2           ; output version 2
; CP/M equates
CPM       EQU       0
BDOS      EQU       CPM+5
CONOUT    EQU       02
PRTBUF    EQU       9
CR        EQU       0DH
LF        EQU       0AH
; program begins
          CSEG
COMB:     LXI       D,SIGNON
          MVI       C,PRTBUF
          CALL      BDOS
COMB0:    MVI       E,'>'
          MVI       C,CONOUT
          CALL      BDOS             ; get first value
          LXI       D,VAL1
          CALL      HPINPUT
          CALL      CRLF
          MVI       E,'>'
          MVI       C,CONOUT
          CALL      BDOS             ; and second
          LXI       D,VAL2
          CALL      HPINPUT
; check for both values = 0
          LDA       VAL1
          MOV       B,A
          LDA       VAL2
          ORA       B
          JZ   '    CPM              ; if both = 0 then abort
                                     ; start computations
          LXI       H,TMP
          LXI       D,VAL1
          CALL      MOOV             ; put n --> TMP
          LXI       D,TMP
          LXI       H,VAL2
          CALL      SB1              ; form (n-m) in TMP
          LXI       D,TMP
          CALL      NFACT            ; form (n-m)! in TMP
          LXI       D,VAL1
          CALL      NFACT            ; form n!
          LXI       D,VAL2
          CALL      NFACT            ; form m!
          LXI       D,TMP
          LXI       H,VAL2
          CALL      MULT             ; form m! * (n-m)! in TMP
          LXI       D,VAL1
          LXI       H,TMP
          CALL      DIV              ; now form n! / m!(n-m)!
                                     ; in VAL1
; VAL1 now contains answer, print it
          LXI       D,VAL1
          CALL      HPOUT2
; and get some more numbers
          JMP       COMB0
; send CR,LF to console
CRLF:     MVI       E,CR
          MVI       C,CONOUT
          CALL      BDOS
          MVI       E,LF
          MVI       C,CONOUT
          JMP       BDOS
SIGNON:   DB        'Combination Test Program -- Version 1.0',
                    CR,LF,LF
          DB        'Enter two integers, terminated by equal signs.',
                    CR,LF,'$'
; data areas
          DSEG
VAL1:     DS        128
VAL2:     DS        128
TMP:      DS        128
          END
```

(continued on next page)

Listing 6

RANDOM NUMBER GENERATOR

```
; This program uses the following linear congruential generator to
; produce random numbers for use with the HP math library:
;
; X(n+1) = (a * X(n) + c) MOD m, (n>= 0)
;
; Where
; m = 2↑127 - 1
; a = c = 2↑16 + 1
;
; Externals
;
            EXTRN     DIVM          ; modulus
            EXTRN     AD1           ; addition
            EXTRN     MOOV          ; multibyte move
            EXTRN     MULT          ; mutliplication

; declare entry points for the library
            PUBLIC    HPRAND        ; random number
                                    ; generator entry
            PUBLIC    RNUM          ; random number
                                    ; integer space
; The storage space for the random number is declared to be PUBLIC
; to facilitate program access to it. In this way the calling program
; declares RNUM to be EXTRN and then references it just like any other
; label. The linker will resolve address values during program and
; library linking.
            CSEG
HPRAND:     MOV       A,D
            ORA       E             ; if DE = 0 then use last
                                    ; seed value
            JZ        HPRND1
            LXI       H,XN          ; else use (DE) as
                                    ; new seed
            CALL      MOOV
HPRND1:     LXI       D,XN          ; generate new
                                    ; random number
            LXI       H,RMULT
            CALL      MULT          ; form (a * X(n))
            LXI       D,XN
            LXI       H,CONST
            CALL      AD1           ; form (a * X(n) + a)
            LXI       D,XN
            LXI       H,MODULUS
            CALL      DIVM          ; take the modulus
            LXI       H,RNUM
            LXI       D,XN
            CALL      MOOV          ; put new random
                                    ; number in public view
            RET
; data areas
            DSEG
RMULT:
CONST:      DB        3,01,00,01    ; a = c = 2↑16 + 1
MODULUS:
            DB        10H,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,
                      0FFH,0FFH,0FFH,0FFH,0FFH
            DB        0FFH,0FFH,0FFH,0FFH,7FH
XN:         DS        128
RNUM:       DS        128
            END
```

Listing 7

TEST RANDOM NUMBER GENERATOR

```
; This program will test the random number generator implemented
; for the HP math library. After seeding the generator with the "value"
; which results by using the starting label as the representation of a
; multibyte integer, the program will enter an infinite loop, generating and
; printing random numbers until aborted by a keyboard input.
            EXTRN     HPRAND        ; the random number
                                    ; generator
            EXTRN     HPOUT2        ; output routine, version 2
            EXTRN     RNUM          ; random number buffer
; CP/M equates
CPM         EQU       0
BDOS        EQU       CPM+5
CONST       EQU       11
PRTBUF      EQU       09
CR          EQU       0DH
LF          EQU       0AH
; program begins
            CSEG
TSTRND:     LXI       D,SIGNON
            MVI       C,PRTBUF
            CALL      BDOS
            LXI       D,HPOUT2      ; use this address (label)
                                    ; for seed
            CALL      HPRAND
; generator is now seeded, start printing values
LOOP:       LXI       D,0
            CALL      HPRAND
            LXI       D,RNUM
            CALL      HPOUT2
            MVI       C,CONST
            CALL      BDOS
            ORA       A
            JZ        LOOP          ; look for abort
                                    ; from keyboard
            JMP       CPM
; messages
SIGNON:     DB        'Random Number generator test program',CR,LF
            DB        'Version 1.0 -- October 23, 1982',CR,LF,LF,'$'
; no data areas are needed
            END
```

[Ed. Note: The author concedes that the Prime Number algorithm in Listing 1 is something of a "Brute Force" approach. One significant improvement would be to define a fourth 128-byte buffer just above END, store into it the square root of each new T value (perhaps somewhere near LOOP1), and compare against that value instead of the whole T, above NEXT1.]

OOPS:

In our May 1983 issue under *Software Notes*, p.35, our author's name Thomas L. Robb was inadvertently left off.

Software Notes

Date Your Disks

by Andrew Hughes

It is often useful to know the date on which a disc was last used or changed: listing 1 shows a short program which stores a date in the Directory of the discs you are using. The program, which with great originality I have called DATE.COM, should be first run by the auto-start feature of CP/M, and "DATE" should be patched into the system.

When run or called in this way, without parameters, DATE searches the Directory of drive A for a filename beginning with two slashes, and a filetype beginning with an apostrophe, thus: //??????. '??. If no such "name" is found, you will be asked to enter two characters for the date of the day (or any other code you might prefer) and three for the month (no CRs are needed, and ANY control character or space will cause an exit to CP/M). Then the program continues as if it had found the "name" in the first place: it shows you what it found or what you entered, as /-??????. '??. A CR confirms the entry and stores it to the disc in drive A:; any other character will allow you to change the year (for which a initial default of 82 is provided by the program). You will then again be allowed to confirm and save the date with a CR, or redo the whole entry with any other character.

What you have stored in the Directory of the disc in drive A is the current date, which will remain available throughout the session unless you change discs in drive A. This current date you can store on any disc, in any drive, by calling DATE with the drive-letters as parameters, thus: DATE A, or DATE CAB. The date is shown to you for confirmation. It will remove any previous such date, and is stored in the form /-15-Nov. '82 (lower case letters will appear as such in the Directory, and the "name" can be deleted only be ERA /*.* because CP/M accepts only upper case for its entries). On the disc in drive A, then, you may have both the // and the /- forms: the former is the current date (which will of course be out of date on the following day, so if you change discs in drive A, you must be careful to check it); the latter is the date the disc was last used.

The slash and hyphen were chosen because they will appear first in alphabetized listings of the Directory, or immediately after the disc name, if you use a cataloguing system which employs the hyphen for that purpose. When I begin to use a new disc, I try to remember to put a name on the disc first, then a dummy date of last use (this must be in the form /-??????./??), so that they show first even in listings which are not alphabetized. These, and subsequent dates saved in the Directory, are in the form of "names" referring to dummy files of zero length: no space is taken up on the data area of the disc. They can be entered directly onto the disc with SAVE 0 NAME.

The program is documented, and where there are no comments the symbolic names explain what is happening. It was assembled with the PASM Z80 system, and uses a few Z80 codes: these should be easy to replace for 8080 operation. The PASM loader (PLINK) automatically assigns a local stack area, and these statements do not appear in the program: as only about a dozen bytes of stack are needed, I doubt whether a local stack is necessary. In my original, the error exits jump to a breakpoint routine which announced an address from which I can, if necessary, tell which error occurred. I've let them all fall through to a warm boot: explicit messages could easily be added.

PSA Macro Assembler [C12011-0102]

.MAIN. -

```
                    .radix 16
0100                .loc 100
                    ; CP/M addresses, in hexadecimal
0005                bdos = 5
005C                fcb = 5c
005D                param.loc  = fcb + 1
0080                buff = 80
0000                warm.boot = 0
                    ; CP/M functions, in hexadecimal
000E                log = 0e
0011                find = 11
0013                delete = 13
0016                create = 16
0010                close = 10
0001                conin = 1
0201                conin2 = 0201
0301                conin3 = 0301
0009                conpr = 9
                    ; miscellaneous equates
00FF                fail = off
                    ; insert a suitable error routine
0000                err = warm.boot
0000                drive.a = 0
0020                end. param = 20
0020                no.param = end. param
000C                length = end.todays.date-todays.date
000F                make.bin = 0f
000D                cr = 0d
                    ;
00100               start:
0100    0E0E        mvi   c,log
0102    1E00        mvi   e,drive.a
0104    CD 0005     call  bdos
                    ;
0107    0E11        mvi   c,find
0109    11027C      lxi   d,prev.date
010C    CD 0005     call  bdos
010F    FEFF        cpi   fail
0111    200E        jrnz  ..1
                    ;
0113    CD 0177     call  input         ; if no prev. date, enter it
0116    21 005D     lxi   h,param.loc   ; and exit if no
0119    7E          mov   a,m           ; other drives are specified
011A    FE20        cpi   no.param
011C    20E2        jrna  start
011E    C3 0000     jmp   warm.boot
0121                ..1:
```

(continued on next page)

```
0121    87          add  a            ; reg. a has the offset, in
0122    87          add  a            ; the default buffer, of the
0123    87          add  a            ; FDB which has the pre-
0124    87          add  a            ; vious date. Calculate off-
0125    87          add  a            ; set in bytes and add it to
0126    C680        adi  buff         ; the buffer address.
0128    6F          mov  1,a
0129    2600        mvi  h,0
012B    11 02C3     lxi  d,todays.date
012E    D5          push d            ; move prev. date to buffer
012F    01 000C     lxi  b,length     ; for today's date.
0132    EDB0        ldir
0134    D1          pop  d            ; point to position for the
0135    13          inx  d            ; hyphen or slash.
0136    13          inx  d
0137    3E2D        mvi  a, ' — '     ; make today's date into
0139    12          stax d            ; the disc date.
013A    CD 01C6     call is.it.ok
013D    2815        jrz  ..2
013F    3E2F        mvi  a,'/'        ; restore prev. date if not
                                      ; O.K. and delete it
0141    12          stax d            ; not OK, and delete it.
                    ;
0142    0E0E        mvi  c,log
0144    1E00        mvi  e,drive.a
0146    CD 0005     call bdos
0149    0E13        mvi  c,delete
014B    11 027C     lxi  d,prev.date
014E    CD 0005     call bdos
0151    C3 0100     imp  start
0154                ..2:
0154    21 005D     lxi  h,param.loc  ; point to drives named.
0157    E5          push h
0158                log.and.save:
0158    E1          pop  h
0159    7E          mov  a,m          ; do until no more drives.
015A    FE20        cpi  end.param
015C    CA 0000     jz   warm.boot
015F    23          inx  h
0160    E5          push h
0161    3d          dcr  a            ; make drive letter, ABC or
0162    E60F        ani  make.bin     ; abc, into 012 binary.
0164    5F          mov  e,a
0165    0E0E        mvi  c,log
0167    CD 0005     call bdos         ; log specified drive.
                    ;
016A    0E13        mvi  c,delete
016C    11 0292     lxi  d,date.last.used
016F    CD 01AC     call save         ; new date of use for this
0175    18E1        jmpr log.and.save ; disc.
                    ;
02C6                day.loc = todays.date + 3 ; this if the place in
                                      ; the "Filename" where
                                      ; the actual date begins.
                    ;
0177                input:
0177    11 01F9     lxi  d,day
017A    CD01F3      call mess         ; prompt for . . .
017D    21 02C6     lxi  h,day.loc
0180    CD 01DF     call enter2       ; today's date (2 chars.)
0183    362D        mvi  m, ' — '     ; separator
0185    23          inx  h
0186    E5          push h
0187    11 022C     lxi  d,month
018A    CD 01F3     call mess         ; prompt for . . .
018D    E1          pop  h
018E    01 0301     lxi  b,conin3
0191    CD 01E2     call enter        ; month (3 chars.)
0194    E5          push h
0195    CD 01C6     call is.it.ok
0198    E1          pop  h
0199    2811        jrz  save
019B    23          inx  h            ; not OK: Look at the year.
019C    E5          push h
019D    11 0259     lxi  d,year
01A0    CD 01F3     call mess         ; prompt for. . .
01A3    E1          pop  h
01A4    CD 01DF     call enter2       ; year (2 chars.)
01A7    CD 01C6     call is.it.ok
01AA    20CB        jrnz input        ; not OK: do it all again.
                    ; This creates the date as a "Filename", and
                    ; saves an empty file, using no more space
                    ; on the disc.
01AC                save:
01AC    0E16        mvi  c,create
01AE    11 02C3     lxi  d,todays.date
01B1    D5          push d
01B2    CD 0005     call bdos
01B5    FEFF        cpi  fail
01B7    CA 0000     jz   err
01BA    DI          pop  d
                    ;
01BB    0E10        mvi  c,close
01BD    CD 0005     call bdos
01C0    FEFF        cpi  fail
01C0    CA 0000     jz   err
015C    C9          ret
                    ;
01C6                is.it.ok:
01C6    21 02CF     lxi  h,end.todays.date
01C9    E5          push h            ; put a $ for CP/M's
01CA    D5          push d            ; print-string function.
01CB    3624        mvi  m,'$'
01CD    11 02AB     lxi  d,showdate
01D0    CD 01F3     call mess
01D3    0E01        mvi  c,conin      ; enter CR if OK
01D5    CD 0005     call bdos
01D8    D1          pop  d
01D9    E1          pop  h
01DA    3600        mvi  m,0          ; remove $ and restore
                                      ; null.
01DC    FEOD        cpi  cr           ; return the zero flag.
01DF                enter2:
01DF    01 0201     lxi  b,conin2
01E2                enter:            ; number of chars. in reg. b.
01E2    C5          push b
01E3    E5          push h
01E4    CD 0005     call bdos
01E7    E1          pop  h
01E8    C1          pop  b
01E9    FE21        cpi  "!"          ; exit to CP/M if space
01EB    DA 0000     jc   warm.boot    ; or control entered.
01EE    77          mov  m,a          ; otherwise store the char.
01EF    23          inx  h
01F0    10F0        djnz enter        ; and repeat
01F2    C9          ret
                    ;
01F3                mess:
01F3    0E09        mvi  c,conpr
01F5    CD 0005     call bdos
01F8    C9          ret
                    ;
                    ; prompt for entry of day
01F9                day:
                    .ascii "
01F9    0D0A43522074     CR to exit, or-
```

(continued on page 34)

VARPTR Cuts Path to CP/M

John S. Coggeshall

(Note: Programs listed herein are copyright © by John S. Coggeshall, who retains the right to distribute them in any form. The reader may put them to private use and incorporate them in programs distributed without source code, but may distribute copies of the listings in any form only with written permission of the author. The reader may, however, use them in articles for this publication if the author is acknowledged.)

It is often necessary to reach CP/M's Basic Disk Operating System (BDOS) facilities from a high-level language like CBASIC or CB-80. This article presents a user-defined function in CBASIC that makes all BDOS services available through a standard mechanism. The VARPTR function provides for efficient execution, and the overhead committed is about the same as for a specialized routine to obtain any one BDOS service.

The arguments to our function—FNBDOS%—will be the BDOS function-code and an integer to be passed in the DE register pair. Obviously, this maps single-byte arguments into register E where they belong. FNBDOS% will return an integer whose low byte is the value returned by BDOS in the A register and whose high byte is register B. Thus access to the BDOS always has the same form, whether data is exchanged or not. The price of this convention, at worst, is negligible code and microseconds of run time.

A reassuring word may be in order at this point; some BDOS functions return (if anything) a single byte in register A, others a word in the HL pair. Now the CP/M 2.0 Interface Guide states on p.3, "For reasons of compatibility, register A=L and register B=H upon return in all cases." So far, so good. But can we be sure that register B contains zero instead of garbage on return from routines meant to deliver a single byte? The answer, it turns out, is yes; and the two-byte integer constructed as above will always have the correct value. Not satisfied with my own results indicating this, I phoned Digital Research, Inc. and spoke with Linda Haigh, who confirmed that the returned value is defaulted to zero during the setup common to all BDOS routines. All routines exit through a sequence that loads the (possibly modified) value back into HL and copies it into B and A. Thus there is no need to distinguish between two-byte and one-byte routines. No end of suffering and inelegance could doubtless have been avoided had DRI mentioned this simple policy in the Interface Guide.

The usage, then will be of the form:

```
BDOS.Return.Code% =FNBDOS%( Function.Code%, DE.Arg% )
```

CBASIC's CALL statement provides transfer of control, but by itself gives no way to pass arguments or return results. We must therefore construct a means of access to CPU registers, for which there are no built-in statements or functions; this requires an assembly-level program. Such a program could be assembled and then brought into memory with the powerful SAVEMEM statement; but the program needed here is so short that it is simpler to encode it as a string variable, obtaining its entry point through the SADD function.

Naturally, we would like the code for FNBDOS% to be both compact and brisk. One easy to achieve both compactness and speed is to reduce the number of CBASIC statements executed during an invocation of FNBDOS%. The approach here is to use indirect register-load instructions to move the argument into DE and to return the result, operating upon an integer variable in place. The pointer (VARPTR) to the variable is patched into the assembly program for this purpose during initialization. Thus only integer assignment statements are required before and after each CALL in order to communicate those values. The function-code is passed each time by patching the program with a POKE statement.

Support for FNDBOS% consists of two "reserved" global variables, which must not be reassigned: BDOS$, the assembly program, and BDOSE%, its entry point. (BDOSE% could be eliminated by calculating SADD (BDOS$) twice on each call, at some penalty in speed.) In addition, the global BDARG% needs no protection but must exist and is modified by each call; we need it only for its permanent address, and it could just as well be any "junk" integer. It would be nice if we could use the dummy parameter itself, but its pointer is not constant because it is local to FNBDOS%.

The only remaining unexplained feature used to initialize for FNBDOS% is FNPOKE2%. It is so valuable for communicating with assembly routines and for other purposes that it has joined an INCLUDE file named @EVRY.BAS. (Just about "evry" program needs this module.):

```
DEF FNPOKE%( ADDR%, WRD% )
    \      Poke 2-Byte Word to ADDR% (LoHi)
    POKE ADDR%, WRD%                         REM Low byte
    POKE ADDR% + 1, Peek( VARPTR(WRD%) + 1 ) REM High byte
    RETURN
FEND
```

(CBASIC's and CB-80's implementation of a continuation character is one of the beauties of the languages: unless otherwise indicated, statements are terminated by linefeeds. Treating the continuation character as a remark is a further stroke of genius, providing an alternative to the somewhat clumsy 'REM.' Has anyone estimated the number of semicolons required by a useful program in PL1, Pascal, or C? Not to mention the number of symmetrically paired special sequences they demand for comments, such as '/*,' which either discourage comments or make them an obsession. The purpose of a high-level language is to do work for programmers, not to discipline them.)

If the foregoing tirade can be forgiven, we are now equipped for FNBDOS% and its initialization:

FNBDOS% now provides, in a single function, direct access to all of the BDOS services. This simplifies the performance of a wide range of operations not implemented by CBASIC, such as:

```
-- Customized, tightly controlled console input;
     BDOS Function 6
-- Directory operations; Functions 17 and 18:
        [ Define DMA$, a 128-byte buffer ]
     Q% = FNBDOS%( 26, SADD(DMA$) + 1 )
                                        REM  Set DMA Address
        [ Format file name into FCB$, a 36-byte string ]
     DIRCODE% = FNBDOS%( 17, SADD(FCB$) + 1)
                                        REM  Search for First
     WHILE DIRCODE% <>OFFH
        [ Get matching file name out of DMA$ and put as desired ]
        DIRCODE% = FNBDOS%( 18,0)     REM  Search for Next
        WEND
-- Control of file attributes: "Read-Only" and "System" status; Function 30
-- Access to files in any User Area; Function 32:
        USER% = FNBDOS%( 32,OFFH )
REM  Get current User Code
        Q% = FNBDOS%( 32,U1% )     REM  Set User Code to #U1%
        [ Work with files or directory in User Area U1% ]
        q% = FNBDOS%( 32, USER% )   REM  Back to original User
```

Typically, the programmer would implement the more complex system calls as user-defined functions which in turn invoke FNBDOS%. This approach can reduce the time spent looking up function usage but, far more importantly, guarantee the proper support for each BDOS service request.

[Listing 1]

```
\  @BDOS.BAS

\  FNBDOS%, for generalized use of CP/M's BDOS services
\  Requires @EVRY.

\  Globals, used by FNBDOS%:
\  BDOS$ and BDOSE% must not be reassigned
\  BDARG% for passing integers; needs no protection

\  Initialize 'Assembly program' to be CALLed by FNBDOS%:
\  -- All it really does is communicate between
\  -- CPU registers and CBASIC variables:
BDOS$ = CHR$(0EH)  + " - "            \    LD C, FnCode
                  \   (to be POKEd before each CALL)
     + CHR$(21H)  + " -- "            \    LD HL, .BDARG%           *
     + CHR$(5EH)                      \    LD E,(HL)                Lo Byte
     + CHR$(23H)  + CHR$(56H)         \    INC HL  ! LD D,(HL)      Hi Byte
     + CHR$(0E5H)                     \    PUSH HL                  Save addr
     + CHR$(0CDH) + CHR$(5) + CHR$(0) \    CALL BDOS
     + CHR$(0E1H)                     \    POP HL
     + CHR$(70H)                      \    LD (HL),B                Hi byte
     + CHR$(2BH)  + CHR$(77H)         \    DEC HL  ! LD (HL),A      Lo byte
     + CHR$(0C9H)                     \    RET
*Note: Double hyphen gets permanently patched with address

BDOS% = SADD (BDOS$)  + 1             REM  Entry point
BDARG% = 0                            REM  For BDOS DE-arg and rtn code
\  Patch the code with its address:
Q% = FNPOKE%(BDOSE% + 3, VARPTR(BDARG%))

DEF    FNBDOS%(FCODE%,DEARG%)
\  For general calls to BDOS
\  Rtns:  BDOS return-code; high byte from B register
           ( Registers A = L and B = H in all cases )
\  Usage:  Rtn.code% = FNBDOS%( code%, arg% )
POKE BDOSE% + 1, FCODE%               REM  BDOS Fn Code
BDARG% = DEARG%                       REM  Arg; lo byte --) E
CALL BDOSE%
\  Returned value to BDARG% by indirect load
FNBDOS% + BDARG%
RETURN
FEND ■
```

Software Notes

New Products

GraphPlan

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

This business package offers a spreadsheet, built-in statistical commands, presentation quality graphics and sorting and ranking capabilities. It has built-in formulas, automatic generation of legends, numerical date, time and logarithmic X and Y axis labels and tic marks. Presentation quality graphics such as explodable pie charts, horizontal or vertical line and bar graphs with stacking capability and scattergrams can be created individually, or can be combined. GraphPlan's spreadsheet changes are automatically recorded in the graphics, and the user can switch between the spreadsheet and the graphics with the push of a key without exiting the program. It can be used with MicroPlan.

Requirements: CP/M-80 or MS-DOS, min. 64K-128K, one double sided disk with 330K bytes of storage.

Price: $395

EXPENSE TRAC

OUTPUT Inc.
2401 E. Washington St.
Bloomington, IL 61701

This program automates fund accounting procedures of school administration, small profit and non-profit organizations, and departmentalized budgeting for divisions of larger companies. It is written in RM/COBOL. EXPENSE TRAC allows users to define values for accounting structures such as funds, cost centers, and account numbers. It maintains a master file of current balances for budgeted, expended, and encumbered funds. It provides a detailed audit trail printout summarizing all transactions entered into the system. It allows the user to see on-screen displays of account balances, account details, requisition details, and vendor code details. It provides up to 15 summary and detail reports in a variety of sequences and totaling schemes. It provides increased file space through a data compaction process.

Requirements: CP/M-80, Printed reports require an 80-column dot-matrix or letter quality printer.
Price: N/A

DECISION ANALYST

EXECUTIVE SOFTWARE INC.
Two N. State Street
Dover, DE 19901

This new program assists professionals in analyzing complex business problems where there are many alternatives and criteria. It structures the decision making process into logical and easy to follow steps. The program is designed for ease of use with menu screens. It contains eight menu selected sections including problem definition, statement of decision purpose, establishing and valuing 'must' and 'want' criteria, calculation of criteria values, defining alternatives, weighing and scoring alternatives against criteria, and final conclusions and choice. The final reports are printed in polished format. DECISION ANALYST is written in CB80 with over 100,000 bytes of compiled code and a 40,000-character help file.

Requirements: CP/M-80, CP/M-86, or MS-DOS, 52K (96K with CP/M-86 and MS-DOS), a 24 X 80 column screen and an 80 column printer.

Price: $139

TECHTYPE

Green Mountain Radio Research Co.
240 Staniford Rd.
Burlington, VT 05401

This multifont, text-formatting system is designed especially for scientific, engineering, mathematical, and multi-lingual document production. It allows unlimited sub- and super-scripting and has the ability to mix up to ten fonts of the user's choice. It also provides control of format, pitch, and emphasis and can even address envelopes and mark classified materials. The three principal programs that make up TECHTYPE are DISPLAY, DRAFT, and DRAFT is used with a multifont dot-matrix printer to produce high-speed drafts and working papers. PRINT is used with a daisywheel printer to PRINT. DISPLAY allows the user to preview the document on the screen with emphasis features displayed. produce camera-ready copy and final reports. Multipass printing allows the printwheel to be changed only ONCE per page per font.

Requirements: CP/M-80, 48K

Price: $300

VAAS

Vertec
PO Box 1116
8079 N. Lake Blvd.
Kings Beach, CA 95719

This integrated accounting package is designed for insurance agencies. It tracks sales and production volume, performs accounting functions and client profiling on a personal computer. It provides historical data and management reports for planning and analysis of agency production. VAAS is completely menu driven and user friendly. It provides a full complement of management reports. The manual and software have been integrated to provide an easy-to-understand method of automating an agency.

Requirements: CP/M-80

Price: N/A

FileDriver

DUNBAR-RIDGE CORP.
102 Sterling Ct.
Syosset, NY 11791

This integrated set of comprehensive file handling utilities for CP/M 2.2, TurboDOS 1.2 and MP/M II operating systems can be accessed though a menu-driven interface as well as from the CP/M command line. File Driver is disk format independent, does not require any BIOS changes, and does not interfere with other programs the user may wish to run. Its consistent syntax and facility for specifying complex operations as single-word commands makes the package easy to learn and use. FileDriver has the ability to access 31 user areas, automate complex user-designed operations, move files between user areas without copying, enter multiple commands on one line, archive disk files, and keep a disk file log of its operations. It also allows access to multiple commands without reading separate COM or overlay disk files and allows for groups of commands to be created and executed in batch by accepting input from a file created by a text editor. FileDriver has disk maintenance features to find and mark bad sector areas and batch process utility commands for creation of archival and other file management systems. FileDriver's utility commands include both new commands and very enhanced functions of CP/M's 2.2 utilities.

Requirements: CP/M-80 2.2, TurboDOS 1.2, or MP/M II.

Price: $75

AudIt

E.F. Haskell & Associates
1528 E. Missouri Ave., A131
Phoenix, AZ 85014

This menu driven set of micro computer tools is designed specifically for independent and internal auditors. It has a run-time interpreter, so no other languages are needed. AudIt enables the auditor to instantly determine lease types using a flowchart analysis of Financial Accounting Standards Board No. 13. It maintains long-term depth scheduling and a complete set of user designed audit working paper forms up to 132 columns wide. It handles financial depreciation analysis and loan amortization schedules. AudIt allows inventory volume analysis computations and sorted random number generation. In addition, it allows the auditor to use the computer as a calculator while using the AudIt system. AudIt includes handy routines to instantly convert from one unit of measure to another.

Requirements: CP/M-80, MP/M or TurboDOS, Z80 or 8080

Price: N/A

BUYSEL

Single Source Solution
2637 Pleasant Hill Road
Pleasant Hill, CA 94523

This menu-driven package is mathematical and statistical routines for making specific buy and sell decisions in the stock, commodities and options markets. BUYSEL is intended for optimization or "tuning" of these methods, indications of appropriate buy and sell signals on a daily basis, and creation/validation of relevant price history files. It includes the moving average method, the Max/min methods the average down/sell up method and the correlation method. BUYSEL computes commission for commodities on a flat rate basis and for stocks and options a a rough percentage basis. The menu structure lends itself readily to experimenting on a single piece file with different techniques.

Requirements: CP/M or CDOS, 64K

Price: $149.95

Z-80 Assembler

King Software
PO Box 208
Red Bank, NJ 07701

This CP/M compatible Z-80 assembler, plus a top-down tutorial on the theory of assemblers features: 1) standard Zilog mnemonics 2) 19 pseudo-op's, including TITLE, XLIST, and nested conditionals with ELSE 3) Ability to accept a source program split up into multiple input files 4) Object file in standard Intel Hex format 5) Listing of sorted symbol table 6) Modular structure, allowing easy revision as a cross-assembler 7) Symbolic definition of all important parameters (for example, the number of characters in a symbol), making it simple to adapt details of language or syntax to individual preference.
Advanced Techniques explained in the tutorial (with many illustrations in pseudo-code) include: Radix 40, expression processing by recursive descent, Op-code analysis, binary search of symbol table, character table look-up, recursive processing of nested conditionals.
The source listing for the assembler is given in Z-80 assembly language, fully commented. A direct translation into 8080 assembly language, suitable for assembly by CP/M's ASM, is also included. The complete source code is also available on a standard CP/M soft-sectored, single-density 8-inch diskette.

Requirements: Z-80 CPU
Price: $37

New Books

Microcomputers Can Be Kid Stuff

Hayden Book Company, Inc.
50 Essex Street
Rochelle Park, N.J. 07662

Microcomputers Can Be Kid Stuff enables young people to learn about microcomputers and about how to use them productively. Written by Anna Mae Walsh Burke, the book prepares youngsters to begin "speaking" BASIC and Pilot with clear descriptions and explanations of microcomputer hardware and software. Information on writing programs, saving programs on diskettes or cassettes, and using commercial software is also provided.

Price: $8.95

Bugs

The IBM-PC has a slight bug in its ROM-BIOS; it is not possible to convince the video BIOS functions that you have both a monochrome and a color monitor. This update changes PCVSUM to "fake out" the BIOS so that you can switch between the two monitors.

1) Missing ENDS Statement
The four statements of the assembly language examples (pg 1-33 of the manual) should read as follows:

```
XCFIND ENDP
XCMAKE ENDP
PROG ENDS
END
```

2)Use Call "_exit," not "exit." A short version of "_main" is presented (on p 1-38 of the manual); however, the final statement before the closing brace should read:

```
_exit(0)
```

If "exit" is called, the level 2 I/O functions are included in the program. Note that the correct version of this function has now been supplied as TINYMAIN.C.

(continued from page 2)

However they are hard at work and are clearly a force to be reckoned with in the months and years ahead.

The advent of eight bit softcards for sixteen bit machines and the amount of eight bit software which has been ported to sixteen bit machines explodes the myth that software technology is keeping pace with hardware technology.

Application programs continue to proliferate with no indication that they will ever be supplanted by anything other than more application programs.

Good databases for micros simply don't exist at this point, although many are hard at work to provide them to what is believed to be a phenomenally large market.

There is little evidence to support the belief that balancing one's checkbook is a good or even interesting use for microcomputers.

While matrix printers have continued to improve in print quality they show no signs whatsoever of replacing formed-character printers.

S100 machines continue to wane and while still powerful machines have at best a limited lifetime. The additional manufacturing costs, the larger physical size, the diminishing techincal user base, etc., all reduce the market demand for this once dominant form of hardware.

As more and more machines offer Lisa-like capability the need for large CRT screens with higher resolution will continue. Five inch portable machines are more likely to result in a large number of people all of whom have poor eyesight and one arm longer than the other. The concept of the portable machine is apparently based upon the assumption that anything moveable with a forklift is portable!

Apple III is struggling but is rumored to be on the verge of extinction, while Apple II clones continue to appear. Lisa, while an innovative concept, seems far too expensive and innovative to have the effect suggested by Apple fans.

As for the myth that IBM has missed the personal computer market . . . need we say anything?

The IBM-PC while an exciting entry in the microcomputer race, has not proven to be a true quantum leap in the hardware technology. However its effects upon the microcomputer world have been incredible and every day new technology arises as a result of IBM's lead.

Softcards continue to enjoy widespread use as an alternative to waiting for sixteen bit versions of eight bit programs to become available. It is paradoxical that one would invest first in a sixteen bit computer, then in an eight bit softcard which relegates the sixteen bit computer to the role of expensive dumb terminal. But absent sixteen bit softcard what other alternative is there ?

Microcomputers are increasing job opportunities enormously and this trend shows every sign of continuing. Microcomputers, as all other forms of computers, need operators, programmers, system analysts, technicians, etc. The mircrocomputer makes it possible to perform many tasks much more readily but also causes most of us to tackle more complex tasks. What self-respecting microcomputer user would ever claim that he spends less time working with a microcomputer? Microcomputers have a tendency to function as infinte time sinks for most of us.

It's sad but true that for all practical purposes documentation is never read. Authors have finally accepted this fact and are now focusing on luring users back to the printed page. Cartoons, detailed illustrations, four color artwork, novel packaging, etc. are all being employed to this end.

As long as the technocrats are with us they will always insist that the "________________" language makes complex programming tasks trivial. Fundamental information theory makes it clear that this is a myth but why bother to explain this to them when we all know better?

Those of you who have had the misfortune to attempt to translate eight bit assembly language problems to sixteen bit environments are well aware that this is not a trivial task and the results are often underwhelming. If all of this seems rather confusing in terms of deciding how you are to interpret the plethora of allegations and prognostications about the microcomputer world don't feel like the Lone Ranger! As long as we are at the mercy of the prophets, gurus, soothsayers, experts, geniuses and visionaries Pogo's observation will prevail, "We have met the enemy and he is us. . . ." Time is the greatest validator and in this industry it doesn't take long. The truth will out, and soon!

(continued from page 28)

```
020B    0D0A456E7465    Enter date of  day (2 chars.): $"
                        ;
                        ; prompt for entry of month
022C                    month:
                        .ascii"
022C    0D0A43522074    CR to exit, or-
023E    0D0A456E7465    Enter month 3 (chars.): $"
                        ;
                        ; prompt for entry of year
0259                    year:
                        .ascii"
0259    0D0A2D2D2D2D    —— Enter year (2 chars.):$"
                        ;
                        ;
027C                    prev.date:
027C    012F2F3F3F3F    .ascii [01] "//??????. '??"
0288    000000000000    .byte [10.]0
                        ;
0292                    date.last.used:
0292    002F2F3F3F3F    .ascii [0] "/-??????. '??"
029E    000000000000    .byte [10.]0
                        ;
                        ; this message
                        ; ends at
                        ; end.todays.date.
02A8                    showdate:
                        .ascii  "
02A8    0D0A
02AA    0D0A4F4B203F    OK? Check years.  CR = Y:   "
02C3                    todays.date:
02C3    002F2F202020    .ascii [0]"//        '82"
02CF                    end.todays.date:
02CF    000000000000    .byte [10.]0
0100                    .end    start

0E0E
```

Take Control

by Steven Fisher

YOU FINISH UNPACKING YOUR NEW GIZMO-6000 printer with its ten-character styles and full graphics capability. The case looks good and the paper goes in easy. You slip the data cable into your trusty computer, turn on the printer, press 'ON LINE' and then begin printing. Great! Now you want to use those fancy features—but how?

Your printer needs to receive special control sequences to engage its extra modes. You cannot modify your payroll program check-printing module, nor do you want to hunt for a contract programmer just to use your new hardware features. But all is not lost, because you can easily make this program send whatever character sequences you want. Here's how it works:

The CONTROL program listed here may be modified with the Dynamic Debugging Tool (DDT) supplied by Digital Research with their CP/M-80 operating system. There are two things to be modified within the CONTROL program: the device being controlled and the command being sent.

The CP/M-80 operating system can send one character at a time to your console, to an auxiliary device (usually a modem), or to your printer. The System Function number selects the destination; the console is two (2), the auxiliary is four (4), and the printer is five (5).

While the specific command character sequence is determined by the needs of your hardware device and what you want it to do, the format of the command is constant. The CONTROL program expects a one-byte count of the command characters, followed by the actual text to be sent to the device. To send a formfeed to your printer, the command length would be 1 and the text would be the formfeed character. Since DDT expects its data as base-16 numbers (hexadecimal), a formfeed command is <u>01 0C</u> for most printers.

Create the prototype CONTROL program with your system editor, following the instructions in Figure 1. Then create a FORMFEED program by typing what is <u>underlined</u>:

```
A>DDT CONTROL.COM
NEXT PC
0100
0200
- S0101              select which device is controlled
0101  05 05          (02=console, 04=aux, 05=list)
0102  21 .           (a period stops memory substitution)
- S0114
0114  00 05          (enter the command length and text)
0115  00 0C
0116  00 .           (stop entry with a period)
- G0000              (reboot, leaving program in memory)
A>SAVE 1 FORMFEED.COM
```

You may want to include your configuration programs into batch files for the Digital Research SUBMIT utility. Select the proper line size and character width and then print checks, for instance. Changing your work from a series of stops to a procedure avoids errors, minimizes training, and keeps things simple. Isn't that why you got the computer in the first place?

The utility programs you create this way won't let you vary line spacing or character widths within a single application program, but they do provide the ability to preset the hardware features you want to use. Now you can take control to get your money's worth from your system.

Figure 1 — How To Create CONTROL Program

You can create a copy of the prototype CONTROL program by using the standard utility programs furnished by Digital Research with their CP/M-80 operating system. This initial CONTROL program is then 'patched,' or modified, to generate hardware-specific control code sequences for your console, printer, or modem. The operator input (what you type) is <u>underlined:</u>

```
A>ED CONTROL.HEX
NEW FILE
*I
:100100000E052114014605F8235EC5E5CD0500E185
:10011000C1C30601000000000000000000000054
:100120000000000000000000000000000000000CF
:100130000000000000000000000000000000000BF
:100140000000000000000000000000000000000AF
:1001500000000000000000000000000000000009F
:1001600000000000000000000000000000000008F
:1001700000000000000000000000000000000007F
:00010000FF
↑Z
*E
A>LOAD CONTROL
FIRST ADDRESS 0100
LAST ADDRESS                    017F
BYTES READ                      0080
RECORDS WRITTEN                   01
A>
```

Figure 2 — Sample Control Sequences

Here are control sequences for a few popular printers, starting with the length of the command text (substituted at memory location 0114H).

Function	Command
10-pitch for Anadex	02 17 12
12-pitch for Anadex	02 17 14
6-lines-per-inch for Anadex	02 1B 48
8-lines-per-inch for Anadex	02 1B 49
10-pitch for C Itoh	02 1B 4E
12-pitch for C Itoh	02 1B 45
17-pitch for C Itoh	02 1B 51
Proportional-pitch for C Itoh	02 1B 50
6-lines-per-inch for C Itoh	02 1B 41
8-lines-per-inch for C Itoh	02 1B 42
Begin emphasized print for C Itoh	02 1B 21
Cease emphasized print for C Itoh	02 1B 22
Begin enlarged print for C Itoh	01 12
Cease enlarged print for C Itoh	01 14
Begin underlined print for C Itoh	02 1B 58
Cease underlined print for C Itoh	02 1B 59
Alphabetic character set for C Itoh	02 1B 24
Greek character set for C Itoh	02 1B 26
Graphics character set for C Itoh	02 1B 23
10-pitch for Epson MX80/MX100	01 12
12-pitch for Epson MX80/MX100	01 0F
6-lines-per-inch for Epson MX	02 1B 32
8-lines-per-inch for Epson MX	02 1B 30
Begin emphasized print for Epson MX	02 1B 45

(continued on next page)

Cease emphasized print for Epson MX	02 1B 46
Begin enlarged print for Epson MX	01 0E
Cease enlarged print for Epson MX	01 14
USA character set for Epson MX	03 1B 52 00
French character set for Epson MX	03 1B 52 01
German character set for Epson MX	03 1B 52 02
English character set for Epson MX	03 1B 52 03
Danish character set for Epson MX	03 1B 52 04
Swedish character set for Epson MX	03 1B 52 05
Italian character set for Epson MX	03 1B 52 06
Spanish character set for Epson MX	03 1B 52 07
10-pitch for InfoScribe	02 1B 36
12-pitch for InfoScribe	02 1B 38
10-pitch for TI Omni-800	02 1B 36
12-pitch for TI Omni-800	02 1B 37
6-lines-per-inch for TI Omni-800	02 1B 34
8-lines-per-inch for TI Omni-800	02 1B 35

Figure 3 — Assembler Language Source Program

```
; CONTROL by Steven Fisher, CDP — device control routine.
; Used to set device attributes, like baud rate.
; For CP/M-80 systems, version 1.x or later.
;
; TO CREATE:          ASM CONTROL
;                     LOAD CONTROL
;
; TO CUSTOMIZE:       DDT CONTROL.COM
;                     S0101
;                     xx              ; 2 = con:, 4: = pun:,
;                                     ; 5: = lst:
;                     .               ; (MP/M has no pun:)
;                     s0114
;                     xx              ; length of command text
;                     xx              ; enter command text
;                     xx
;                     .               ; end with a period
;                     G0000
;                     SAVE 1 yourname.COM
;
BASE      EQU         0000H           ; bottom of memory
                                      ; segment
SYSTEM    EQU         BASE + 0005H    ; entry point for system
TPA       EQU         BASE + 0100H    ; transient program area
DSPLYF    EQU         02H             ; display char on console
AUXOUF    EQU         04H             ; send char to auxiliary
PRINTF    EQU         05H             ; print char on printer
SEND      EQU         PRINTF          ; this controls printer

          ORG         TPA             ; where the program
                                      ; starts
SNDCTL:   MVI         C,SEND          ; function to send
                                      ; to device
          LXI         H,CMDLEN        ; address of length
                                      ; to send
          MOV         B,M             ; get length of text
SENDIT:   DCR         B               ; when minus, no
                                      ; more left
          RM                          ; if so then return to CCP
                                      ; to avoid delay of reboot
          INX         H               ; point to next character
          MOV         E,M             ; prepare to send it
          PUSH        B               ; save count and function
          PUSH        H               ; save address of
                                      ; this char
          CALL        SYSTEM          ; send a character
          POP         H               ; get address of byte sent
          POP         B               ; get count and function
          JMP         SENDIT          ; check for more
;
CMDLEN:   DB          0               ; how many control bytes
                                      ; control data begins here
          DB          0,0,0,0,0,0,0,0,0,0,0
          DB          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
          DB          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
          DB          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
          DB          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
          DB          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
          DB          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

          END         SNDCTL
```

OOPS! OOPS!

In last month's article *A Review of Alpha Software;s Data Base Manager*, p.31, Table 3 was not included. Here it is in its entirety.

Table 3—Data Management Capabilities

A. Underlying Data Model.

1. Data types.
Alphanumeric only. Type can not be specified.

2. Relationships.
None exist as part of file definition.

B. Functions provided.

1a. Data dictionary maintenance.
No data dictionary exists. A header file is used to record field names and lengths. Once established, only the field names can be changed.

1b. Data reorganization and conversion.
No facility provided for either.

2a. Data entry and editing.
Uses Basic "INPUT" statement to read one field at a time from keyboard. No programmatic edits are provided, except for excessive length. Very poor operator interaction.

2b. Report generation.
Maximum of ten reports can be defined, one of which may be in mailing label format. Defined field length must be used (no truncation). One sub-total and one calculated field are allowed. Program will determine column positions or user may override. No facility to alter report format once defined. Suitable for only the most trivial reporting requirements.

3a. Data selection by predicate.
Six relational operators are available to compare a maximum of three fields with three constants in an "and" relation. Separately, a selection may be made by context or "sounds like."

3b. Data joining and relating multiple data sets.
No facility available.

3c. Calculation on data.
One of nine operations may be performed to calculate one field on a report.

4a. Data independent interface.
None provided.

YOU SPENT $4,000 ON A PERSONAL COMPUTER. FOR ANOTHER $12.50, YOU CAN GET YOUR MONEY'S WORTH.

Today's personal computers have an extraordinary range of capabilities.

For a variety of reasons, however, many business people

are unaware of just how much their computers are capable of.

As a result, they aren't realizing the full potential of their investment.

THE KEY TO GREATER PRODUCTIVITY IN A WORD: SOFTWARE.

Computers do the work. Software does the thinking.

Expanding the amount of work a personal computer can do is merely a matter, then, of gaining access to a broader array of software.

And the software programs available to business and professional people number in the *thousands.*

But where do you go to find them?

THE KEY TO SOFTWARE IN A WORD: *LIST.*

LIST is the first publication that puts software first.

It contains articles by some of the most respected names in the computer field. Written to help you get the most out of your personal computer. No matter what brand it is. No matter what you need it to do.

More importantly, *LIST* contains the *LIST Software Locator,*™ a comprehensive guide to over 3,000 personal computer programs—conveniently indexed by application, industry, operating system and hardware. You'll find detailed descriptions of applications software that pertains specifically to the type of business you're in. And the type of needs you have.

LIST is sold at leading computer stores and bookstores. Or, you can phone our toll-free number (1-800-821-7700, Ext. 1110) or send in the coupon below, and receive a copy by mail. The price, exclusive of postage and handling, is $12.50.

Which, when you think about it, is a pretty small price to pay for something that can maximize a much larger investment.

LIST *is published by Redgate Publishing Company, an affiliate of E.F. Hutton.*

I'D LIKE TO GET THE MOST OUT OF MY PERSONAL COMPUTER.

Please send me ______ copies of *LIST* at $12.50 a copy plus $2.00 each for postage and handling. (Tax will be added where applicable.)

☐ VISA ☐ MasterCard (Interbank No. ______)

Card No. ______ Exp. Date ______

Signature ______

Print Name ______

Address ______

City ______ State ______ Zip ______

Send to *LIST,* Redgate Publishing Co., 3407 Ocean Drive, Vero Beach, FL 32960.

Or phone, toll-free: **1 800 821-7700 Ext. 1110**

LIST™

The Software Resource Book For Personal Computer Users

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

Second Class Postage Paid
At New York, N.Y.

EXPIRATION DATE: 12/83