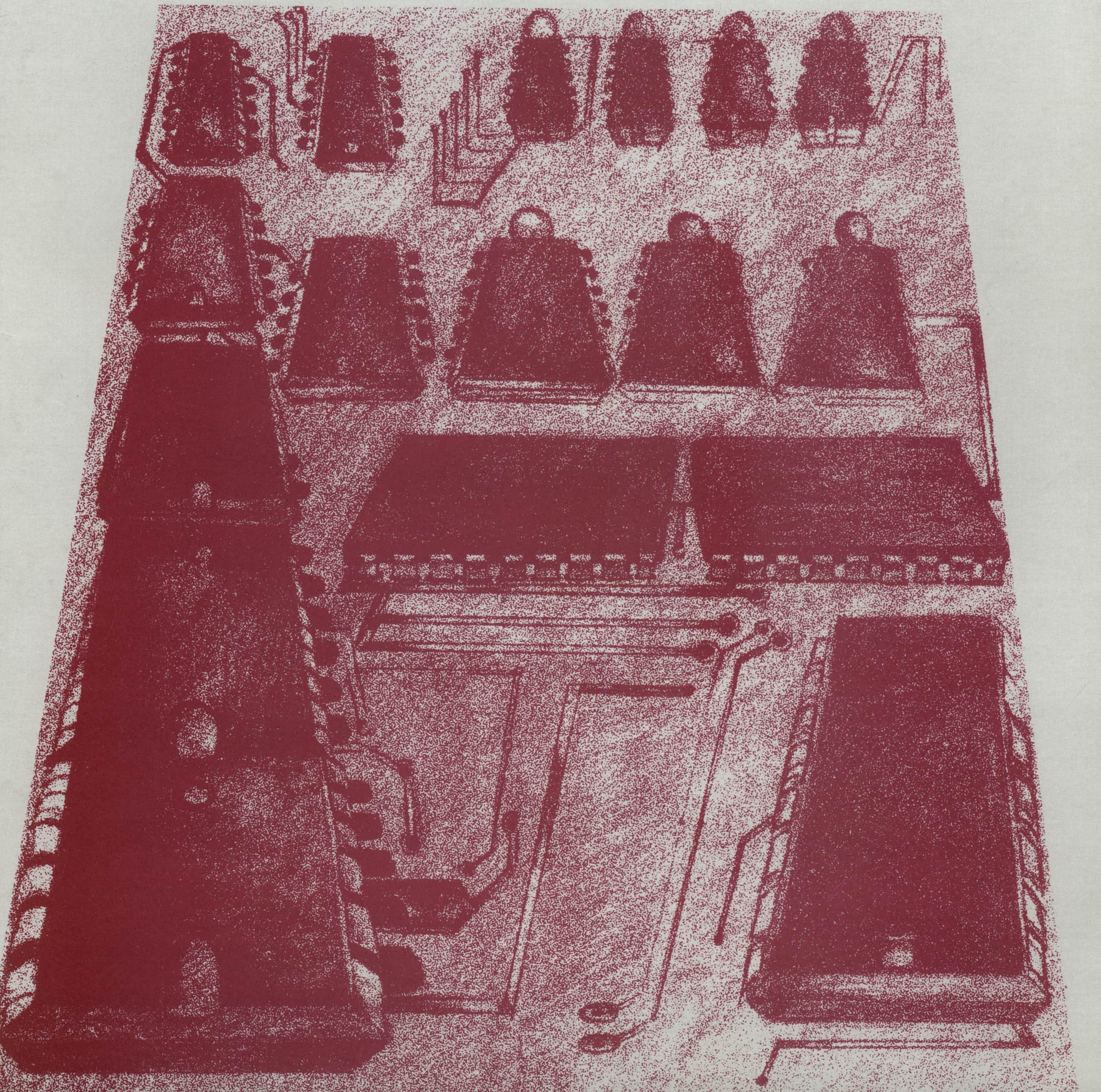
LIFELINES™

The Software Magazine™

$3.00 April 1983 Volume III, No. 11 (ISSN 0279-2575, USPS 597-830)



dBASE II™ made easy!

QUICKCODE™

The dBASE II Program Generator

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

FAST AND SIMPLE

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

NO PROGRAMMING REQUIRED

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

YOUR CONTROL

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

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

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

FOX&GELLER

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

QUICKCODE trademark of Fox & Geller, Inc
dBASE II is a trademark of Ashton-Tate

LIFELINES™ The Software Magazine™

Publisher: Edward H. Currie
Production Manager: Harold Black
Art and Design Manager: Kate Gartner
Operations: Carolann Abrams
Typographer: Harold Black
Software Consultant: Susan Sawyer

Managing Editor: Susan Sawyer
New Versions Editor: Lee Ramos
Technical Editor: Al Bloch
Advertising Manager: Carolann Abrams
Cover: K. Gartner
Printing Consultant: Sid Robkoff/E&S Graphics

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 Ave., New York, N.Y. 10028.

Opinion
Editorial Comments

Edward H. Currie

**Form Over Substance...
And Substance Over Form ...**

Recent computer shows in San Francisco reflect the fact that heavy attendance at such shows does not in and of itself imply that such extravaganzas are commercially worthwhile.

Vendors at the recent West Coast Computer Faire, which has been thought to be something of an institution, were overheard complaining about the type of potential customers who turned out for this show. Software and hardware discounters abounded and booths were filled to their capacities with show attendees wielding joysticks. An auctioneer bellowed in the background as people pawed through boxes of integrated circuits, cables, etc.

The Heathkit booth was packed with spectators looking for the Hero, Heath's entry into the robot market. However the units had been hauled away for a seminar and when they finally did appear they were under the purview of Heath salesmen who preferred the sounds of their own voices to demonstrating what the thundering hordes had waited for in vain. The highlight of one such presentation occurred when the Heath representative demonstrated that Hero knew the time (if you typed in enough stuff on the integral keyboard).

The oldtimers at the show roamed the aisles in a futile search for some new hardware or software innovation. Instead what they found was that the West Coast Faire had degenerated into the West Coast Arcade Game Show!!!

Book vendors were also there in strength and featured a number of books on Ada and some new offerings on UNIX. We'll review some of these in future editorials.

Anyway it's becoming clearer why Jim Warren is looking to sell his interest in the show. The bytes are on the wall. NCC is also losing its grip and has been largely replaced by COMDEX as the important show for microcomputers. Long live COMDEX!!!

PC World Day at the West Coast Faire culminated in the most fantastic party this industry has seen to date. The trend towards such lavish parties was begun by Microsoft and seems to grow with each show. A number of fine seminars and panels were sponsored by PC World and may well represent the only significant contribution of such shows other than bringing the various members of the industry into close proximity.

The second edition of PC World is out and this publication is well on its way to establishing new standards in microcomputer publications. David Bunnell has done it again for the third time!!! Its available at the newsstand so pickup a copy ... you'll be pleased.

As mentioned in last month's column Kathy McMahon has graciously agreed to provide a monthly column for those of you just entering microcomputerdom. She's on vacation this month but her first column will appear in the May issue of the Software Magazine. Watch for it. It's one column which can be read by all of us each month. Her charter is to give away all the secrets!!!

There's a rather interesting new book out on the C language that you should consider for addition to your library. As you know C is rising in ascendancy in the celestial hemisphere of microcomputerdom as THE language of the future. "Learning to Program in C" by Thomas Plum is published by Plum Hall, 1 Spruce Avenue, in Cardiff, New Jersey 08232. (609-927-3770)

Interestingly enough this particular treatment is completely self-contained and used as the basis for a course given throughout the United States. This text is a tutorial amd not the typical reference text on C. Plum's stated intent is "to give you the information you need to be a competent programmer in a real software engineering environment". If you are interested in the traveling classes in C or workshops in UNIX and advanced C topics, contact Plum Hall for the schedule. There are additional publications available as well, so ask for them when inquiring about the seminar. There is one particularly disturbing development in the C world and that has to do with documenting of C source code. The casual collector of public domain C source may in fact successfully compile, link, and yes Ladies and Gentlemen, even run such programs.

The problem arises because of the tendency to use "include" statements for external function calls. Often the external functions have been modified over time in ways which are not consistent with the various programs which might call the function. It's not at all unusual to find either at compile time or link time that some external function or routine is nowhere to be found. This is particularly distressing when you discover that some files provided with submit programs won't compile. Typically there is no discussion of external functions or CRL files which are needed to successfully compile, link and execute C programs.

Come on fellas, how about a little more attention to detail and a little better documentation in source and documentation files as to what is needed to use the many exciting public programs available in C.

The integrated packages such as Context MBA and "123" by Lotus continue to get the lion's share of the attention of the trade press as evidenced by the exhibits at the West Coast Faire. These packages, while well designed, are not likely to continue to dominate the market's attention for long, however. The microcomputer has far too great a potential to end up with such restrictive software. The future for micros lies in the areas of data movement, i.e. telecommunications, and data manipulation,

(continued on page 35)

Feature

A Review of the Spellbinder Word Processor

Walt Jung

Introduction:

This review is on the Lexisoft, Inc. word processing and office management system, Spellbinder (a trademark of Lexisoft, Inc.). The program is available for a wide variety of 8 bit CP/M-80 microcomputers, as well as in 16 bit CPM-86 and MSDOS formats, for the IBM PC and ZDOS for the Zenith Z-100. The version of the program reviewed here is 5.12, as configured for a Heath/Zenith H/Z-89. Table I summarizes the program's particulars.

My experience with Spellbinder began in the fall of 1981, when I was looking for a powerful CP/M word processor to replace a previously used program. I needed full support of the Diablo printer's capabilities for my technical writing, and the fact that Spellbinder was available configured for the H-89 and included the macro features, as well as effective screen editing and printing, sold me on it. The original version was V5.04, which I later upgraded to V5.12 in mid 1982. V5.12 added some additional macros, an improved command structure, a type-ahead buffer, an improved help system, a potent CP/M-like command mode, and miscellaneous less obvious enhancements.

For those unfamiliar with this program, I would like to make a point that Spellbinder is not likely to be suitable for a novice computer user, or anyone without some familiarity with word processors.

This is simply because the program is extremely powerful not only in terms of editing, but printing as well, as it literally makes a letter quality printer do all the tricks. If these two factors are not sufficiently intimidating, throw in the macro programming features, and you can overwhelm many of us. And that's the point: it takes more than the average technical skill to realize the full potential of this program, and all of its features. There are no menus to hold your hand, you wing-it free form, and the commands have options on options. If you don't like them, you can even write your own!

Having used the program for a length of time, I find that the tendency towards being overwhelmed does dissipate after use. The nicely chosen defaults let even the less experienced manage, without the problem of choosing variables, and, once the commands are mastered, you may find the program useful not only for word processing, but also assembly language text editing. Spellbinder is not highlighted as such, but I find it quite useful, with the rigorous search/replace, read-and-insert functions, at arbitrary cursor locations. The print-to-file function makes nicely-formatted CP/M-listable ASCII files (with tame high bits, too).

What does a word processor do?

Before jumping into the discussion on Spellbinder, it is appropriate to consider what functions a word processing package is expected to do. With such a frame of reference, we can compare with more meaning one *versus* another, and weigh overall which one best suits a particular need.

Given the background of Ward Christensen's series of text editor-reviews, *Lifelines* readers are already familiar with most of the general text editing criteria. However, an editor is not simply an editor when it comes to word processing, even if we restrict the discussion to full screen editors. A word processor, to be truly termed such, includes not only the screen editor necessary for fast and efficient text entry and editing, but also the ability to format documents for printing, and finally, to print the document.

Thus, while all editors edit to various degrees, and most can also imbed format commands for external printing, only full-fledged word processing programs do all three, and are optimized for such. Differences in exactly how and to what degree a given package realizes these criteria are what separates different programs. With the highly competitive world of today's CP/M-based word processors, some very powerful programs exist for the micro, many of them featuring additional bell-and whistle capabilities such as spelling checkers, merged printing capability, macro ability, indexing, footnoting, *etc.* In reality, many of them are actually program *families*, involving a series of related modules which work in concert with the main program. Sometimes the set is available as a one price package, sometimes it is not, so comparisons must be made carefully here.

This review will focus on the three basic capabilities of word processing; editing, formatting and printing, and will discuss how the Spellbinder package realizes these requirements, since it is these functions that are common to other comparably equipped word processors. The additional options available with Spellbinder are treated as well.

Installation:

Installing Spellbinder is relatively easy, as the distribution disc comes with a COM file already configured for your hardware as well as a healthy assortment of ready to go macros (see Listing 1). Here, the file HSB.COM is a self-patching version of SB for the Heath environment, and when called, prompts you with a series of questions to select either the standard keyboard or function keys, row/column numbering (on/off), printer type, and help guides (on/off). Custom keytops are available, if desired.

After completion of the menu you enter the program, and are told to "SAVE nnn SB.COM" on exit. The "nnn" varies from 102 to 110, depend-

(continued on next page)

ing upon your options. Thus the resulting fully configured COM file will be 26K or more; the smaller versions will yield more work space, of course, and other machine versions of SB.COM will vary in size. You'll want to keep the help guides initially, and later re-configure the program if and when they are not necessary to gain memory. Programmers can easily further customize Spellbinder with the IOS.ASM file, which allows customized screen prompts, keyboard and terminal configurations, and printer drivers. Spellbinder supports three different general classes of printer: device 0 is a precision or letter quality unit, device 1 is a dot matrix unit (ASCII and ESC characters), while device 2 is a CP/M LST: device (ASCII only). All are selectable from within the program, and may be SAVEd as default configurations.

Documentation:

The documentation which comes with Spellbinder is divided into a number of loose-leaf sections: a pull-out 73-page tutorial manual, a quick reference command summary section, a general reference section, a macro feature section, a peripheral interfacing section, and a section on how to write your ownmacros, called "M-Speak Programming'. The table of contents indicates general access to specific broad topics, but the manual is not indexed. Both the main manual and the tutorial sub-manual are testimonials to Spellbinder's print capabilities, since they employ such features as L/R page numbering, footers and headers, all interspersed with standard boldface, justification, underscore *etc.*, and the unique Spellbinder 2 column print macro, driving a Sanders typographic quality printer.

Mechanics aside, however, the content and arrangement of thedocumentation I would say is better than average. The tutorial manual begins by bringing up the program, and leads through text entry and editing, the command mode, disk operations, formatting, and printing. Screen examples are used, and the commands are explained in context. The tutorial section will be most useful to the beginner; once some degree of familiarity is gained, the general reference section is likely to be used, and the quick reference very often.

While no command summary reference card is supplied with Spellbinder, the program has a good balance of help available. Integral to the COM file are optional 3 line command summaries, for both COMMAND and EDIT modes, which serve as on-screen reminders for less experienced users. If not desired, the help summaries can be toggled off. If more detailed help is desired, the HE command can be used to read in any of the detailed help files from disc. (Note that this command can also read in any text file, and is one way to check a disc file for content, without exitting from the program.) My only gripe of a serious nature with regard to the documentation is that the manual is not indexed, so it is difficult to find a specialized topic.

Using the program:

Once configured for your system, when called Spellbinder signs on with the version number, your serial number, and copyright notice, prompting you to hit any key. Striking any key other than ESC at this point enters you directly into the EDIT mode, ready for text entry. The ESC key is (here) a file recovery command, which will retrieve from memory a file just previously active, after a crash. You'll only have to use this feature once to love it; it will bail you out from re-entering text which would otherwise be gone forever.

Spellbinder has two distinct operating modes, the COMMAND mode and the EDIT mode. The COMMAND mode is for I/O related commands, printing, macros, search/replace operations and the larger scale operations in general. The EDIT mode is primarily for text keyboard entry and general editing. However many commands overlap between COMMAND and EDIT modes, by use of both control and ESC character sequences. There are no menus for either the COMMAND or EDIT modes to select a given command, such as for example with Wordstar. Some may view this as a lack of friendliness, but programmers are apt to like this stand-on-your-own-feet approach. It won't hold your hand, and you alone determine how much of the inherent power you realize from the program.

In both main modes, the top-screen status line gives the current cursor position by line and column, and the currently operational mode is indicated by either a leftmost EDIT or COMMAND:. For example, listing 2 illustrates the EDIT mode screen, with the two Help lines at the bottom. These 3 line displays summarize the EDIT mode control character commands (1ST set (top)), and the ESC commands (2ND set (bottom)). The underscored characters appear in reverse video on the screen.

Listing 3 shows the COMMAND mode screen, with the main Help line illustrated. Entering any of the numbers shown from the COMMAND: prompt fans out to further Help lines, with further detail on the individual commands (not shown). When a COMMAND mode command is executed, the Help lines dynamically prompt you for input; drive, FILENAME.TYP, *etc.* (Nice!). As can be noted, this system detracts little from the screen workspace, and if so desired, the help displays can be toggled off (HE0 command). Detailed help is available from disc, if desired (HE command).

For EDIT mode text entry, Spellbinder operates continuously in a wordwrap mode with overtype, and stores word processing files as continuous ribbons of text, with paragraphs delimited by hard carriage returns (CP/M CR/LF pair). Within paragraphs, individual lines are delimited with soft CR's, in the form of control N's for end-of-line. While this makes Spellbinder word processing files non-printable from CP/M, they may also be written to disc with an optional switch (/1) for CP/M compatibility. (Don't do this if you plan to edit later, though, as you will then need to remove the CR/LF's!)

With this scheme of operating, Spellbinder files are easily manipulated for any desired print format. Unlike some word processors, Spellbinder prints from memory, not disc. Once a document has been formatted as desired, it can be immediately printed, even without saving to disc (not a good habit, however, so don't take this observation as a suggestion). Details of formatting and printing are covered below, under printing.

A file under work is displayed in the COMMAND and EDIT modes showing the line length by hard carriage returns, where present. For example, listing 4 illustrates a sample CP/M ASCII file, where the "<" character represents the CR/LF pairs in the file. Were this a Spellbinder word processing file type, the various <'s within the paragraph would not be present, and the file itself would have soft CR's in their place, which do not show on-screen. This allows easy editing, and relining to any line length. If and when you should need to edit standard (ASCII) files, with the displayed (real) CR's, you always know the length of your lines.

Commands

Spellbinder has a very powerful command set, and many of the commands are functional from both COMMAND and EDIT modes. Further, COMMAND mode commands can not only be stacked, but also written into macros. This discussion will briefly summarize most of the major commands between COMMAND and EDIT; macros are discussed separately.

COMMAND mode commands

In the COMMAND mode, Spellbinder has commands related to cursor movement and/or editing, printing, and file I/O. Since printing commands are also discussed under that section, just those commands related to printing already formatted text will be discussed here.

Editing-related commands concern cursor movement, deletion, block moves, and search/replace operations. Cursor movement can be forward or backward in the file, to a mark or to the end of file, or by "n" lines. Deletions can be from the cursor position to end of file, "n"lines, or all text. The potentially catastrophic forms employ prompting, like "REALLY?(Y/N)." Block definition is from the cursor position to a mark, to end of file, or "n" lines, with movement into a hold buffer. Attempts to write to a non-empty buffer result in a prompt, before the buffer can be updated. The held block can then be re-inserted one or "n" times, at any cursor position in text. The hold buffer itself is size limited only by the computer's available workspace memory, less the size of the file being edited.

Search/replace in Spellbinder is one of the more impressive aspects of the program. It can be a simple search and find, queried search and replace, as well as an automatic search and replace. Commands can be either stacked or interactive, and operable on text in memory only, or the entire file (including disc). Modifiers to the syntax allow search/removal, string modifiers for whole words and case, or both, as well as wildcard searches for ? characters, numbers, non-numbers and non-letters.

File manipulation with Spellbinder is generally well handled in almost all regards. Files are not necessarily opened, until you wish to read in previous text, or write out just entered or edited test, and you cannot exit the program with files open. Writing to a previously used FILENAME.TYP will create an automatic backup (FILENAME.BAK), and a second write bumps the first BAK file (the originally named file). A write to the same filename need not be explicitly named; if you wish the same name, the "/" command will automatically use the correct name, and handle the housekeeping. An aborted write to a full disc does not dump your keystrokes to oblivion; it just provides a calm error message. Discs can be changed while in the program; one simply logs in a new disc by doing a directory, after which I/O may be performed.

Writing out a file can be from any cursor positon, and by "n" lines or all text, and multiple writes to the same file are even possible. A conventional (complete) file write is via W/WD (write/write done) which writes out the entire file with the cursor at the top, and closes it. The text remains in memory for another write if desired. If no further work on the text is desired, it can be written out and cleared from memory with the GD command. In either case, you stay in the program.

Reading files from disc can be either complete via the R command, or in terms of "n" lines, via Rn. If text is already in memory, the new text goes at the end of the first. Alternately, an RI command can read and insert a file at an arbitrary cursor position, or RIn will read and insert "n" lines at a cursor position. The limit of "n" is 250 lines.

The above discussions cover files which fit entirely in memory. With a 64K system, the amount of workspace Spellbinder allows is on the order of 25K, so it behooves one to use all RAM possible. For files larger than memory, disk buffering is used, with a write file being opened for the edited file. As the file is entered into memory, the remaining workspace RAM is shown on the display. (The "M" command may also be used to check remaining memory at any time, from the COMMAND mode.)

The file handling for larger-than-memory files is automatic only in the sense that Spellbinder recognizes the need for it, and prompts you for a write filename when asked to read the larger than memory file. Segments of the file are read in sequentially, edited, then written out with the G command. This is a one way operation, and what has been written can't be retrieved and further edited, except by finishing the operation and restarting.

This method of larger than memory file handling is, as noted, uni-directional, and it can be a real source of irritation, if you are used to the transparent disc buffering available with editors using swap files. My feeling regarding the oversized files is that the operation(s) should be made transparent, to allow such files to be manipulated with the same degree of ease and flexibility as in-memory files. This would hopefully include file naming and I/O, where the combination of features is actually one of Spellbinder's strong suits. (Note: Discussions with Lexisoft indicated that some changes in this area are being considered.)

Spellbinder has a built-in disk directory, which sizes the files and reports the total disc file usage in terms of actual usage (not blocks allocated). The directory is not sorted, which provides more than a small problem of confusion with high capacity disks, even 8" double density. There is no means to do a selective directory, just all or none. Fortunately, there is a way around these limitations, which is covered below. Files can also be deleted from disk, but they cannot be

(continued on next page)

renamed or copied. (There is some question in my mind whether the latter two really are necessary of a word processor, as they seem more system utilities.)

Printing in Spellbinder is always from memory, allowing just entered text to be printed immediately. The P command prints one page, Pn "n" lines, and PA the entire file in memory. PG will print an entire file, from disc.

A variation of the print command I find extremely valuable is 'PO', which is the print-to-disc command. This provides a CP/M compatible ASCII file, which can be listed later with PIP or any comparable CP/M utility. The Y table must be set for LST: device, and the print features of such a device are supported, including space justification, underscore, boldface, *etc.*

The final COMMAND mode command discussed is one of the more powerful, and the one I feel puts Spellbinder in a unique class. This is a CP/M-like command mode, from within the program, called by 'C/'. From this mode, any CP/M COM file (20k or less) on the A: drive can be executed, such as STAT or PIP or your favorite directory program, you name it. The ostensible reason for this feature is to allow the companion program "Spellcheck" to be executed, without leaving Spellbinder. Of course "any COM file below 20K" allows other stand-alone spelling checkers to be used, as well.

However, the power of this command is that it gives you a CP/M command line without program exit or loss of text. This not only gets around the lack of directory flexibility mentioned earlier, but also greatly enhances the program's capability. However, parameters cannot be easily passed from the command line (ie, enter PIP for the "*" prompt, not PIP B:fil.1=C:fil.2), and when exiting this mode, your default drive will be A: (if different previously).

EDIT mode commands

The EDIT mode commands are highly terminal dependent, as would be expected from different keyboards and function key setups, from machine to machine. However, even in a given version, such as the Heath, one has a choice of using either the standard keyboard, or the special function key set. Any system using Spellbinder with the H89 or H19 is very much enhanced in operation by activating the function keys, as otherwise virtually all commands are control characters.

In the EDIT mode, text is entered continuously, with wordwrap. For insertion of new text the INSERT mode (an EDIT sub-mode) breaks away text below the cursor, and allows additional text to be appended from the break onward. Toggling INSERT back restores the text to continuous, including the newly inserted material. A quirk of Spellbinder which I do not care for is that it will not allow you to move backwards from the current line, while in INSERT. The screen defaults to 79 columns, but can be lined to other widths, and it will then show the width via a dotted vertical line. Horizontal scrolling is supported in the EDIT mode, for lines up to 159 characters. The screen will automatically rewrite as the line entered passes 80 characters, and indicate the left (or right) screen display by a /1 (or /2) on the line counter.

Cursor movement in Spellbinder's EDIT mode is very flexible, as it can be both forward or backward, and by character, mark, word, sentence or paragraph. The cursor mode is displayed on the status line, as in listing 2, which shows the default WORD mode. The cursor can also be moved up and down with the cursor arrows (or ↑K, ↑J). The cursor can also be SCANed, from the line extreme left to right, with alternate SCAN keystokes. Repeat is via the REPEAT key for the H89, or ↑R (standard keyboard).

Similar to the cursor movement in terms of flexibility are the EDIT mode deletions, which are also by character, mark, word, sentence and paragraph, from the cursor position. The paragraph deletion is prompted, but there is no "oops!" key function for any mode, so new users should be careful.

'Enhancement" in Spellbinder is a dynamic change in a print character, according to enhance character text toggles, and the special character entry in the Y table. This can enable underscore, boldface, shadow, *etc.* Enhancement may be entered into text in a number of ways. One of these is by a toggle which starts enhancement, with subsequent typed characters enhanced as long as desired, after which the toggle is turned off. Or, another flexible way is to use mode enhance, which enhances from the cursor position by the current mode. This is obviously the quicker way for existing text, particularly with the function keys used. Both these methods highlight the enhanced portions of the text on screen.

While screen rewrites are automatically done by Spellbinder as required, the screen can be manually re-written with the cursor at the top with the REWRITE command. Paging backward or forward in text is done with the PREV and NEXT SCREEN commands, but these commands are not preemptive. This can slow things up and be a distraction. There is no true line-by-line scroll capability *per se*, and this would be a desirable addition.

TAB settings are defaulted to 8 spaces, but can be set as desired, and decimal TABs are supported as well (TABs are changed as desired with the COMMAND mode "Z" command). A powerful complement to the standard TAB function is INDENT, which indents the current line to the next TAB stop.

The mark character used by Spellbinder is the "↑" character, which means it can be imbedded in text (its function), but it will not (normally) print. If desired, it can be forced to print with an enhancement technique.

The above described commands are executable either via control characters on a standard keyboard, or the function keys on the H89/H19 (and other terminals). They are summarized in Listing 5, with the underscored characters displayed enhanced (on screen).

In addition to the above commands, the EDIT mode also overlaps a number of COMMAND mode commands, by the use of the ESC key, followed by a single letter. For example, ESC T and ESC E put the cursor at the top and end of file, while ESC H and ESC U hold and unhold, and ESC F and ESC B go forward and back to a mark. Note that although there is apparent overlap, it is not redundancy, as key strokes are eliminated by avoiding the requirement to go to COMMAND from the EDIT mode, then back again.

Printing

Before actual printing, a document can be previewed for correct formatting using the print-to-screen feature of Spellbinder. This shows special characters such as underscore, boldface, *etc.* with highlighting, formatted line spacing, right justification (LST: device), and page breaks. While the display is not precisely like the actual final type (for display limitation reasons), it is a workable approximation, and prevents wasting actual printing. This preview function has built-in checks for suitable hyphenation, and will prompt you to correct errors as they are needed. Simply stated, if the file being formatted passes the preview test, it will print. This test can be either visual as described (V), in memory (J), on only one page, the entire file in memory, or the entire file including disc.

Formatting text with Spellbinder is similar in one sense to many editor/formatters, in that it uses line leading-period "dot commands'. However, that is a little like saying all beers are made with hops - - it doesn't tell much about how they taste! Spellbinder uses a single dot command string for most of the formatting commands, called the "Y" table. This Y table is in the form of .Y n1 n2, etc., with the period the first character of the line, to suppress printing. In addition, there is a related title/page format table, or YT table. The main Y table controls the major printing format functions, while the YT table controls titling, headers, footers, L/R page numbering, *etc.*

Listing 6 shows both the sample Y table itself (top), and also the menu by which the parameters are selected (bottom), with the defaults I use. Because of the defaults, one can actually use Spellbinder without ever referring to or altering the Y (or YT) tables. But, since that would shut off much of the power, it behooves the user to become comfortable with their use.

The first two entries on the menu cannot actually be altered dynamically by the Y table as shown at the top, but are manually selected when you set the program up. The menu is brought up from the COMMAND mode by a "Y" command for altering the Y table. The printer type can be any of three shown; the normal default is 2, but I use device 0. The destination is the assignment where your printer lives for your computer (10 for Heath). Once these two parameters are set, you should SAVE the version on exit, to avoid the necessity of reset whenever the program is recalled. You can of course SAVE multiple versions for different printers and port assignments, such as a dot matrix for drafts, a letter quality for final copy, *etc.*

The remaining entries in the table are set by walking through the menu (a CR = skip), and keying in the change desired. Many of these parameters are generally familiar, but some are unique. The print routine entry, for example, allows printing to be done either by the editing assigned line length, or by character. The latter is more useful, and when used the printed line length maximum will be equal to LINE WIDTH. MAX and MIN set the spacing between words, for right justification. Characters per inch is set by CHAR SIZE; from 0 = 8CH/inch, up to 3 = 15 CH/inch. Line spacing is via LF SIZE; from 0 = 3L/inch, up to 3 for 8L/inch. The SPEC CHAR is the selected character corresponding to a type of enhancement, which is toggled on-off, by an in-line text character pair for the enhancement.

The most powerful feature of this approach to formatting lies in its flexibility. Here, flexibility is a term which applies not only to the range of control menu selectable, but more so in the sense that Y (and YT) statements can be imbedded within the file by a simple "FY" (or "FT') command. This allows a Y table to be altered line by line if necessary, and later recalled from disc, along with the document. The Y and YT tables (and other formatting commands) are dynamically read by the program with the text, and the entire re-formatting process is transparent to the user.

In addition to the Y and YT table, Spellbinder also supports simple DOT commands such as .c = center, .r = remark, .e = FF, .h for headers, .t = vertical TAB, and so on. Along with these dot commands, Spellbinder also supports inline commands, which can be placed anywhere on a line. The mark character is the most obvious example, but there are a number of others, for super and sub script (both of these can be nested), firm hyphen, enhancement, absolute TAB (allows vertical alignment under proportional print mode), line tweaker, and printer control sequences.

A good number of the inline commands dynamically modify the one or more Y table entries, which extends flexibility even further. For example, there are commands for the change of the special character, the font, the line spacing, and the ribbon color. This allows powerful changes, even within individual words. There is even a "user defined" section in the IOS.ASM file, where one can write custom controls, and there is a user alterable space table.

For optimum control of word breaks and print appearance, Spellbinder has 3 kinds of hyphens: hard, soft, and firm, and two kinds of print formats: line and character oriented. The character oriented format uses the Y table entry for line width and will break lines only between words, a hard or soft hyphen, or conditionally, a firm hyphen. It is the generally preferred routine, particularly for justified text. The line oriented routine prints according to the line width set on the screen, and thus actual printed length depends upon this, and the letters used.

Printwise, a hard hyphen will always print, whenever it occurs. A soft hyphen (with the line routine) is used for wordbreaks if needed, and will print. Under the character mode, it will not print mid-line. A firm hyphen is used only with the character print routine, and will print if it occurs at the end of the line. Hyphenation can be one of the trickier parts of printing and formatting, but with the screen preview feature and prompted hyphen help, it is manageable and yet flexible

(continued on next page)

Obviously one of the major determinants of print appearance is how the space between words is handled under right justification. With Spellbinder, justification can be either incremental character spacing with an ordinary printer, or with a precision type, proportional in increments of 1/120". Further, the maximum and minimum limits on the distributed inter-word spacing allow good uniformity of text appearance, from line to line.

The total combination of printing support and range of control variables is most likely unique to Spellbinder, and will even drive a Sanders typographic printer (the manual is an example). Unfortunately, there is enough that could be said on printing to fill a small book, let alone a review, so we'll summarize by saying that you are not likely to be disappointed by the print capabilities of Spellbinder, once you've mastered them.

Macros

As mentioned above, COMMAND mode commands can be entered in 'stacked' form in Spellbinder, to create some quite powerful functions. Spellbinder calls such commands 'Autocommands'. An autocommand is defined as a sequence of commands, entered on one command line.

Simple and obvious examples of such stacking might be "T/W/WD" (go to top of file, open file for write, write out entire file in memory and close file). This one will do just that, after you answer the prompt for a file name. Another would be "H2/B10/D5/U" which holds two lines, goes back 10 lines, deletes 5, and inserts the previously held 2 lines.

Prefix an autocommand with the modifier 'n', for 'n' executions (up to 250), and you have a macro. Any number of commands can be stacked, as long as they don't exceed one line length, and the autocommand executes upon a CR.

Intervention can be used within an autocommand, by inserting '/I/', which will cause Spellbinder to prompt you. A good example is a multiple print command, "3 P/T/I" which will print 3 copies, but with

README.ART

Art	Description	File
Table I	Facts and figures	TABLE1.WS
Listing 1	Distribution files	LISTING1.WS
Listing 2	EDIT screen	LISTING2.WS
Listing 3	COMMAND screen	LISTING3.WS
Listing 4	COMMAND screen w/text	LISTING4.WS
Listing 5	EDIT command keys	LISTING5.SB (HC)
Listing 6	Y table	LISTING6.WS
Listing 7	Line number macro demo	HARD COPY
Listing 8	2 column print demo	HARD COPY

TABLE 1
Facts & Figures

Program Package:	Spellbinder Version 5.12; Word processing and Office Management System
Supplier:	Lexisoft, Inc. P.O. Box 267 Davis, California
Suggested Retail Price:	$495.
Operating Systems:	8 bit CP/M-80, OASIS, MP/M 16 bit CP/M-86, MSDOS, ZDOS
Memory Requirements:	48k (minimum), 56k practical
Disc Requirements:	1 drive, 150k or more
Disc Formats:	8" standard, plus a variety of 5"
Systems supported:	Televideo 910/920/950, Zentec, Adds Viewpoint, Volker Craig, Heath/Zenith Z-19 (89), TRS-80 II, Cromenco, Hazeltine, Interbute, Soroc (all CP/M-80) IBM PC (MSDOS, CP/M-86)
File Package:	Self patching COM file, IOS.ASM for customization, 13 macro programs, help file, terminal specific DOC file, sample macro files
Documentation:	Complete manual in binder, subdivided into 5 sections
Printer Support:	ASCII only (LST:), dot-matrix, precision (Diablo, Qume, NEC), and typographic quality (Sanders)

```
2CPRNT   .WPM  4k ADDIT    .WPM  4k ALPHA   .WPM 6k AUTOLF  .TAB 4k
BATCH    .WPM  2k BOILER   .A    2k BOILER  .LET 2k BOILER  .WPM 2k
CALC     .SMP  2k CALC3    .WPM  6k COL     .A   2k CUESORT .WPM 4k
CUSLIST  .DEM  2k FORMS    .WPM  8k HEATH89 .DOC 4k HELP    .HEP 8k
HSB      .COM 30k HTIOS12B .ASM 32k INSTALL .WPM 2k INVOICE .TEM 2k
KEYS     .A    2k KPHRASE  .WPM  2k LETTER  .DEM 2k LETTER  .TEM 2k
LINENB   .WPM  4k MMERGE   .WPM  4k MOVEIT  .WPM 6k ORDER   .TEM 2k
PCONTROL .TAB  4k SPACE    .TAB  2k SYS     .OVL 2k
          Drive B, user 0 contains 160K in 31 files with 266K free
```

Listing 1: Spellbinder V5.12 distribution files

```
EDIT          L 0001      C 001        *WORD*
↑
24 lines high (with help lines)
↓
_________________________________________________________________________
insert(E) indent(Y) cursor(HJKLS) Prev(G) Next(V) Dtab(Z) COMMAND(Q)
mode: change(O) forw(F) back(B) delt(D) enhance(U) Clear(C) 2ND SET(ESC)

_______/ 2ND SET(ESC)
Unhold Hold Top End Next Page Prev Indent clear Back Forward
```

Listing 2: Spellbinder EDIT mode screen with Help lines [1ST (top) 2ND (bot)]

```
COMMAND:          L 0001      C 001        *WORD*
↑
24 lines high (with help lines)
↓
_________________________________________________________________________
1: Disk 2: Search 3: Move/Delete 4: Print 5: Tables Exit(X) Help(HE)
```

Listing 3: Spellbinder COMMAND mode screen with Help line

```
COMMAND:          L 0001      C 001        *WORD*
Installation:<
<
     Installing Spellbinder is relatively easy, as the distribution disc<comes
with a COM file already configured for your hardware in general, as well as a
healthy assortment of ready to go macros (see listing 1). Here, the
file<HSB.COM is a self patching version of SB for the Heath environment,
and when<called, prompts you with a series of questions to select either the
standard<keyboard or function keys, row/column numbering (on/off),
printer type, and<help guides (on/off).<
<
```

Listing 4: Spellbinder COMMAND mode screen with sample text (no Help line)

pauses between them, to allow you load new paper.

You can define your own macro, by use of the COMMAND mode "AT", which will place entered text from the cursor position into the macro buffer. For example, if you key in the sequence "S//*↑", (where ↑ is the CR), then place the cursor over "S", and do an AT, you have a mini-macro which prints a *. Strike the CONTINUE key in either EDIT or COMMAND modes, and you repeat the macro operation. Do an "A40" from the COMMAND mode, and you have a line of 40 *'s.

To edit the macro, do an AT to recall it, modify, then an AT again to move it back to the macro buffer. Edit to a 'S63//*//S//↵↵', and you have a mini-macro which prints a line of *'s, with terminating CR.

Obviously, these short examples are rather trivial, but they are purposely chosen to illustrate how close macro writing is to stacked commands. These two mini-macros are in fact just stacked commands, but they are functional, and could be saved to disc as a macro, with a file type of 'WPM'.

Loading a disc macro is done with the command 'AD', and the entry of the macro name. This loads the macro, and starts execution. A macro finished execution remains in the buffer, until changed by a new AD command, or edited via AT. It can be re-executed (once) by the command 'A', or "n" times by the command 'An'; or, once (also) by the CONTINUE key, as mentioned.

The manual devotes 10 pages to much more detail of the macro programming language, which can be quite a powerful tool in the hands of an experienced programmer, as many custom utilities can be implemented. An M-speak programming manual is separately available for $25.

The Spellbinder package comes with 13 macros, which are ready to load and excute (see Listing 1, *. WPM). These macros are designed around a variety of office tasks, and enhance the utility of the word processor quite considerably. As the names imply, there are macros for file line numbering (useful for legal drafts), forms creation and fill-in, boiler plating, batch printing, sorting, merging of data lists into letters for mailing, addition of columns and rows of

(continued on next page)

numbers, a calculator, a key phrase locator, columnar movement, and 2-column printing. Install is a configuration macro, necessary for some of the others.

Two of the macros demonstrate the utility of the language, in printing tasks. The LINENB macro is shown in Listing 7, and numbers each of the lines to be printed, with all of the normal Spellbinder formatting possible (shadow print shown here). The 2CPRNT macro is shown in Listing 8, printing the same file, but in a 2 column format. These (and other) macros have configuration menus, which prompt you for parameters, and allow formatting to be optimized for the specific task.

Support:

Experience with regard to support of the program has been good. My dealer, Ray Massa of Studio Computers (999 South Adams, Birmingham MI, 48011) has always been helpful answering questions, as has Perry Gee and his staff, at Lexisoft. Licensed holders of Spellbinder may update older versions of the program to a new revision, by returning the distribution disc with $50.

Summary:

In summary, I would have to say that anyone looking for a powerful word processor package should consider Spellbinder, qualified in light of your own intended usage and general experience with word processors. If you are comfortable with free-form commands, it could well suit you. But, if you are used to the environment of menu driven systems, it is not likely to be your cup of tea.

While the review comments on the program's learning difficulty are largely intended to warn against the assumption that use might be trivial, it cannot be too hard, as my teenage son Mark uses it proficiently, for term papers! I have used it for countless letters, my latest book *(IC Timer Cookbook, 2nd Edition)*, as well as many articles over the last year or so.

Compared to some other currently available packages, it seems that two function(s) not presently supported in Spellbinder V5.12 are indexing and footnoting. Since the field is such a competitive one, it should not be long before such features are considered standard items for word processors, and they will likely be considered in the next release.

The review has not treated 16 bit versions of the program, which should offer healthy bonuses of speed, with the other general characteristics as noted. This will enhance overall operation, as well as the execution of some the macros, which (on my 2mHz H89), are slow. Similar comments of relative improvement would apply to the higher speed Z80 processors, in CP/M-80.

Editors and word processors do tend to become quite personal attachments with heavy use, and we all tend to become comfortable with one of them over others. In actual practice, there is no such thing as the perfect editor or word processor; they *all*

STANDARD CONTROL SET:

MOVE CURSOR >>>	LEFT(**H**)	DOWN(**J**)	UP(**K**)	RIGHT(**L**)	SCAN(**S**)
MODE EDITING >>>	FORW(**F**)	BACK(**B**)	DELT(**D**)	ENHANCE(**U**)	CHANGE(**O**)

INSRT/CLS(**E**)	INDENT(**Y**)	SOFT HYP(**N**)	REPEAT(**R**)	DEC TAB(**Z**)
DELETE (**DEL**)	CLEAR(**C**)	REW TOP(**T**)	ENTER ENH(**W**)	MARK(**X**)

HEATH/ZENITH FUNCTION KEY SET:

f1 - INDENT
f2 - SOFT HYP
f3 - MODE ENHANCE
f4 - ENTER ENH
f5 - PREV SCREEN
ERASE - NEXT SCREEN
BLUE - CONTINUE
RED - MARK
WHITE - REW

CHANGE	UP	DELT
LEFT	SCAN	RIGHT
MODE BACK	DOWN	MODE FORW
EDIT/COMMAND	DEC TAB	INSRT/CLS

Listing 5: H/Z89 EDIT mode command set.

,Y 1 90 110 2 0 1 0 65 2 2 1 0 30 7}-Note: Begins with ","; "," used to print!

PRINTER TYPE	0	precision(0) dot matrix(1) system(2)
DESTINATION	10	default printer(0)
PRINT ROUTINE	1	line oriented(0) char oriented(1)
PRINT LENGTH	90	length of printed text (90 = 9 in.)
FORM LENGTH	110	length of paper (110 = 11 in.)
PAGE EJECT	2	stop each page(0) space(1) form feed(2)
LEFT INDENT	0	indent from left margin in tenths
SPACING	1	single space(1) double(2) triple(3)
JUSTIFICATION	0	left(0) right just(1) center(2) right(3)
LINE WIDTH	65	print width (65 = 6.5 in)
LINE FEED SIZE	2	6 per inch(2) 8 per inch(3)
CHARACTER SIZE	2	pica(1) elite(2)
SPECIAL CHAR	1	shadow(0) underline(1) bold(4)
PROPORTIONAL	0	fixed pitch(0) proportional(1)
MAXIMUM SPACE	30	set for hyphenation check
MINIMUM SPACE	7	

Listing 6: Spellbinder Y table with defaults (top), and menu (bottom)

have weaknesses and tradeoffs, to one degree or another. There are very few absolutes or panaceas in the real world, but there certainly are such things as poor decisions, arising out of incomplete understanding of the various programs.

Spellbinder has some areas of performance which could be improved, as are noted in the text. Of these weaknesses, none have been catastrophic to me, nor have they totally resisted bypass. However, I can (for example) appreciate the possibility of a new user becoming frustrated with the larger-than-memory file handling, as it is simply difficult to use. It is highly important that any software tool used intensively be both comfortable and efficient, otherwise it becomes a burden, not a boon. In an overall sense, I consider Spellbinder to be a very powerful word processing tool, and a highly useful one. Hopefully, the detailed comments of this review will communicate whether or not it can also become that, for you. I can be reached for questions and comments in care of *Lifelines*, or via modem at the BHEC RCPM, (301)-661-4447.

```
1:          Installing Spellbinder is relatively easy, as the distribution disc
2:  comes with a COM file already configured for your hardware in general, as well
3:  as a healthy assortment of ready to go macros (see listing 1). Here, the file
4:  HSB.COM is a self patching version of SB for the Heath environment, and when
5:  called, prompts you with a series of questions to select either the standard
6:  keyboard or function keys, row/column numbering (on/off), printer type, and
7:  help guides (on/off).
```

Listing 7: Line number macro demo

```
     Installing Spellbinder is relatively    version of SB for the Heath environment, and
easy, as the distribution disc  comes with a when called, prompts you with a series of
COM file already configured for your         questions to select either the standard
hardware in general, as well as a healthy    keyboard or function keys, row/column
assortment of ready to go macros (see listing numbering (on/off), printer type, and help
1). Here, the file HSB.COM is a self patching guides (on/off).
```

Listing 8: 2 column print demo

FORTH-79

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

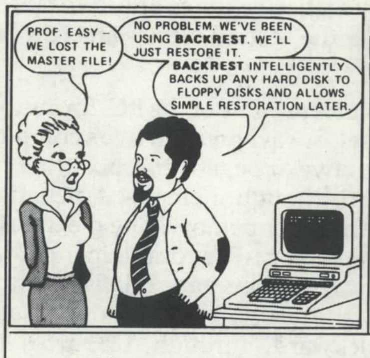
The complete professional software system, that meets ALL provisions of the FORTH-79 Standard (adopted Oct. 1980). Compare the many advanced features of FORTH-79 with the FORTH you are now using, or plan to buy!

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

FORTH-79 V.2	$99.95
ENHANCEMENT PACKAGE FOR V.2: Floating point	$ 49.95
COMBINATION PACKAGE (Base & Floating point)	$139.95

(advantage users add $49.95 for Hi-Res)
(CA. res. add 6% tax; COD & dealer inquiries welcome)

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

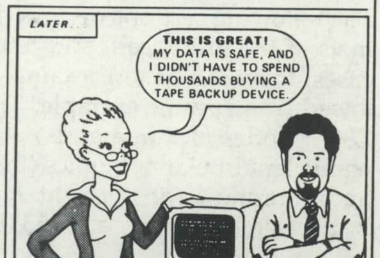

PROF. EASY – WE LOST THE MASTER FILE!
NO PROBLEM. WE'VE BEEN USING BACKREST. WE'LL JUST RESTORE IT.
BACKREST INTELLIGENTLY BACKS UP ANY HARD DISK TO FLOPPY DISKS AND ALLOWS SIMPLE RESTORATION LATER.

BackRest™

Hard Disk Backup, Restore and more!

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

$99.95
Order Toll-Free!
800/431-1953 ext 183
In NY 800 / 942-1935 ext 183
Stok Software Inc.
17 West 17th St.
New York, NY
10011
212 / 243-1444


LATER...
THIS IS GREAT! MY DATA IS SAFE, AND I DIDN'T HAVE TO SPEND THOUSANDS BUYING A TAPE BACKUP DEVICE.

CP/M - MP/M are trademarks of Digital Research

(continued on next page)

Feature

Z80 Tutorial – Jumps, Calls, and Returns – Controlling the Z80's Program Counter

Kim West DeWindt

Now you, and not the Z80, can have ultimate control over a program's progress. (Sounds like a late night TV ad, right?) No longer will you be subject to the constraints of a program counter that chunks forward one address at a time. This month's section discusses the three groups of instructions that let you alter the Z80's program counter (PC), and shows you how to use them to control the Z80's execution of your programs. These PC altering instructions are the jump, call, and return instructions. All of them replace the existing value of the program counter with a new value. Changing the value in the PC alters the flow of a program, directing the processor to a new address.

Jump instructions send the processor on a one way path to a new address. Call instructions save the old value of the PC on the stack, thus remembering where they came from. Return instructions recover the old value of the PC from the stack, providing a way for the processor to return to that old address. Normally, jumps redirect the flow of the main program. Calls and returns are used to get in and out of subroutines.

A direct jump instruction is three bytes long. The first byte holds the opcode, bytes two and three contain the new address. A relative jump instruction requires only two bytes. The first is for the opcode, the second carries an eight bit offset.

This offset, added to the current value of the PC, forms the new PC address. The offset, always given in two's complement notation, can be positive or negative. A positive offset moves the PC forward through memory. A negative offset move the PC back through memory. The maximum values that can be represented by two's complement notation are plus and minus 128. Therefore, relative jumps have a total range of 256 addresses. They can only move the PC, forward or backward, through 128 bytes of memory.

Call instructions are always direct. The instruction is three bytes long, consisting of the call opcode and a new two byte address for the PC.

Return instructions need just one byte for the opcode. They get the new value for the PC from the stack.

The following sections discuss the various forms and formats of the jump, call, and return instructions. In most cases, there is a generic example of each instruction, followed by a specific example. The Zilog mnemonics (the Z80 opcodes and operands) are shown on the left, the equivalent Intel mnemonics (the 8080 opcodes and operands) are shown on the right. Hex numbers are identified by a trailing letter 'H'; e.g. 2533H.

JUMP INSTRUCTIONS

Both the Z80 and the 8080 know how to jump. A typical 8080 jump replaces the old PC value with a new sixteen bit value. The Z80 instruction set not only includes this type of jump (a direct jump), it adds a short form of the instruction (the relative jump). A direct jump replaces the current value of the PC with a sixteen bit value that is contained in the instruction. A relative jump alters the PC with a single byte offset.

Jumps can be unconditional or conditional. An unconditional jump executes every time it is processed. A conditional jump allows the programmer to select internal processor conditions that must be met before the jump is executed.

Unconditional Jumps

The following examples show some of the variations of the jump instruction. The first example is the basic jump:

```
    JP 1234H        JMP 1234H
```

However, when you are using an assembler, use labels in the operand field of the jump instructions. Let the assembler plug in the correct addresses. Using labels, instead of the absolute numerical address, allows you to add or delete lines in a program without having to recalculate all of the addresses for every single jump instruction. For example:

```
    label       opcode      operand
                JP          HERE
                .
                .
                .
    HERE        ADD         A,03
                etc.
```

When the program is assembled, the assembler assigns an address to the label 'HERE', then inserts that address into the operand field of the jump instruction.

Conditional Jumps

The state of any one of the Z80's flags can be tested during a conditional jump. There are a number of logical conditions that can be used to gate the jump (see Table 1). Logical conditions are defined by the state of a flag. When a flag is set to one, a condition is considered to be true. If the flag is set to zero, the condition is false (not true). The Z80 will jump only when the condition selected is true. If the condition is false, the program counter is incremented by two (to skip over the embedded address in the jump instruction) and the program continues on its course.

A conditional jump can test for any one of the following conditions:

cc	Condition	Relevant flag
NZ	Non-Zero	Zero flag is clear (Z = 0)
Z	Zero	Zero flag is set (Z = 1)
NC	Non-Carry	Carry flag is clear (C = 0)
C	Carry	Carry flag is set (C = 1)

PO	Parity Odd	Parity flag is clear (P/V = 0)
PE	Parity Even	Parity Flag is set (P/V = 1)
P	Positive	Sign flag is clear (S = 0)
M	Minus	Sign flag is set (S = 1)

Table 1

It is up to the user to ensure that the proper conditions have been set up BEFORE testing the state of any flag. Zilog mnemonics use one opcode for all jump instructions and include the qualifying condition in the operand. All of the conditional jumps have the following format:

```
JP cc,nn   Jcc nn
```

Here, cc is one of the eight conditions shown in the table above, and nn is a sixteen bit value (the new PC address). This type of construction is different from Intel's set of jump instructions. Intel created a different mnemonic for each conditional jump. For example:

```
JP NZ,1234H     JNZ 1234H
```

This instruction will jump to location 1234H, if the Zero flag is not set (Z = 0).

Relative Jumps

Zilog has expanded on Intel's basic set of jump instructions to include relative jumps. Relative jumps are sometimes referred to as branch instructions. These instructions do not replace the old value of the PC. Instead, they add a one byte value, known as an offset, to the PC. This forces the processor to jump (branch) to another, nearby location. Only one byte is used as an offset, therefore relative jump instructions are limited to a range that is within 128 bytes (plus or minus) of the current location. This is not as limiting as it might at first appear. Many subroutines and loops are short, well within the 128 byte boundary. The judicious use of relative jumps reduces the length of a program.

A relative jump can be unconditional or conditional. However, the number of conditions that can be tested by a relative jump are limited. These conditions are C, NC, Z, and NZ (Carry, Non-Carry, Zero, and Non-Zero). When writing programs, you should use a label to identify the jump destination and let the assembler calculate the proper offset. Here is an example of a relative jump that will always branch to the location 'THERE'.

```
JR THERE          no equivalent
```

The following relative, conditional jump will only branch to 'THERE' when the Carry flag is set (C = 1).

```
JR C,THERE     no equivalent
```

Specialty Jumps

There are two special forms of the jump instruction. The first replaces the old value of the PC with the contents of the HL register pair. Intel fans may recognize this as the Peechl command (PCHL). The format of the Zilog instruction, consistent as ever, uses the JP opcode:

```
JP (HL)          PCHL
```

Remember that the use of the parentheses around the operand HL means that the processor is using the contents of the addressed register pair as sixteen bits of data. In this case that data is the new PC address.

In addition to loading the PC from HL, the Z80 can also load the PC with the contents of one of the index registers (IY or IX). This particular instruction is a little tough for the 8080. It has no index registers:

```
JP (IY)              no equivalent
    or
JP (IX)
```

The other special jump is relative, and it is unique to the Z80. Known as the Decrement - Jump if Not Zero command (DJNZ), it is one of my favorites. The command automatically decrements the contents of the register pair BC. Then, if the value of BC is not zero, the processor executes the relative jump. When the contents of BC reaches zero, the processor stops jumping and the program continues along its merry way. Note that because this is a relative jump, it is only a two byte instruction. Just imagine what this can do for your looping routines. Load the BC register pair with the number of times that you wish to loop through a particular segment of code. At the end of the segment DJNZ is waiting to decrement BC, and, if it is not yet zero, the processor reexecutes the loop.

```
        MOV  BC,10
LOOP    ADC  A,22
          :
        DJNZ  LOOP   no equivalent
```

The assembler will figure out the proper offset value that is required to get the processor back to the address located at the label 'LOOP'.

CALLING

Jumps are fine and dandy if you know that your new destination is where you want to be, permanently. However, should you wish to mark your current location for future returns, you must use the call instructions. Unlike jumps, calls save the current value of the PC in memory (on the stack). Normally, calls are used to enter a subroutine. Then a return instruction, at the end of the subroutine, can be used to restore the old value of the PC, sending the processor back to the address that was saved by the call instruction.

A call can be unconditional (executed every time that it occurs), or it can be conditional. The set of conditions that can be tested by calls is the set of conditions that can be tested by conditional jumps.

Unconditional Calls

Calls are always direct. In other words, the instruction always consists of the call operand (one byte) and the destination address (two bytes). There is no relative addressing form of the call instruction. Again, if you are using an assembler, use labels in the operand field. Let the assembler do the work of calculating the absolute numerical address. Here is an example of a direct call:

```
CALL SUB       ALL SUB
```

...where 'SUB' is the address of the subroutine that the main program is calling.

(continued on next page)

Conditional Calls

When a conditional call begins, the processor tests the state of the appropriate flag to determine if the condition is true. If the condition is true, the call is executed. If the condition is not true, the processor skips the call operation and executes the next instruction.

In the Z80 assembly language, conditional call instructions look like conditional jump instructions. 'CALL' is always in the opcode field. The operand field contains the condition to be tested as well as the destination address. The generic form for a conditional call instruction is:

```
CALL cc,nn      Ccc nn
```

Where cc is one of the eight logical condition codes, and nn is an absolute sixteen bit address. Note that Intel mnemonics create a different mnemonic for each type of conditional call.

Here is a call instruction that tests the state of the carry flag. If the Carry flag is set (C = 1), the processor puts the current value of the PC on the stack, then jumps to 'SUBC':

```
CALL C,SUBC  CC SUBC
```

RETURNING

Return instructions undo the work of the call instructions. Like jumps and calls, they replace the old value of the PC with a new value, sending the processor to a new location. However, unlike jumps and calls, returns do not include the new address as part of the instruction.

Instead, the return instruction gets the new PC address (two bytes) from the top of the stack. Therefore, it is crucial that the appropriate address is residing on the top of the stack when a return instruction is executed. Mismatched calls and returns will most certainly send the Z80 into the ozone. Returns can be unconditional or conditional. Multiple conditional returns are used within a subroutine to provide a number of exits, each dependent upon a different state of the processor.

Unconditional Returns

Normally, an unconditional return is the last instruction in a subroutine. This return opcode is quite simple:

```
RET             RET
```

(Intel and Zilog finally agree on something.)

Conditional Returns

Returns can also be conditional. Conditional returns allow a subroutine to have several different exit points. Frequently, a subroutine will perform a number of different operations (example: reading multiple I/O ports). The actual point of departure will depend on which condition (set by one of the operations) is the first to be true.

Returns use the same set of conditions (see Table 1) that are used by jumps and calls. Generic returns look like this:

```
RET cc          Rcc
```

...where cc is one of the eight logical condition codes. Again, Intel has created eight different mnemonics, one for each conditional return. A return that checks for odd parity (i.e. the Z80 only returns when PV = 0) looks like this:

```
RET PO          RPO
```

Returning from interrupts

Zilog has a unique set of returns that are specifically designed to return to the main program AFTER handling an interrupt. There are two versions of this interrupt return. One returns from Non-Maskable Interrupts (NMIs), the other returns from Maskable Interrupts (MIs).

Non-maskable interrupt returns

When returning from an NMI, use the following opcode:

```
RETN            no equivalent
```

Here is how it works. When the Z80 recognizes an NMI (the NMI pin is pulled low by an external device), it automatically pushes the current value of the PC on the stack, and saves the state of the interrupt flags (IFF1 and IFF2). The Z80 then disables external maskable interrupts, and jumps to address 0066H.

You, the programmer, will have placed an elegant interrupt handling routine at address 0066H. If the last instruction in that interrupt routine is the RETN, (RETurn from Non-maskable interrupt), the Z80 will get the old value of the PC off of the stack and restore the pre-interrupt state of the interrupt flags.

Maskable interrupt returns

The Z80 recognizes a maskable interrupt when the NMI pin goes low while the interrupt flags are enabled (IFF1 and IFF2 = 1). When that occurs, the Z80 disables any future interrupts (IFF1 and IFF2 are cleared) and then saves the current value of the PC on the stack. Now, an RETI instruction at the end of your interrupt handling routine will recover the value of the PC before the interrupt. That return looks like:

```
RETI            no equivalent
```

When returning from a maskable interrupt, the Z80 does not restore the state of the interrupt flags. (This allows the programmer to decide if and when interrupts should be enabled or disabled.)

Wait, let's try that again (Restarting the Z80)

There is yet another way to replace the contents of the PC. The restart instruction will replace the current value of the PC with one of eight preselected addresses. These addresses are hardwired into the processor, permanently etched into the Z80's microcode.

A restart instruction consists of the restart opcode (one byte). Within that opcode is a three bit code that selects one of the eight restart addresses.

In the Z80 assembly language, RST is in the opcode field. The operand field holds the actual destination address. This format is different from Intel's format. Intel assigns a restart number (0 through 7) to each address, then uses that number (instead of the actual address) as the operand. The following table matches the Z80's restart addresses with Intel's restart numbers.

Zilog address (p)	Intel restart number (n)
00H	0
08H	1
10H	2
18H	3
20H	4
28H	5
30H	6
38H	7

To construct a restart instruction, use the following form:

```
RST p          RST n
```

For example, a restart instruction that forces the processor to jump to address 30H looks like:

```
RST 30H        RST 5
```

That about wraps up this section on altering the state of your PC. Next month, I am discussing the leftovers - those miscellaneous groups of instructions that aren't big enough to fill an entire section. This includes the I/O instructions (Input and Output), the bit testing instructions (these are unique to the Z80), and the CPU control instructions.

Any feedback that you may have is always welcome. Send any questions or comments to me, care of Lifelines.

Feature

RECLAIM.Com

Reviewed by Robert P. VanNatta

There are a number of utilities on the market designed to identify and handle disks with defective sectors. RECLAIM happens to be one that is marketed by Lifeboat Associates

The concept behind such a utility is essentially quite simple. RECLAIM and other similar utilities do not actually "fix" your disk. What they do is endeavor to identify portions of the disk that are unreliable (bad sectors) and then "fool' the operating system into avoiding access to these sectors. Since the CPM directory is by definition a road map directing traffic to various sectors, this is often done by means of a directory entry.

Anyone who has used CP/M more than a week has figured out that, if a particular file dies with a "bad sector', it is usually possible to continue to use other files on the disk. Furthermore, by the time that you have used CPM for two weeks you have figured out that if you erase the bad file, you shoot yourself in the foot. The reason is, of course, that as long as you leave that bad file on the disk, the disk sector will remain assigned to that file and thus will not be accessed (as long as you can remember which file is bad and avoid it). If you erase the bad file (which is done by delete flagging the directory), sooner or later the operating system is going to decide the use the bad spot over again and bingo!

A three-week veteran of CPM will figure out that he can protect himself from such disaster by renaming his bad files by some descriptive code such as BAD.FIL, and then by leaving them on the disk forever.

A four-week veteran of CPM will learn another thing. If the disk error is in the directory area, lessons one through three won't work.

A five-week veteran will be smart enough to know that a disk that does any of the things he learned about in the first four weeks should promptly be donated to charity for use as a frisbee.

What does all this have to do with RECLAIM? Well, RECLAIM is about as smart as a two and one half week veteran. RECLAIM reads your disk sector by sector and track by track from beginning to end. Unreadable sectors are identified and are ultimately locked out of use by an almost invisible directory entry. Specifically, the unreadable sectors are collected into a file that is placed in USER area 15 and given R/O SYSTEM attributes.

If you really have the urge to look at the directory entries you can do so by moving STAT.COM to USER 15. Otherwise the directory entries are "invisible".

How Well Does It Work

RECLAIM supports three types of tests. The quickest is a read only test. The others are a non-destructive read-write test and a destructive read-write test.

As far as identifying bad blocks goes RECLAIM works just as well as the "Verify" option that is provided as standard equipment by Lifeboat as a part of the COPY program that comes with CPM versions for the TRS80 Model II (and perhaps other CPM versions as well.)

The design flaw in RECLAIM is that, since it is track oriented, it does not reveal whether or not an identified bad sector is currently within an active file. In fact, it doesn't even bother to tell you where physically on the disk the problem is (the Lifeboat COPY program does). The upshot of this is that, if the problem is outside a file, the sector will be successfully locked out and won't likely bother you until you attempt to run a diskcopy program. However, if the problem is within an assigned file area, you will have the peculiar situation of having two directory entries pointing to a single sector (the one generated by RECLAIM and your original one). For this reason, until the file is identified (by trial and crash) and deleted, you can still access the bad sector by attempting to use the file containing the error.

Conclusions

RECLAIM is user friendly and menu driven. I have not identified any bugs in it. It works exactly like the six page manual says. The first page of the manual is devoted to telling you that Lifeboat makes no warranty with respect to the program, and does not claim that the product is fit "for a particular purpose." If you don't happen to have a bit copy program such as the COPY utility that Lifeboat gives you for free when you purchase CPM for the Model II, this program might have some use.

For my part, however, I have found it almost as convenient as simply renaming the bad files to BAD.FIL.

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**

MasterCard VISA

LIST™

The Software Resource Book For Personal Computer Users

Feature

A Review of Microshell – A Unix Like Utility

Bruce N. Hunter

One of the most pleasant surprises I have had in a long time was reviewing New Generation Systems' CP/M program MicroShell. I am presently completing a book in the C programming language, and I was wondering how to demonstrate to micro users how C and Unix blend and work together. Then I heard of MicroShell. I had been writing a number of C programs in Leor Zolman's version of BDS C (version 1.5) for redirected I/O, and when MicroShell arrived, I jumped on it like a 5 year old at Christmas. There were no disappointments.

The Unix operating system provides command line capabilities that gives almost unlimited input/output redirection coupled with built-in utilities to perform tedious, frequently done programming tasks. In fact, its input/output redirection are legendary by now. Unix uses a shell that is both a command language and a programming language. It is this shell that is the interface to the Unix operating system. What it does is effectively remove the need to program file and device I/O into the applications programs since the Unix operating system will do it for the program by way of the shell. Unix provides a wealth of built-in utilities such as sorting routines, formatting routines, encrypting, text comparison, searching, printing, counting, etc. The shell commands allow these utilities to interact with the program and/or programs by linking them together by way of channels called pipes. The net effect is a system that allows a great deal of work to be accomplished with a minimum of programmer effort.

The shell is now available to CP/M users in the form of MicroShell. What is great about MicroShell is that now all those who enjoy doing creative programming in C can now do so in a more "C"-like environment, which of course is a Unix-like environment. Unix, as we all know, is the property of Bell Laboratories, and was written for DEC PDP series computers operating in a 16/32 bit environment. So, except for the fortunate comparative few lucky enough to have access to a machine using Unix 7, C users did without Unix. However, everyone who has read Kernighan and Ritchie knows that Unix and C go together like Laurel and Hardy. Their development and evolution were interwoven, an operating system and a language that were of, for, and by programmers. And now many of the innovative ideas of Unix can finally live and function in the more familiar, far more user-friendly, and more compatible 8 bit environment of CP/M.

Rick Rump, creator of MicroShell and president of New Generation Systems, was introduced to Unix in 1980 on a PDP-11. Like all of us small system programmers, he was less than enthusiastic about having to learn to live with a very large operating system. Finding that Unix was not only very powerful, but to the surprise of all, easy to learn, and gaining more experience on a DEC VAX, Rick became convinced that what CP/M needed was a "shell". By 1981 he made his dream a reality. Rick created a Unix-like shell to replace CP/M's CCP, the console command processor. Originally written in BDS C, it was rewritten in assembly to keep it down to a mere 8 K bytes.

The CP/M Shell, MicroShell, functions much like the Unix Shell. To fully illustrate some of these features, I'll cover them one at a time.

The big feature, I/O redirection, is accomplished by the traditional Unix characters ">", "<", ">>". The command

```
myprog > progdat
```

will run the program myprog and place its output in progdat. Had the command myprog been given alone, the results would have gone to the screen, since programs written for Unix have only three standard I/O files, stdout (the standard output to the console), stdin (the standard input from the console), and stderr (the standard error routines). The redirection character, ">", has redirected the program output from stdout to the file progdat. On the other hand,

```
myprog > progdat
```

inputs or receives its data from the file progdat. Now, to output the data to a file, all you do is

```
myprog < progdat > outfil
```

This will read the data from the file progdat to the program myprog and redirect its output to the file outfil. There are variations on a theme to be sure. ">>" will append the redirected output to an existent file and ">*" will redirect a file intended for the printer to a file. A command like

```
aprog< + a.dat
```

will run aprog and send its output to the file a.dat while echoing the results to the console. As most C programmers know, C has no provision for a line printer. When C and Unix were created, the I/O was designed to be handled by Unix. If you are running under CP/M, it can be upsetting to find this out, especially for those uninitiated in the writing of drivers! Never mind, because with Microshell you can do this:

```
cprog >$p
```

This sends the results of the program off to the line printer without hesitation. And the wonders haven't even started. The most remarkable feature of the shell is pipes. A pipe is a feature whereby the shell creates a temporary file and sets all the necessary flags to keep a program's output intact, then it feeds the output to the next program as its input, resets the flags and erases the temporary file. The way it appears to the user, the output is "piped" to the input of the next program. The classic example is the creation of a dictionary by piping the following UNIX-like utilities to a filter

```
wrdflter < file.txt | sort | uniq | col >+$p
```

The command line will have the program wrdflter strip off

(continued on next page)

the unneeded white space and remove non-alpha characters from the file file.txt. Then the results will be piped to sort, a system utility to sort ASCII records. The output of sort will then be passed (by pipe) to uniq(ue) to remove any repetitions. In turn, that output will be piped to col, another system utility to put the output in columns. The end result is redirected to the lineprinter and echoed to the screen (+$p). Now, all you programmers pause to think of the accomplishment of this half a line of code. This is just a simple filter consisting of less than a half a page of code coupled to the system utilities by the shell. What it creates is a neat little dictionary of all the words in file.txt, and it is created in minutes. Imagine writing a free standing program in a high level language to do that.

Writing A Filter

Writing the filter itself is a small matter. Here in typical C is a filter to strip all the extraneous characters from any text file (including program source code).

```
/*
                    WFTR.C
          filters words from text files
              (c) Bruce H Hunter 83
*/
#include "bdscio.h"

int c;

main ()
{
      int inspace, inword;
      inspace = FALSE;
      inword = FALSE;
      while ((c = getchar ()) != EOF) /*C uses '!' for NOT*/
      {
            if (c = = '   ' || c = = '\n' || c = = '\t')
            {
                  inword = FALSE;
                  if (!inspace)
                  {
                        c = '\n';
                        putchar(c);
                        inspace = TRUE;
                  }
                  else
                  continue;
            }
            else
            {
                  inword = TRUE;
                  inspace = FALSE;
                  if          (isalpha          (c))
                        putchar(c);
            }
      }
}
```

This is the way it breaks down:

```
while ((c = getchar () ) ! = EOF )
{
      if (c = = ' ' || c = = '\n' || c = = '\t')
```

The file standard input is scanned for input. The keyword is standard input. Since you have to rely on I/O redirection, you must rely on the primitives getchar and putchar for all the I/O. Input is taken a character at a time and checked for the end-of-file mark. When the EOF is seen, the loop will exit. The if statement now looks for the occurence of white space characters.

```
      inword = FALSE;
      if (!inspace)
      {
            c = '\n';
            putchar(c);
            inspace = TRUE;
```

If white space is encountered, the inword flag is set to false, and if the flag inspace is not true, signifying the first entry into the white space, a newline (line feed) is output to demarcate the "record". The flag inspace is now set to true.

```
      else
      continue;
```

If the character pointer is already in white space, you do not want to copy the character, so a null is accomplished by sending execution back to the loop to get the next character without copying this one to the output stream.

```
      else
      {
            inword = TRUE;
            inspace = FALSE;
            if (isalpha (c))
                  putchar(c);
```

The default action is that you are now back in a word, so inword is set to true, inspace is set to false, and if the character is an alpha character, it is put to the output stream.

Again, notice how small the program is and how much it does when used in conjunction with the shell and the system library.

Command Line Flags

In addition to the redirection characters and the pipes, there is a multitude of flags and special characters for the command line. I will touch on them just briefly:

↑	marks a character to be imbedded in the command line to be used as a control character
:	marks a comment in a shell file
;	separates commands on a single line
+	echos redirected I/O to the screen
+	turns on a flag
-	returns char ready (status)
"	quote pairs trap their contents as a single command or argument

\	causes the shell to ignore any special meaning of the next character
$	argument substitution, CPM's familiar $1 etc.
!	invokes a Wordstar-like editor for the command line
$T	Console redirect
$P	Printer redirect
+f	auxiliary file search enable (-f disable)
+g	gobble line feeds during redirected I/O (-g disable)
-l	login disk (+l same thing)
-p	prompt string (you have to read this one)
-s	display shell status
+u	uppercase translation of command line
+v	enable verbose mode (echo)
-v	disable verbose mode
-x	exit the shell
+m	CP/M eof
-m	UNIX eof
-t	transparency character (ignores or recognizes special characters)

The use of the characters is simple. They are used in the command line as required in the order that they are required.

```
-v; -m; wftr < wftr.c | sort | uniq | col >+$p ; +v ; +m
```

This directs the shell to:

1- "Turn off the verbose mode and don't echo the characters" (which would produce double everything)
2- Go to the Unix end-of-file character and use a ↑Z followed by a linefeed
3- Have word filter invoked and read its source code in.
4- Filter it and pass it to sort
5- Pipe its results to unique and its output to column.
6- Output the results to the printer echoing the results to the console
7- Turn on the character echo and CP/M end-of-file flags.

That's a bunch for a one liner!

Shell Files

Should you have any (well founded) objection to typing a command line like the line above more than once, more good news, the shell file. A shell file is very much like a Submit file with some noteworthy differences. The first difference is that shell files work, which is more than can be said for Submit because it is not always reliable. Also, all shell commands can be used. There is no special invocation for a shell file. Just call it by name and the shell will search for it and invoke it. Shell file programming is an art form. It is a language in itself. There are gotos, ifs, returns, wheres, etc., command line argument substitution, variable assignment, input statements (from the console), break characters, output print statements, and that doesn't even scratch shell programming possibilities. Shell files will group programs and system utilites into very complex systems. The only limitations of their capabilities are those of the programmer's imagination and ability to use them.

Command Line Editing

MicroShell also has interactive command line editing. Suppose a typo sneaks in during the typing of a command line for the shell. Just type a "!" and the shell goes into an edit mode. It will retype the command line, inform you that it is indeed in edit mode (with the insert mode either on or off), and you are ready to edit. The editing commands are those of WordMaster or WordStar. You can skip over words, insert, delete, go to the end of the line, the beginning, all that neat WordStar "stuff". Once the command line has been executed, the edit mode automatically goes off. It is an extremely useful, friendly and familiar facility.

All the CP/M features have been retained. PIP, DIR, STAT and the others so familiar to CP/M users, are all there. The shell does not change CP/M, it enhances it. The same familiar control characters like ↑x and ↑u are still there. The difference is that CP/M commands can be grouped on the command line or in a shell file to save all the unpleasantness of retyping (and don't forget the editing).

```
pip a:microshel.rev = b:microshl.rev,wftr.c;type microshel.rev >+$p;ws
```

The line above will create a copy of this article on my B drive, append the source code of the filter program, type it to the printer echoing it to the console, and invoke WordStar when it is done, all this while I'm in the kitchen getting yet another cup of tea.

UNIX-like Tools

There are a number of utilities or tools in Unix that, when coupled with its ability to redirect and pipe, give it enviable power. Most of them are character oriented, since one of the keys to the system's operation is the streaming of "character", as opposed to "binary", data. You have already seen some of them in the above examples such as sort and uniq. MicroShell provides a large library of tools which they call "Microtools". The following is a brief description of their tools.

col

Prints files in multicolumned format

Col will print the input file into a multicolumnar format. The default is a four column output with a page length of 20 lines and the data scanned vertically.

Switches are available for

(-a)	horizontal scanning
(-cn)	number of columns options
(-en)	tab expansion options
(-ln)	page length
(-pn)	blank padding between pages
(-rn)	record mode
(-s)	column delimiters
(-x)	page cut marks

(continued on next page)

cut

cut (and save) a character or field

Cut will print the specified columns or fields, not printingthose not specified.

Switches are available for

(-cs) cutting the characters specified
(-dc) changing the ":" delimiter default
(-fs) field cut
(-r) retain original
(-x) cutting columns or fields *not* specified.

crypt

encrypt or decrypt files

Crypt will do encryption or decryption of the specified input file to a user specified key

deform

deformats a file

Deform removes text formatter control lines (like WordStar's dot commands) from the input file. It can optionally remove inline commands such as imbedded control characters.

If you have ever had to reformat a WordStar file, say for example to feed it to TEX for reformation, the usefulness of this one is self-evident.

Switches are available for

(-ec) change the character recognized for in-line commands from the"\"
(-fc) from the "."
(-i) enable inline command stripping
(-n) retain numerics
(-r) retain control lines, discard text
(-s) enable control character stripping
(-w) filter only words separated by returns into a file
(-x) filter but keep uppercase distinctions

The last two switches perform nearly the same function as the word filter program coded in the middle of this article.

diff

compares text files

Diff will do a line by line comparison of two text files and report all differences found. High order bits are stripped for WordStar compatibility.

echo

echo arguments

Echo will print its arguments to the screen.

The usefulness of this is as a vehicle for printing messages to the screen during shell file execution. It is a nicety for the creation of "warm furry buttons" so seldom seen in Unix execution.

find

find a pattern in a file

Find searches for specific patterns in a file and prints the lines containing them.

Switches are available for

(-a) search across boundaries in the raw mode
(-b) search across boundaries in the text mode
(-i) search white space delimited words
(-n) number input lines
(-p) use the next argument as a search pattern
(-w) verify wildcard expansions
(-x) print everything not containing the pattern.

p

implement a pipe (at CP/M level)

P is used to invoke a Submit file to execute a pipeline.

Switches are

(-o) save the output
(-s) save the pipelne simulation
(-x) do not execute the generated .sub file

paste

horizontal file concatenation

Paste will concatenate two or more files horizontally, e.g. record to record.

Switches are

(-cn) passes each file into a separate column
(-d) delimit with argument
(-p) paste prefix

pr

print files

Pr prints the specified file. The default has a header, pagination, and page numbering. Pr makes up for C's lack if a printer driver in spades.

Switches are

(-bc) break enable
(-cn) multiple copies
(-d) double space
(-en) tab expansion
(-f) form feed
(-g) "display on glass"
(-h) header modification
(-i) secondary header
(-kn) page number reinitialization
(-ln) page length
(-m) reformat header
(-n) five digit line numbers
(-on) left margin
(-p) page pause
(-s) header suppression
(-t) suppress pagination
(-u) map output to upper case
(-x) send printer control codes

rec

reformat record lines

Rec reformats single line multi-fielded records to multi-line records.

The switches are

(-dc) delimiter specifier
(-n) swap names in the first record
(-s) line feed suppression of extra cr/lf's between

records.

sort

sort the file

Sort will sort the input file before passing it to the output file. Default sort is lexicographic by line.

The usefulness of this program speaks for itself. The routine uses a shell sort which is confined to memory and will allow a sort of 5000 lines (memory permitting).

Switches are

(-b)	ignore leading white space
(-dc)	respecify delimiter
(-fn)	Sort on field n
(-m)	sort upper case and lower case together
(-n)	key on numeric strings
(-r)	reverse sort

spl

split the file into manageable pieces

Spl splits the file into sections of (default) 100 lines and renames the fragmented files x1, x2....,xn

Switches are

(-cn)	split by characters
(-ln)	slit by n lines
(-n)	rename output files

str

display printable strings

Str will search a (usually) non-text file for printable strings

Switches are

(-a)	display the strings' address
(-o)	address from the origin of the file and
(-r)	enable raw mode

tee

save the pipeline file

uniq

filter (remove) duplicate lines

Uniq compares adjacent lines and effectively removes duplicates.

Switches are

(-d)	output duplicate lines only
(-n)	mark the count of occurences
(-u)	output unduplicated lines only

wc

count lines, words, and characters

WC will give a statistical count of the number of lines, words, and characters within the specified file

Switches are

(-c)	shar count only
(-f)	include dot commands
(-gc)	command line dot replacement
(-l)	line count only
(-w)	word count only

Microshell vs. Other UNIX-like Utilities

MicroShell has successfully given us the best of two worlds, Unix and CP/M. No compatibilities need be lost with the host operating system CP/M, but most of the advantages of the UNIX shell are now also available and at a minimum of overhead.

MicroShell is not the only utility for CP/M to provide Unix-like features. Unica by Knowlogy also has many Unix-like features, but is available in a Z-80 implementation only. With today's proliferation of 8085/8088 coprocessors, this is an unfortunate oversight. I also understand that MARC, the operating system originally authored by Edwin Ziemba (whose untimely death must have nearly destroyed the project), is near completion by Vortex Technology. MARC is a complete operating system with very Unix-like features. I will be investigating both MARC and UNICA shortly.

At CP/M-83 I was introduced to Carousel MicroTools by Deborah K. Scherrer and the gang at Carousel MicroTools, Inc. Carousel MicroTools (Unicorn Systems) is one of the oldest of the Unix like utilities. Going all the way back to The original Kernighan and Plauger's "Software Tools" (in Ratfor), MicroTools is an enormous package that creates a "virtual operating system," an operating system within a system to allow programs written in and for the system to be ported to any other environment that supports its shell. Like Unix and all the Unix-like systems, the command line interpreter provides the vehicle to bring the system together into one powerful cohesive entity. The library of functions available are the largest I have encountered yet for a set of micro tools, and they can easily be the entire subject of another article.

As almost a post script to this article, at CP/M-83 Digital Research announced their very own, internally developed C. Written in C, it is the full Unix 7 C implementation. The package will not be available until spring and will be offered, at least initially, in the 8086 (8088) version. It fully supports I/O redirection and will have an independent preprocessor for it. Again, we move closer to C and its environment. Back to Microshell.

The documentation for Microshell is well written and organized. It has a nice format, and once I picked it up, I found it impossible to put down until it was completed (a statement rarely made about manuals). The only bug I found was an occasional clipping of the last letter of words run through a series of pipes. Don Graff of MicroTools (associated with New Generation Systems, not to be confused with Carousel MicroTools, Inc., a different company) is writing a merge routine to sort/merge previously sorted files. This will be a welcome addition. The only other idiosyncracy I noticed is that during file read/writes there is a great deal of head activity indicating unusually small file buffers, no doubt to allow operation in small memory areas. I have been told that this is going to be improved upon in a future release.

Microshell and MicroTools are available from New Generation Systems, 2153 Golfcourse Drive, Reston, Virginia 22091. Phone: (703) 476-9143

It is not often that one finds a piece of software that fulfills a need as well as Microshell has, and it does it so well. It is a welcome addition to the field of microcomputing. ■

Feature

A Review of Wordix, A Text Formatter

Ron Watson

Why would anyone use two programs when one would do? If one program can be used to edit text and prepare the final document, why would anyone ever consider using two separate programs, one to prepare a file of data and another to format that file into a completed document?

With a wide selection of one-step word processing programs available for the IBM PC, why should anyone consider a two-step process like Wordix? The whole idea of using a text editor to input the data, along with all those formatting commands, and then running another program just to translate that into a final document seemed archaic to me. That's the way it would have been done in the old days when batch processing was the cat's meow.

After all, WordStar will respond to over 130 commands. That's very nice, lots of capability. Anyone who uses it very often will soon learn the commands. But what of those who only need to prepare a few documents a week? They are not going to spend enough time working with the program to become facile in the use of so many commands. (I'm not picking on WordStar, any full function word processor contains a rich command language), and if they also want to prepare other text files, like programs, they will probably want a different editor for that. Sounds like, for the present anyway, we might be stuck with two programs: one for word processing documents that are destined to be printed, and another to edit program files.

Now, if you chose Wordix, or a similar product, you could use just one program, any text editor, to enter the text, thereby eliminating the need to learn two methods of data entry. Whether you put the format commands into the program interactively, as with WordStar, or into the text file, as with Wordix, there is no way to avoid the need to remember them, and the problem of familiarity remains the same. Or does it?

Text formatters usually have a macro facility that allows you to redefine commands or groups of commands so that they may be invoked with terminology meaningful to you. In the case of Wordix, and others, once these macros are defined and tested they can be stored in a disk file and used over and over. Maybe there is some relief for the occasional user. If the macros were defined to suit a particular application, with names that were meaningful to the user, perhaps the effort to remember the commands could be reduced.

Any good full-screen text editor with a print function could be used for informal notes, short letters and memos. Combine the text editor with this product and a set of macros defined to handle the specific needs of the user for more formal presentation and you've got both power and ease of use.

The occasional user is not the only one who might benefit from such a product. A program that can concentrate on the preparation of documents, one that needn't be concerned with text entry, can provide some extra features found in few all-in-one word processors available today: features like automatic table-of-contents and footnote positioning, for instance. Large documents should be easier to handle, too.

The program comes in a standard size binder and includes one program diskette and about 200 pages of documentation. The manual is divided into a tutorial section and a reference section, and although it is well written, it could use an index and more examples.

Several printer personality files are provided, covering most popular letter quality printers, but the only dot-matrix printer included is the Epson. The first thing I had to do was learn how to build a personality file for my printer, an Okidata Microline 84. The list of options seems quite complete, but there is an anomaly in the way the combinations of options interact. The list allows for a printer that has underline and autobold commands, but does not always use them. Apparently, the program works in two modes: "single pass print", when all the enhancements can be done with one pass of the print mechanism; and "two pass print", when it is necessary to make two passes with the print mechanism because the printer does not have a command to implement a needed feature. However, the mode is not determined by whether or not the printer has the enhancement command, but by whether or not it can backspace. There is a certain logic to this, but while my printer has enhancement commands, even more commands than they anticipated, it will not backspace unless it is in "incremental mode", at which time the enhancement commands won't work. When I told Wordix the truth, *i.e.*, no backspace, it implemented all the enhancements using two passes of the print head over the same line. This worked o.k., except in what Okidata calls "near letter quality" mode, wherein the printer automatically makes two passes over the line, slightly offsetting the print for each pass.

This turned the output into garbage. So, I lied - I told the program the printer would backspace. This brought good results except when I tried to use the Wordix overstrike feature and it tried to backspace. I had to give up "near letter quality" or overstrike. There was also no provision for the Okidata's subscript/superscript commands: Wordix implements these by positioning the carriage in half line increments.

The description of printer customization in the manual is the weakest section, and it's still not bad. The whole process is made fairly painless by the thoughtful inclusion of a printer test file on the program diskette. I tried the program on several printers, and was able to get good results from all of them. Despite the problems I had with my printer, Emerging Technology, Inc. has anticipated the features (or lack thereof) of most letter quality and dot-matrix printers.

The tutorial is written in a casual style that leads a new user gently through the basics. After going over it, I was able to prepare a simple report with page headings, footings, and print enhancements, and had even learned how to set up simple macros. I had a report to prepare, so I used my newfound knowledge to turn what would have been a handwritten document into a very professional looking piece of work with numbered pages, right justified text and cleverly placed underlines. The recipient was so impressed he paid very little attention to the content, which wasn't that important anyway.

Emboldened by success, I plunged into the reference section of the manual, seeking the really exotic stuff they had promised at the end of the tutorial: things like automatically numbered lists and sections, footnotes, tables-of-contents and other advanced formatting features.

The reference section starts with a description of how to invoke the program from DOS. There are many options that can be put on the DOS command line, and most are quite useful. One can vary the overall page offset and the printer personality file to be used, and can specify values for user-defined variables to control the output or to appear in the text. The command line can get quite long and a regular user would want to set up .BAT files to ease the pain.

The reference section then goes on to describe the various types of commands available, alternative ways to get data into the program, the composition rules used to create an output line, how page layout is done, hyphenation rules and footnotes. This part of the manual goes a little fast and is not as well organized as the tutorial, but it does cram a lot of information into 25 small pages.

It is not necessary to spend much time in the reference section to discover that what you have here is another programming language, complete with assignment statements, "IF"s and "GOTO"s. There are also more than thirty format commands, another thirteen or so to implement the macro features, and five or six to control input and output files. There are an additional eight commands that are included with the package as pre-defined macros whose functions are described in the reference. Now that may seem somewhat overwhelming to you; it certainly did to me, until I began to dig in to it.

The format commands are, with a few exceptions, very straightforward. A two-character mnemonic preceded by a period to identify it and placed in the first position of a line does the job. The command names are well chosen and easy to remember. The descriptions in the documentation are concise and complete. Likewise, the I/O commands are implemented in a straightforward manner with commands like "read" and "write".

The macro facility is more complicated. If you have ever worked with an assembly language macro assembler, you will be quite at home here. While not as powerful as most assemblers, it allows parametric substitution, a simple "IF" statement, loops and some function calls. A macro may call other macros including itself, and calls may be nested fifty levels deep. Macro names and variable names can be up to nine characters long and the names are case-sensitive so that "PAGE" is distinct from "page". Emerging Tech adopted a convention of using upper case values for macros to aid in distinguishing them from commands. There are more than twenty pre-defined variables that are maintained automatically by the program. These contain things like the current date, page number and line number on the page.

Several useful macros are provided with the program and some of these serve as examples in the documentation to describe how the macro facility might be used. The descriptions are easy to follow, showing how the development process might proceed, but there are not as many examples as I would like.

One set of macros provided with the program and described in the documentation can be used to define a list of items to be numbered or labeled as it is printed. I started to experiment with macros by enhancing these to vary the location of the item label depending on the length of the label. If the label would fit, it would show on the same line as the first line of the item description, otherwise the label would be on a line by itself. This presented no real difficulty, so I tried a page heading macro that would allow a line in the page heading to vary so it could be used as a section description. The macro to do this was easy enough, but I did turn up an error that crashes the program if a heading command is mis-coded.

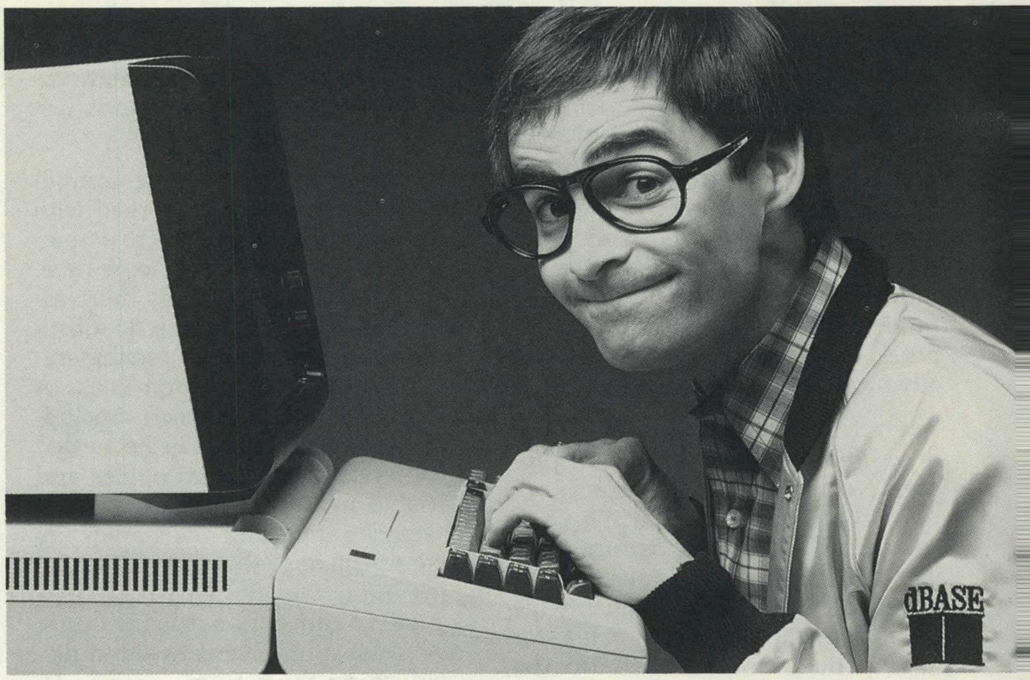
The program was generally very helpful when an error was made, telling the nature of the error and the line where it occurred. The one exception I found is mentioned above. My only complaint about the macro development process is the time needed to switch from text editor to Wordix to text editor in order to input the macro, test the results, and then correct the macro. Wordix is a huge program (over 90k of disk space) and it takes a while to load. I put Wordix and a text editor in a virtual disk and that, of course, speeded things up.

Macros may be used to divert processed output text into a named buffer. Output thus diverted can be recalled into the output file at any time by using the buffer name as a macro. This very powerful feature can be used, for instance, to keep a part of the output from being split across two pages, or to assure that an empty area is left in a specific place.

The only serious lack of capability I found is the control the user has of the program's composition rules. For instance, there is no way to request proportional spacing on right justified output, and the space between words is always reduced to one blank, even following punctuation such as periods and semicolons. This actually sounds worse than it looks, but I would have liked to see an option to control the spacing between sentences. A incremental spacing is a fairly common option on high-ticket, letter quality printers, they really should have provided the capability to use it. These shortcomings preclude using the program to prepare camera-ready output for proportionally spaced, right justified text.

In summary, I believe the program has a very good balance between capability and ease-of-use. It is certainly powerful enough for most text processing needs, and once macros have been prepared for a specific application, it can be very easy to use. I would also recommend the program for consideration because of the overall high quality of the documentation and program reliability.

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

Feature

Sliding into BDOS (Part III)

Michael Karas

The time has arrived to complete the third and final part of this series on the operation of the CP/M BDOS as viewed from the assembly language programmer's perspective. Presently we will build upon the extensive treatment of sequential files presented in Part II of the series to provide a basis for understanding the CP/M 2.2 random file I/O capability. Please note that functions of the BDOS presented here are specific to CP/M Versions 2.2 and 3.0. Older CP/M systems using Version 1.4 do not directly support random access file I/O and as such are not compatible with the programming examples presented below.

Why Random File I/O Anyway?

In the beginning of the CP/M era, sometime around the release of Version 1.3 by Digital Research, small inexpensive single-user micro processor systems were typically used for simple-minded data processing applications. Most computing operations were linear with respect to the data handling by the CPU. Data entered from paper tape, cassette, card readers, or human entry from a keyboard tended to be limited to sequential processing from start to finish. The usage of such data by the computer in data analysis, program compilation, or logging applications was also largely sequential. Finally the data output operations based upon the needs of hard copy, backup, and transmission from micro to micro were relegated to sequential processing applications.

Anticipated applications of micro type computer hardware by operating system designers, at that time, seemed to dictate that the disk file structures of the operating systems should be sequential in nature. This was true for the earliest releases of CP/M and Intel's ISIS II operating system. Other simple floppy disk operating systems like PERTEC's FDOS and MITS' Disk Extended Basic operating systems were also strictly sequential in the treatment of disk file allocation and storage. However, these two systems permitted random record I/O within the bounds of an already existing file provided the space to store the records was previously pre-allocated as contiguous disk space in the file structure. The process of random I/O was then easy as a relative offset between the beginning record number for the file and the offset desired within the file.

As the micro processor applications market opened up in the late 1970's it seemed that new uses for computers were being found weekly. It has gotton to the point that micro processor computer users have a large array of very sophisticated software packages to choose from and utilize in their business and hobby activities. The main thing that can be pointed out about many of these packages is that the processes they perform are hardly linear with respect to the handling of data. Interactive programs like word processors, data base managers, spelling checkers, and spread sheet analysis programs may very well need to be able to store or access data to/from a disk file in a manner that cannot be handled in the old sequential manner. The sequential philosophy generally limited file update to appending to the end of the file, and read access to a particular record had to read N-1 records from the beginning of the file prior to being able to read record N.

Random access file I/O within an operating system anticipates the requirements of non-sequential I/O by permitting access to various records directly. Any record that was previously written may be read upon demand. Likewise the user/programmer may write any record desired. The Digital Research CP/M operating system supports this type of I/O in a powerful yet elegantly simple manner through a set of four BDOS system functions. These calls allow random access disk files to be implemented within the standard CP/M compatible file structure.

Random File Structure Under CP/M 2.2

The structure of random files under the CP/M operating system is much the same as that for sequential files. Part II of this series (**Lifelines,** January 1982) described and illustrated the sequential structure in detail. The reader will recall that CP/M treats disk data in fixed records of 128 bytes. These records are collected together into "groups" that are stored on the disk as an allocated group. The disk space reserved for a given file, in its directory entry, is always marked, identified, and allocated in even multiples of the 'allocation group size'.

I previously mentioned two older operating systems that supported random file I/O within the confines of a pre-allocated file. This system required that all of the space for an "N" record file be reserved as contiguous disk space even if the file only contained two records (-0 and -N). Making a random access file bigger than the pre-allocated size was virtually impossible. The CP/M Ver 2.2 random file access system has overcome the problems described above. A random file under CP/M contains only the number of allocated groups required to hold the stored records. The holes between the defined records do not consume unused disk space.

If a file under CP/M is created with only random record 0 of the file written then that file contains 128 bytes of real data and consumes one allocation group of disk space. The allocation group consumed also may contain other adjacent random records to fill out the size of the group. For instance, on single density 8″ disks with a 1024 byte allocation group size, a one record (-0) file would be able to be written with additional record numbers 1 to 7 within the same allocation group. Likewise if a single record file was created with only record number 9 written, that file would consume only one allocation group of disk space. Additional record numbers 8, and 10 to 15 could then be written without requiring additional disk space.

Random File I/O System Calls

Let us next investigate the five BDOS system calls that

(continued on next page)

CP/M supports for random I/O within files. The chart of Figure 1 on page 34 details the look of a random access file control block. Note that the file control block contains three bytes at the end that are used to store the random record number that will currently be accessed. The random access system calls all utilize this field to determine the portion of the file to access at read/write time.

A CP/M random file may contain up to 64K records of 128 bytes numbered from 0 to 65535. Two bytes of the file control block hold this record number, r0 as the low byte and r1 as the high byte. This provides accessability to records up to a maximum file size of 8 megabytes. The r2 byte of the file control block is not used except as the overflow or carry out of the r1 byte. If byte r2 ever contains a value that is non-zero the record number is beyond the end of the 8 megabyte limit for the file.

To access a random file, it must first be opened in the normal manner with the "open" BDOS function call. In the case of creating a new random file, the make file BDOS call is sufficient in that the results of the make operation are equivalent to the open function on a zero length file.

Read Random Record: Function 33.

This system call is made with the (DE) register pair pointing to a 36 byte file control block. Bytes r0-r2 are set up with the random record to read. The BDOS then fetches the addressed record from the file and places it in the caller's record buffer pointed to by the last set buffer address function call. The r0-r2 fields of the file control block are not changed as a result of the random read function such that a subsequent random read operation would read the same record. The random read function may return a number of error codes as described below:

Error Code 00 - The random read function worked without error and the user buffer contains the desired data.

Error Code 01 - The random read operation addresses a record that is contained in a disk allocation group not allocated to the file. This means that the group field number slot of the appropriate extent of the file that should contain the record is equal to 0.

Error Code 03 - The random read operation just requested required that a different extent descriptor directory entry had to be open for the impending operation, however prior to opening the new extent the current extent could not be closed due to disk read/only status or a disk change.

Error Code 04 - The random read operation just requested required access to an extent of the file that does not exist on the disk.

Error Code 06 - The random read operation just requested required access to a record number beyond the bounds of the disk drive, *i.e.*, the disk drive is less than 8 megabytes and the record requested is within an allocation group beyond the end of the disk.

Write Random Record: Function 34.

This system call is made with the (DE) register pair pointing to a 36 byte file control block. Bytes r0-r2 are set up with the random record to write. The BDOS then moves the data in the caller's record buffer pointed to by the last set buffer address function call to the addressed record in the file. The r0-r2 fields of the file control block are not changed as a result of the random write function such that a subsequent random write operation would write the same record. The random write function may return a number of error codes as described below:

Error Code 00 - The random write function worked without error and the user buffer contains the desired data.

Error Code 03 - The random write operation just requested required that a different extent descriptor directory entry had to be open for the impending operation, however prior to opening the new extent the current extent could not be closed due to disk read/only status or a disk change.

Error Code 05 - The random write operation just requested required access to an extent of the file that does not exist on the disk. In the process of creating the new extent the disk directory was found to be full.

Error Code 06 - The random write operation just requested required access to a record number beyond the bounds of the disk drive, *i.e.*, the disk drive is less than 8 megabytes and the record requested is within an allocation group beyond the end of the disk.

WRITE RANDOM RECORD WITH ZERO FILL: Function 40.

This system call is made with the (DE) register pair pointing to a 36 byte file control block. Bytes r0-r2 are set up with the random record to write. The BDOS then moves the data in the caller's record buffer, pointed to by the last set buffer address function call, to the addressed record in the file. The r0-r2 fields of the file control block are not changed as a result of the random write function such that a subsequent random file operation would access the same record. If the random write operation caused a new allocation group to be allocated to the file the other records of the same block are filled with zeros. The random write with zero fill function may return a number of error codes identical to those described for function number 34 above.

COMPUTE FILE SIZE: Function 35

This system call determines the number of 128 byte records in a file and sets the number of records into the r0 and r1 bytes of the 36 byte file control block addressed by the (DE) register pair. The returned size is a virtual size in that if the file was created by random write operations and the file contains 'holes" the file size function does not take the holes into account. Another way of looking at this is to think of this function as returning a record number that is one greater than the maximum record number currently in the file. If the file had no 'holes" or it had been written in the conventional sequential fashion, then the file size reported by this function is the real file size. This function provides a convenient means of positioning a file at its end so that subsequent sequential or random update may be performed.

SET RANDOM RECORD: Function 36:

The (DE) register pair is set to point to a 36 byte file control block that has previously been used to reference a file in the sequential mode. Upon reference with this system call the r0 to r2 fields are filled in with the random record number that corresponds to the current file position, *ie.* the BDOS simply computes the real current record number as follows:

The current extent number is multiplied by 128, the number of records per extent, and to this product is added the numerical value of the CR field, current record in this extent. The final result is placed into the r0-r2 fields of the FCB.

Looking At Some Examples

The following simple assembly language program is designed to write record numbers 0 and 143 into a file on the disk. The write random function is used to write the first record with all A's and the second record, – 143, with all B's.

```
;
;
;RANDOM RECORD I/O DEMONSTRATION FOR CP/M 2.2
;
; THIS FIRST LEVEL DEMONSTRATION IS DESIGNED TO
; SHOW HOW TO INITIALLY SET UP A FILE TO BE A RANDOM FILE
; AND TO WRITE TWO RECORDS INTO THE FILE SUCH THAT THE
; FIRST RECORD (RECORD NUMBER 0) AND THE SEVENTEENTH
; RECORD OF THE SECOND EXTENT (RECORD NUMBER 143) BOTH
; CONTAIN DATA. THE PURPOSE IS TO DEMONSTRATE THE
; RESULTING DISK DIRECTORY ENTRIES THAT RESULT FROM
; AN INCOMPLETE FILE. THIS DEMO PROGRAM DOES NO RANDOM
; WRITE ERROR CHECKING.
;
; SYSTEM LEVEL INTERFACE EQUATES
;
BDOS    EQU  0005H          ;SYSTEM INTERFACE VECTOR
MAKE    EQU  22             ;MAKE NEW FILE FUNCTION
SBADDR  EQU  26             ;SET DISK BUFFER ADDR
OPEN    EQU  15             ;OPEN FILE FUNCTION
CLOSE   EQU  16             ;FILE CLOSE FUNCTION
DELETE  EQU  19             ;DELETE FILE FUNCTION
RRAND   EQU  33             ;READ RANDOM FUNCTION
WRAND   EQU  34             ;WRITE RANDOM FUNCTION
WRANDF  EQU  40             ;WRITE RANDOM WITH 00 FILL
;
;
        ORG  0100H          ;START OF A PROGRAM
;       XRA  A              ;ZERO BYTES OF THE FCB
        STA  EXT            ;EXTENT FIELD
        STA  CR             ;CURRENT RECORD COUNT
        STA  RR+2           ;AND THE R2 FIELD
        LXI  H,0000H        ;ALSO ZERO RANDOM RECORD
                            ;FIELED
        SHLD RR
;
        LXI  D,BUFFER       ;SET DISK BUFFER ADDRESS
        MVI  C,SBADDR
        CALL BDOS
;
        LXI  D,RANDFCB ;POINT AT OUR FCB
        MVI  C,DELETE       ;ERASE TEST FILE IF IT ALREADY
                            ;EXISTS
        CALL BDOS
;
        LXI  D,RANDFCB ;MAKE A NEW FILE FOR TEST
        MVI  C,MAKE
        CALL BDOS
;
        MVI  A,'A'          ;FILL FIRST RECORD WITH A'S
        CALL FILL           ;GO FILL
        LXI  H,0000H        ;SET RECORD NUMBER TO WRITE A'S
                            ;INTO
        SHLD RR
        LXI  D,RANDFCB ;WRITE RECORD OF A'S
        MVI  C,WRAND        ;NORMAL WRITE RANDOM
                            ;FUNCTION
        CALL BDOS
;
        MVI  A,'B'          ;FILL NEXT RECORD WITH B'S
        CALL FILL           ;GO FILL
        LXI  H,143          ;SET RECORD NUMBER TO WRITE B'S
                            ;INTO
        SHLD RR
        LXI  D,RANDFCB ;WRITE RECORD OF B'S
        MVI  C,WRAND        ;NORMAL WRITE RANDOM
                            ;FUNCTION
        CALL BDOS
;
        LXI  D,RANDFCB ;CLOSE JUST WRITTEN FILE
        MVI  C,CLOSE
        CALL BDOS
;
;
        RET                 ;BACK TO CCP BY IMMEDIATE RETURN
;
;
; SUBROUTINE TO FILL BUFFER WITH A PATTERN
;
; ENTRY WITH (A) CONTAINING BYTE TO FILL BUFFER WITH
;
FILL:   LXI  H,BUFFER       ;POINT AT BUFFER FOR FILL
        MVI  B,128          ;FILL BYTE COUNTER
FILLP:  MOV  M,A            ;PUT A BYTE INTO BUFFER
        INX  H              ;BUMP POINTER
        DCR  B              ;DECREMRNT BYTE COUNT
        JNZ  FILLP          ;CONTINUE TILL BUFFER FULL
        RET
;
;
;RANDOM FILE TEST DATA AREA
;
RANDFCB: DB  00             ;USE CURRENT LOGGED DRIVE FOR
                            ;TEST
        DB   'RANDFILE'     ;NAME OF FILE TO PLAY WITH
        DB   'TST'          ;...AND THE EXTENSION NAME
EXT:    DB   00,00,00,00    ;EXTENT, S1, S2, AND FCBSZ BYTES
        DS   16             ;STORAGE FOR THE ALLOCATION
                            ;NUMBER
CR:     DS   1              ;CURRENT RECORD BYTE
RR:     DS   2              ;RANDOM RECORD NUMBER (R0,R1)
        DS   1              ;RANDOM RECORD OVERFLOW BYTE
                            ;(R2)
;
;
;RANDOM DISK I/O DATA BUFFER
;
BUFFER: DS   128            ;ONE RECORD BUFFER
;
END
```

The above program was assembled and caused to run on an empty single density disk in the default disk drive. The following display shows how the directory upon the disk

(continued on next page)

looked after running the program. Notice that the file only consumes two allocated groups. Due to the fact that this was a single density disk with 1024 byte allocation groups of 8 records each, then if record number 8 was subsequently written the directory entries would change to include an allocation block number in the second group number slot of the first extent of the file.

G=00:00, T=2, S=1, PS=1

```
00 0052414E 4446494C 45545354 00000001 *.RANDFILETST....*
10 02000000 00000000 00000000 00000000 *................*
20 0052414E 4446494C 45545354 01000010 *.RANDFILETST....*
30 00030000 00000000 00000000 00000000 *................*
40 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 *eeeeeeeeeeeeeeee*
50 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 *eeeeeeeeeeeeeeee*
60 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 *eeeeeeeeeeeeeeee*
70 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 *eeeeeeeeeeeeeeee*
```

The following two sector displays off the single density disk show the A's and B's written by the program above. All other sectors in the group numbers 02 and 03 were empty, *i.e.*, contained whatever data used to be there. This brings up the subject of the write random with zero fill function. A small segment of the first demonstration program was changed to cause the second write operation to be done with zero fill. The changed portion of the program is shown below:

```
        LXI   D,RANDFCB ;WRITE RECORD OF A'S
        MVI   C,WRAND   ;NORMAL WRITE RANDOM
                        ;FUNCTION
        CALL  BDOS
;
        MVI   A,'B'     ;FILL NEXT RECORD WITH B'S
        CALL  FILL      ;GO FILL
        LXI   H,143     ;SET RECORD NUMBER TO WRITE B'S
                        ;INTO
        SHLD  RR
        LXI   D,RANDFCB ;WRITE RECORD OF B'S
        MVI   C,WRANDF  ;WRITE RANDOM ZERO FILL
                        ;FUNCTION
        CALL  BDOS
;
        LXI   D,RANDFCB ;CLOSE JUST WRITTEN FILE
```

G=02:00, T=2, S=17, PS=20

```
00 41414141  41414141  41414141  41414141   *AAAAAAAAAAAAAAAA*
10 41414141  41414141  41414141  41414141   *AAAAAAAAAAAAAAAA*
20 41414141  41414141  41414141  41414141   *AAAAAAAAAAAAAAAA*
30 41414141  41414141  41414141  41414141   *AAAAAAAAAAAAAAAA*
40 41414141  41414141  41414141  41414141   *AAAAAAAAAAAAAAAA*
50 41414141  41414141  41414141  41414141   *AAAAAAAAAAAAAAAA*
60 41414141  41414141  41414141  41414141   *AAAAAAAAAAAAAAAA*
70 41414141  41414141  41414141  41414141   *AAAAAAAAAAAAAAAA*
```

G=03:07, T=3, S=6, PS=5

```
00 42424242 42424242 42424242 42424242 *BBBBBBBBBBBBBBBB*
10 42424242 42424242 42424242 42424242 *BBBBBBBBBBBBBBBB*
20 42424242 42424242 42424242 42424242 *BBBBBBBBBBBBBBBB*
30 42424242 42424242 42424242 42424242 *BBBBBBBBBBBBBBBB*
40 42424242 42424242 42424242 42424242 *BBBBBBBBBBBBBBBB*
50 42424242 42424242 42424242 42424242 *BBBBBBBBBBBBBBBB*
60 42424242 42424242 42424242 42424242 *BBBBBBBBBBBBBBBB*
70 42424242 42424242 42424242 42424242 *BBBBBBBBBBBBBBBB*
```

Note from the directory display below that there is no change in the appearance of the entries from the first example. This time the only thing that changed was the data in allocation group 3. Due to the second write this allocation group contains a sector of B's at GROUP=03:07 with the other seven sectors of the group now containing zeroes from the zero fill operation. The function of zero fill is to leave a clean slate on record numbers subsequently read from the same allocation block. The BDOS is capable of reporting unwritten record information for records that correspond to group number slots in the directory entries that contain a '00' byte indicating unallocated. However, once a group is allocated for one record the BDOS cannot determine if other sectors of that group have been written or not. Thus an error function may be issued when creating a random access file for the first time. The programmer may then use a record of 128 zeroes to indicate that the record is not used, as opposed to accidentally mistaking the garbage data from un-initialized sectors written without zero fill as real data.

G=00:00, T=2, S=1, PS=1

```
00 0052414E 4446494C 45545354 00000001 *.RANDFILETST....*
10 02000000 00000000 00000000 00000000 *................*
20 0052414E 4446494C 45545354 01000010 *.RANDFILETST....*
30 00030000 00000000 00000000 00000000 *................*
40 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 *eeeeeeeeeeeeeeee*
50 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 *eeeeeeeeeeeeeeee*
60 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 *eeeeeeeeeeeeeeee*
70 E5E5E5E5 E5E5E5E5 E5E5E5E5 E5E5E5E5 *eeeeeeeeeeeeeeee*
```

The next example program is included here to show a clever means of implementing arbitrary record selection I/O within a file without resorting to random file I/O. The intent is not to indicate that the following scheme is the preferred method. The program below was developed with the CP/M Ver 1.4 operating system in mind. However the algorithm works fine with CP/M 2.2 as well. The technique used to play with random records by using sequential read and write operations is to manipulate the 'cr" field of a standard 33 byte file control block. The "cr" byte is the only means that the BDOS uses to indicate the next record to access. The programmer may change this byte value to force the BDOS to go to any record within the current extent.

If the first extent of a file is opened, the group allocation values for that extent lie in the file control block. If the technique of performing your own random I/O is done, the code must access record numbers not to excede 07fh without first closing the current extent and opening the next. This can be done with either the conventional open and close operations, or the programmer, when done working with the current extent, may open the next automatically by performing a dummy read of record 080H of the current extent. The programming example below uses the "roll your own" technique but does not anticipate a file size greater than 16K (one extent size).

The program below is a skeleton structure of a .COM file serialization procedure. The idea is to insert a six byte serial number string into the target file PROG.COM on drive B:. The serial number is inserted into the file at the places specified by the labels in the table at the start of the listing. These values are stripped out of the symbol table that is generated at the assembly of the PROG.ASM file. If the assembler does not generate a symbol table then the label values may be pulled off the .PRN listing output. The insert points are places within the 'to be serialized" program where the programmer has determined that he would like to place the serial number string. Within the file itself, the labels point to the place where the string is to be inserted with respect to run time load address. The real file offset is 0100H bytes less. In addition, the scheme does not insert all six bytes of the program serial number at each location. The byte at each label address minus one contains a value between 1 and 6 of the number of serial number bytes that should actually be inserted at serialization time.

The list of label values in the program below is used to build, at assembly time, a table of record numbers where the specific serial number strings are to be inserted. This table is then used to fill in the "cr" byte of the file control block as each serial number is to be inserted. The table also contains the byte offset within the record where the insert point is to start. As each serial number is to be inserted the appropriate record is read, the number is inserted (with length specified by the value from the file record just accessed), and the record is written back to the disk. Sequential read and write operations are used for both operations. Logic within the code listing below also provides for the occurrance that the serial number string may cross the end of the first record and flow into the next record. In this case the first is rewritten followed by reading of the next with the remainder of the insert proceeding from the beginning of the second record.

Please note that the program example is given as a skeleton only and the serial number entry process, increment process, and the disk I/O error exit points are left for the reader/programmer to fill in with code of his own choosing.

```
;
;
;PROGRAM SERIAL NUMBER INSERTION EQUATES
; EACH ADDRESS IS A VALUE INSIDE OF THE "PROG.COM"
; FILE THAT IS THE PLACE TO PUT THE SERIAL NUMBER.
;
SERA     EQU  0132
SERB     EQU  01E9
SERC     EQU  0278
SERD     EQU  039A
SERE     EQU  06FF
SERF     EQU  0732
SERG     EQU  0BBC
SERH     EQU  0C08
;
;
;CP/M BDOS SYSTEM CALLS FUNCTION NUMBERS
;
BOOT     EQU  0000H        ;REBOOT LOCATION ENTRY POINT
BDOS     EQU  0005H        ;BDOS FUNCTION ENTRY POINT
RESET    EQU  13           ;RESET DISK SYSTEM
OPEN     EQU  15           ;OPEN FILE FUNCTION
CLOSE    EQU  16           ;CLOSE FILE FUNCTION
DMAADR   EQU  26           ;SET DATA BUFFER ADDRESS
READ     EQU  20           ;READ SEQUENTIAL
WRITE    EQU  21           ;WRITE SEQUENTIAL
;
;
;DEFINE BASE EXECUTION AREA FOR THIS PROGRAM
;
START    EQU  0100H
;
;
         ORG  START          ;BASE OF EXECUTION AREA
;
;
;START UP HERE WITH PROGRAM INITIALIZATION AND
;DEFINE PROCEDURE TO FETCH IN SERIAL NUMBER TO INSERT INTO
;THE FILE
;
SERASK:
;
;ENTER APPROPRIATE CODE HERE TO PUT A SIX BYTE SERIAL
;NUMBER INTO VARIABLE "SERSTR"
;
;
;
;SERIAL NUMBER INSERT POINT PROCESSING
;
;
SERCOPY:
         MVI  C,RESET        ;RESET DISK SYSTEM UPON INSERT
         CALL
         BDOS
         LXI  D,PROGFCB ;SET TO OPEN THE PROG.COM FILE
         MVI  C,OPEN
         CALL BDOS
         INR  A              ;CHECK IF OPEN ERROR
         JNZ  SERCP1         ;OPEN SO GO START WRITE
;
;PRINT ERROR MESSAGE HERE AS TO INDICATE THAT THE FILE
;'PROG.COM" IS NOT PRESENT ON DRIVE B:.
;
         JMP  SERASK         ;IF ERROR BACK TO GET A NEW
                             ;..SERIAL NUMBER OR TO EXIT
SERCP1:  MVI  B,00H          ;INDEX COUNTER FOR TABLE VALUES
SERIST:  MOV
         L,B
         MVI  H,00H
         DAD  H              ;DOUBLE TO WORDS
         LXI  D,INSTAB       ;INTO TABLE
         DAD  D
         MOV  A,M            ;GET RECORD NUMBER FOR PLACE
         STA  PROGFCB
              +32            ;SET TO READ THIS RECORD
         INX  H
         MOV  C,M            ;GET BYTE LOCATION OF COUNTER
         PUSH B
         LXI  D,PROGFCB ;USE PROG FCB TO READ
         MVI  C,READ
         CALL BDOS           ;GO READ SECTOR
         POP  B              ;INDEX TO LENGTH
         MOV  L,C
         MVI  H,00H
         LXI  D,080H         ;BASE OF DEFAULT BUFFER
         DAD  D
         MOV  C,M            ;GET LENGTH
         INX  H              ;POINT TO NEXT BUFFER BYTE
```

(continued on next page)

```
         LXI   D,SERSTR      ;POINT (DE) TO SERIAL LOCATION
;
MOVLP:   MOV   A,H           ;SEE IF PAST THE END OF BUFFER
         CPI   01H
         JNZ   SAMSEC        ;STILL IN THE SAME SECTOR
;        MVI   H,00H         ;RESET TO NEXT SECTOR BASE
         PUSH  B
         PUSH  H
         PUSH  D
         LXI   H,PROGFCB
               +32           ;DECREASE RECORD FOR WRITE
         DCR   M
         LXI   D,PROGFCB
         MVI   C,WRITE       ;WRITE LAST SECTOR
         CALL  BDOS
         LXI   D,PROGFCB
         MVI   C,READ        ;READ NEXT SECTOR
         CALL  BDOS
         POP   D
         POP   H
         POP   B
;
SAMSEC:  PUSH  B
         LDAX  D             ;GET A SERIAL NUMBER BYTE
         MOV   M,A           ;AND SLAM INTO BUFFER
         POP   B
         INX   H
         INX   D
         DCR   C             ;DONE ALL BYTES HERE YET
         JNZ   MOVLP
;
         PUSH  B
         LXI   H,PROGFCB
               +32           ;SET BACK CURRENT RECORD FOR
                             ;WRITE
         DCR   M
         LXI   D,PROGFCB
         MVI   C,WRITE       ;REWRITE THIS SECTOR
         CALL  BDOS
         POP   B
         INR   B             ;BUMP TABLE SCAN INDEX
         LDA   TABLEN        ;CHECK FOR DONE
         CMP   B
         JNC   SERIST        ;GO FOR NEXT TABLE ENTRY
;
;PUT IN LOGIC HERE TO SPECIFY THE NEXT OF SEQUENTIAL SERIAL
;NUMBERS OR TO GO BACK TO THE TOP OF THE PROGRAM TO GET A
;NEW SERIAL NUMBER.
;
;
;
;PARAMETER DATA AREA FOR SERIAL NUMBER PROGRAM
;
;
;'PROG.COM" FILE ACCESS CONTROL BLOCK
;
PROGFCB:
         DB    'B'-040H      ;DISK DRIVE B: ALL THE TIME
         DB    'PROG    COM',0,0,0,0
         DS    17            ;ALLOCATION SPACE
;
;
;
```

```
;SERIAL NUMBER INSERTION POINT REFERENCE TABLE
;
INSTAB:  DB   ((SERA-0100H-1)/128)          ;RECORD NUMBER
         DB   ((SERA-0100H-1) AND 07FH)     ;BYTE OFFSET
         DB   ((SERB-0100H-1)/128)          ;RECORD NUMBER
         DB   ((SERB-0100H-1) AND 07FH)     ;BYTE OFFSET
         DB   ((SERC-0100H-1)/128)          ;RECORD NUMBER
         DB   ((SERC-0100H-1) AND 07FH)     ;BYTE OFFSET
         DB   ((SERD-0100H-1)/128)          ;RECORD NUMBER
         DB   ((SERD-0100H-1) AND 07FH)     ;BYTE OFFSET
         DB   ((SERE-0100H-1)/128)          ;RECORD NUMBER
         DB   ((SERE-0100H-1) AND 07FH)     ;BYTE OFFSET
         DB   ((SERF-0100H-1)/128)          ;RECORD NUMBER
         DB   ((SERF-0100H-1) AND 07FH)     ;BYTE OFFSET
         DB   ((SERG-0100H-1)/128)          ;RECORD NUMBER
         DB   ((SERG-0100H-1) AND 07FH)     ;BYTE OFFSET
         DB   ((SERH-0100H-1)/128)          ;RECORD NUMBER
         DB   ((SERH-0100H-1) AND 07FH)     ;BYTE OFFSET
;
TABLEN:  DB   (($-INSTAB)/2)-1              ;NUMBER OF TABLE
                                            ;ENTRIES MINUS 1 FOR
                                            ;LOOP EASE
SERSTR:  DS   10H                           ;PLACE TO KEEP BINARY
                                            ;SERIAL NUMBER
;
;
END
;
;
;...END OF SERIAL NUMBER INSERT PROGRAM
```

The next and final example is a fully functional program that uses random record I/O under CP/M 2.2 to perform a useful function. The program mixes up the records of a file in an ordered yet bizarre way in order that the file contents may be encoded to prevent its use until such time that it is unscrambled. The unmixing process is also performed by the program below. The records or "sectors" of the file are mixed and unmixed in place on the disk in that the disk file is not copied. Random access file I/O is used to swap records directly. The comment block at the beginning of the program listing contains an explanation of the program intent and the record mixing algorithm chosen. Operation of the program, should the reader wish to utilize the encoding and decoding functions provided, is also described in the listing.

This example program is presented as a working example of random file I/O in use. Detailed description of the internal workings of the program are beyond the scope of this tutorial but may be inferred by studying the listing and reading the rather prolific comment statements. For readers that would like to avoid the aggravation of typing in the source code for the program below or for the other programs presented in this BDOS tutorial series, Part I in *Lifelines*, November 1982 and Part II in *Lifelines*, January 1983, a machine readable copy of the source code files on an eight inch single density diskette may be obtained from Michael J. Karas, 2468 Hansen Court, Simi Valley, California 93065. Please send diskettes preformatted, labeled and in a returnable mailer of some sort. Also include either stamps or money for return postage (no postage meter tapes, those are accepted on date of printing only) for your return package.

LISTING FOR SECRET.ASM A RANDOM I/O PROGRAM EXAMPLE

```
;
;
;RANDOM RECORD I/O DEMONSTRATION FOR CP/M 2.2
;
; THIS THIRD LEVEL DEMONSTRATION PROGRAM IS DESIGNED
; TO DEMONSTRATE RANDOM FILES BY DEVELOPING A
; 'NOT NECESSARILY PRACTICAL' ALGORITHM FOR EN-
; CODING A PROGRAM FILE ON A DISK. THE INTENT IS TO
; MAKE THE TRANSMISSION OF AN OBJECT FILE ARBITRARILY
; SCRAMBLED ON A 128 BYTE BY 128 BYTE RECORD BASIS
; SUCH THAT IF THE TRANSMITTED FILE, EITHER ON FLOPPY
; DISKETTE OR ON THE PHONE LINE WERE INTERCEPTED BY
; AN ILLICIT THIRD PARTY, THEN THE THIRD PARTY WOULD
; RECEIVE GARBAGE UNLESS HE HAD POSSESSION OF THE
; DECODING ALGORITHM. THIS PROGRAM WILL IMPLEMENT
; SUCH AN ALGORITHM IN BOTH AN ENCODING AND
; DECODING FORMAT. HERE IS THE ALGORITHM USED. (OB
; VIOUSLY THE FACT THAT THIS APPEARS IN THE
; PUBLIC IMAGE AS A MAGAZINE ARTICLE WILL PREVENT THE
; FOLLOWING ALGORITHM TO BE OF 'SECRET' USE).
;
; THE OPERATOR ENTERS THE COMMAND TO RUN THE PRO-
; GRAM AS:


; A>SECRET filename.typ E<cr>
;
;     where filename.typ is the
;     file to encode. And "E"
;     indicates to encode the file
;
;or:
;
;A>SECRET filename.typ D<cr>
;
;     where filename.typ is the
;     file to decode. And "D"
;     indicates to decode the file
;
; THE ENCODING PROCESS WRITES THE ENCODED FILE RIGHT
; IN PLACE WITHIN THE USER SPECIFIED FILE. NO MEANS IS US
; ED TO SPECIFY IN THE ENCODED FILE THAT IT IS ENCOD
; ED. THE DECODE PROCESS READS AND DECODES THE FILE
; RIGHT IN PLACE WITHIN THE USER SPECIFIED FILE NAME. THE
; ALGORITHM LEAVES THE FIRST RECORD OF THE FILE INTACT
; AND DOES NOT ENCODE THE PART OF A FILE BEYOND 128
; RECORDS IN SIZE. FOR FILES LARGER THAN 128 RECORDS THE
; FINAL RECORDS BEYOND THE 128'TH ARE LEFT UNTOUCHED.
; THE BDOS IS CALLED TO DETERMINE THE SIZE OF THE FILE
; SO THE NUMBER OF RECORDS IN THE FILE ARE KNOWN. THIS
; NUMBER OF RECORDS WILL BE REFERRED TO HERE AS "NR'. IF
; "NR" IS GREATER THAN 128 THEN "NR" IS SET TO 128. THEN
; THE FIRST "NR-1" BYTES OF THE FIRST RECORD ARE READ SE
; QUENTIALLY TO MAKE A LIST OF ONE BYTE BINARY NUMBERS
; WITH A NUMBER OF ENTRIES EQUAL TO THE NUMBER OF
; RECORDS IN THE FILE MINUS ONE, UP TO A MAXIMUM OF 127
; NUMBERS. THIS LIST IS THEN PROCESSED TO CONVERT ALL
; OF THE NUMBERS IN THE LIST TO BE WITHIN THE RANGE OF 1
; TO "NR-1'. THIS CONVERSION IS DONE BY FIRST "ANDING"
; EACH OF THE BYTES IN THE LIST WITH A MASK. THE MASK
; HAS A NUMERICAL VALUE EQUAL TO "NR-1" ROUNDED UP
; TO THE NEXT BIGGEST [(2 ^ N) - 1] VALUE, IE IF THE FILE HAS 5
; RECORDS THE MASK IS 07H. IF THE FILE HAS 59 RECORDS THE
; MASK HAS A VALUE OF 3FH. THE LIST IS THEN SCANNED FOR
; VALUES THAT ARE GREATER THAN "NR-2'. EACH VALUE
; THAT IS GREATER THAN 'NR-2" IS DIVIDED BY TWO IGNOR
; ING THE REMAINDER. FINALLY EACH LIST VALUE IS IN
; CREMENTED BY ONE TO MAKE A REAL FILE READABLE
; RECORD NUMBER. THE LIST IS THEN USED AS A RECORD
; SCRAMBLE/UNSCRAMBLE LIST. FOR SCRAMBLING IT IS
; SCANNED FROM THE BEGINNING WHILE UNSCRAMBLING
; SCANS THE LIST FROM THE END. SCRAMBLING PROCEEDS AS
; FOLLOWS (THE UNSCRAMBLE PROCESS IS THE REVERSE): THE
; SECOND FILE RECORD IS NOW INTERCHANGED IN POSITION
; WITH THE RECORD POINTED BY THE FIRST NUMBER IN THE
; LIST. THE THIRD FILE RECORD IS INTERCHANGED WITH THE
; RECORD POINTED TO BY THE SECOND LIST VALUE. THIS PRO
; CESS CONTINUES UNTIL THE END OF THE LIST. DURING THE
; PROCESS OF INTERCHANGING THE FILE SECTORS IN THIS
; RATHER BIZARRE MANNER, EACH TIME A LIST VALUE IS
; FOUND TO HAVE A LEAST SIGNIFICANT BIT THAT IS EQUAL
; TO "1" THEN THAT RECORD HAS EACH BYTE XOR'ED WITH
; THE RECORD NUMBER.
                                  WRITTEN BY:
                                  MICHAEL J. KARAS
                                       2468 HANSEN COURT
                                       SIMI VALLEY, CA 93065
                                       (805) 527-7922

;
;
;
;SYSTEM LEVEL INTERFACE EQUATES
;
BDOS     EQU  0005H          ;SYSTEM INTERFACE VECTOR
MAKE     EQU  22             ;MAKE NEW FILE FUNCTION
SBADDR   EQU  26             ;SET DISK BUFFER ADDR
OPEN     EQU  15             ;OPEN FILE FUNCTION
CLOSE    EQU  16             ;FILE CLOSE FUNCTION
DELETE   EQU  19             ;DELETE FILE FUNCTION
RRAND    EQU  33             ;READ RANDOM FUNCTION
WRAND    EQU  34             ;WRITE RANDOM FUNCTION
WRANDF   EQU  40             ;WRITE RANDOM WITH 00 FILL
PRINT    EQU  9              ;PRINT STRING TILL $
FSIZE    EQU  35             ;COMPUTE FILE SIZE FUNCTION
DEFCB    EQU  05CH           ;DEFAULT FILE CONTROL BLOCK
DEFBUF   EQU  080H           ;DEFAULT BUFFER LOCATION
;
EXEC     EQU  8000H          ;EXECUTE SPOT FOR SMALL
                             ;PROGRAM
BOOT     EQU  0000H          ;SYSTEM REBOOT ENTRY POINT
;
;
;ASCII CHARACTER DEFINITIONS
;
CR       EQU  0DH            ;CARRIAGE RETURN
LF       EQU  0AH            ;LINE FEED
;
;
         ORG  0100H          ;START OF A PROGRAM
         LXI  SP,STACK       ;SETUP A STACK FOR EXECUTION
         LXI  D,SNGMSG
                             ;PRINT SIGNON MESSAGE
         MVI  C,PRINT
         CALL BDOS
```

(continued on next page)

```
;
;CHECK IF THERE WAS A COMMAND LINE FILE NAME
;
        LDA   DEFCB+1    ;IF FIRST BYTE 20 THEN NO NAME
        CPI   ' '
        JZ    CMDERR     ;IF NO FILE NAME PRINT ERROR
        LDA   DEFCB
              +17        ;GET OPTION CHARACTER
        CPI   'E'        ;CHECK FOR ENCODE
        JZ    PROCESS    ;GO TO PROCESS IF ENCODE
        CPI   'D'        ;CHECK IF DECODE
        JZ    PROCESS    ;GO PROCESS OF DECODE
;
CMDERR: LXI   D,ERRM1    ;PRINT ERROR MESSAGE
        MVI   C,PRINT
        CALL  BDOS
        JMP   BOOT       ;EXIT IF NO FILE NAME OR OPTION
;
;
;HERE IF AN ENTRY FILE NAME AND A VALID OPTION
;
PROCES-
S:      STA   OPTION     ;SAVE OPTION CHAR FOR LATER
                         REFERENCE
        XRA   A          ;SETUP FCB FOR OPEN
        STA   DEFCB
              +12        ;ZERO EXTENT BYTE
        STA   DEFCB
              +32        ;ZERO CURRENT RECORD BYTE
        STA   DEFCB
              +35        ;ZERO R2 BYTE
        LXI   H,0000H
        SHLD  DEFCB
              +33        ;ZERO RANDOM RECORD NUMBER
;
        MVI   C,OPEN     ;OPEN FILE USER SPECIFIED
        LXI   D,DEFCB    ;USE DEFAULT FCB BUILT BY CCP
        CALL  BDOS       ;GO ATTEMPT OPEN
        INR   A          ;CHECK IF FOUND
        JNZ   FOUND
;
        MVI   C,PRINT    ;PRINT NOT FOUND ERROR
        LXI   D,ERRM2
        CALL  BDOS
        JMP   BOOT       ;EXIT
;
;
;FOUND FILE SO LET'S NEXT COMPUTE ITS FILE SIZE
;
FOUND:  LXI   D,DEFCB    ;THAT SAME FCB AGAIN
        MVI   C,FSIZE
        CALL  BDOS       ;GET THE FILES SIZE IN RECORDS
        LHLD  DEFCB+33   ;GET SIZE OF THE FILE
        MOV   A,H        ;CHECK IF GREATER THAN 128
                         RECORDS
        ORA   A
        JNZ   TOBIG
        MOV   A,L
        ORA   A          ;CHECK IF FILE EMPTY OR ONLY ONE
                         RECORD
        JZ    TOSMALL
        CPI   1
        JZ    TOSMALL
        CPI   129
        JC    SIZINA     ;WE HAVE SIZE IN (A)
TOBIG:  MVI   A,128      ;SET SIZE TO 128 DEFAULT
SIZINA: STA   NR         ;SAVE NUMBER OF RECORDS
        JMP   READFST
;
TOSMALL:
        MVI   C,PRINT    ;PRINT FILE SIZE ERROR MESSAGE
        LXI   D,ERRM3
        CALL  BDOS
        JMP   BOOT
;
;
;READ FIRST RECORD INTO LIST BUFFER
;
READFST: LXI  D,LIST     ;SET DMA ADDRESS TO LIST BUFFER
        MVI   C,SBADDR
        CALL  BDOS
        LXI   H,0000H    ;SET FIRST RECORD
        SHLD  DEFCB+33
        XRA   A
        STA   DEFCB+35   ;CLEAR R2 BYTE
        MVI   C,RRAND    ;READ RANDOM FIRST RECORD
        LXI   D,DEFCB
        CALL  BDOS       ;NO NEED TO CHECK READ ERROR
                         BECAUSE WE KNOW THAT THESE
                         RECORDS EXIST
;
;
;HERE TO PROCESS LIST INTO A SET OF NUMBERS THAT FIT OUT
;FILE
;RECORD COUNT RANGE.
;
        LDA   NR         ;FETCH NUMBER OF RECORDS
        DCR   A          ;SET NR-1;
        MVI   B,0FFH     ;INITIAL MASK VALUE
        MVI   C,07H      ;NUMBER OF TIMES TO ROTATE FOR
                         ;MASK
;
MKLP:   RAL              ;CHECK FOR ZERO BIT IN NR-1
        JC    HMSK       ;EXIT WE HAVE OUR MASK ONE BIT
                         ;FROM (A)
        PUSH  PSW
        MOV   A,B        ;PUT A ZERO BIT INTO MASK
        ORA   A          ;CLEAR CARRY
        RAR              ;PUT ZERO IN
        MOV   B,A
        POP   PSW
        DCR   C          ;DEBUMP SHIFT COUNT
        JNZ   MKLP
;
HMSK:                    ;HERE IF (B) HAS LIST MASK VALUE
        LDA   NR         ;GET NUMBER OF VALUES IN LIST
        DCR   A
        MOV   C,A        ;PUT LOOP COUNTER INTO (C)
        MOV   D,A        ;SAVE NR-1 IN (D)
        LXI   H,LIST     ;POINT AT LIST
LSTPROC:
        MOV   A,M        ;GET A LIST BYTE
        ANA   B          ;MASK IT
        CMP   D          ;IS RESULT GREATER THAN NR-2
        JC    VALOK      ;VALUE IS OK
        ORA   A          ;DIVIDE BY TWO IF TOO BIG
        RAR
VALOK:  INR   A          ;SET VALUES TP FOR REAL RECORD
                         ;NUMBERS
        MOV   M,A        ;PUT CONVERTED NUMBER INTO LIST
                         ;AGAIN
        INX   H          ;BUMP LIST POINTER
        DCR   C          ;DEC LOOP COUNTER
        JNZ   LSTPROC    ;DO ALL BYTES OF LIST
;
;
;ENCODE/DECODE THE FILE HERE
;
ENCODE: LXI   H,LIST     ;KEEP A POINTER TO THE LIST
```

```
         LDA  OPTION        ;IF OPTION IS 'E' WE GO FORWARD
         CPI  'E'
         MVI  A,1           ;DEFAULT FORWARD CURRENT
                            RECORD
         JZ   FORWA         ;GO FORWARD
         LDA  NR            ;INDEX TO END OF LIST FOR DECODE
         DCR  A             ;SET START RECORD FOR DECODE
         MOV  E,A
         DCR  E             ;ZERO BASE INDEX
         MVI  D,0
         DAD  D
;FORWA:  SHLD LISTP         ;SAVE LIST POINTER
         STA  CURR          ;SET CURRENT RECORD NUMBER TO
                            START
         LDA  NR
         DCR  A
         STA  CNTR          ;SET NUMBER OF SWAPS
;
ENCLP:   LXI  D,BUF1        ;SET BUFFER ONE AS DMA ADDRESS
         MVI  C,SBADDR
         CALL BDOS
         LDA  CURR          ;READ CURRENT RECORD
         MOV  L,A
         MVI  H,00
         SHLD DEFCB+33      ;SET RECORD NUMBER
         LXI  D,DEFCB
         MVI  C,RRAND       ;READ THAT RECORD
         CALL BDOS
         ORA  A             ;CHECK ERROR
         JNZ  DSKERR
;
         LXI  D,BUF2        ;SET BUFFER 2 AS DMA ADDRESS
         MVI  C,SBADDR
         CALL BDOS
         LHLD LISTP         ;GET SWAP POSITION
         MOV  L,M
         MVI  H,00
         SHLD DEFCB+33      ;SET SWAP RECORD NUMBER
         LXI  D,DEFCB
         MVI  C,RRAND       ;READ SWAP RECORD
         CALL BDOS
         ORA  A             ;CHECK ERROR
         JNZ  DSKERR
;
         LHLD LISTP         ;IS SWAP RECORD AN ODD NUMB
         MOV  B,M           ;SAVE XOR PATTERN IN (B)
         MOV  A,M
         RAR
         JNC  SWRT          ;GO DO SWAP WRITE DIRECTLY IF
                            ;EVEN
         LDA  OPTION        ;WHICH BUFFER TO XOR
         LXI  H,BUF2        ;DEFAULT FOR 'E'
         CPI  'E'
         JZ   INB2          ;USE BUFFER 2
         LXI  H,BUF1        ;IF DECODE USE BUFFER 1
INB2:    MVI  C,128         ;BYTE COUNT OF XOR
XORLP:   MOV  A,M           ;GET A BYTE TO XOR
         XRA  B
         MOV  M,A           ;PUT BYTE BACK
         INX  H             ;BUMP BUFFER POINTER FOR XORING
         DCR  C             ;DEC BYTE COUNT
         JNZ  XORLP
;SWRT:   LXI  D,BUF1        ;SET BUFFER ONE AS DMA ADDRESS
         MVI  C,SBADDR
         CALL BDOS
         LHLD LISTP         ;GET SWAP POSITION
         MOV  L,M
         MVI  H,00
         SHLD DEFCB+33      ;SET SWAP RECORD NUMBER
```

```
         LXI  D,DEFCB
         MVI  C,WRAND       ;WRITE SWAP RECORD
         CALL BDOS
         ORA  A             ;CHECK ERROR
         JNZ  DSKERR;
         LXI  D,BUF2        ;SET BUFFER 2 AS DMA ADDRESS
         MVI  C,SBADDR
         CALL BDOS
         LDA  CURR          ;WRITE CURRENT RECORD
MOV      L,A
         MVI  H,00
         SHLD DEFCB+33      ;SET RECORD NUMBER
         LXI  D,DEFCB
         MVI  C,WRAND       ;WRITE THAT RECORD
         CALL BDOS
         ORA  A             ;CHECK ERROR
         JNZ  DSKERR
;
         LDA  CURR          ;FETCH LOOP PARMS
         MOV  B,A
         LHLD LISTP
         ;LDA OPTION        ;CHECK OPTION
         CPI  'E'
         JZ   INCF          ;IF ENCODE INCR FORWARD
;
DECB:    DCX  H             ;DECREMENT DOWN THROUGH
                            ;LOOP
         DCR  B
         JMP  PSVE          ;SAVE PARMS
INCF:    INX  H
         INR  B
PSVE:    SHLD LISTP         ;SAVE NEW LIST POSITION
         MOV  A,B
         STA  CURR
;        LDA  CNTR          ;FETCH LOOP COUNTER
         DCR  A
         STA  CNTR
         JNZ  ENCLP         ;GO TO LOOP TO PROCESS MORE IF
                            ;NOT DONE YET
;
;
;HERE WE ARE DONE WRITING SO LET'S CLOSE UP AND GO HOME
;
         LXI  D,DEFCB
         MVI  C,CLOSE
         CALL BDOS
         INR  A             ;CHECK ERROR CODE
         JZ   DSKERR
;        MVI  C,PRINT       ;PRINT DONE MESSAGE
         LXI  D,DONMSG
         CALL BDOS
         JMP  BOOT          ;EXIT
;
;
;EXIT POINT WITH ERROR MESSAGE IF THE DISK WRITE OPERATION
RESULTED IN AN ERROR
;
DSKERR:  LXI  D,ERRM4       ;PRINT GARBAGE FILE ERROR
         MVI  C,PRINT
         CALL BDOS
         JMP  BOOT          ;EXIT FOR THE POOR GUY
;
;
;PROGRAM OPERATIONAL MESSAGES
;
SNGMSG:
         DB   CR,LF,'MICRO RESOURCES Disk File Scramble and'
         DB   CR,LF,'Unscramble Utility Designed to Demonstrate'
         DB   CR,LF,'CP/M Ver 2.2 Random Record I/O. (1/24/82)','$'
```

```
;
DONMSG:
          DB    CR,LF,'File Processing Complete','$'
;ERRM1:   DB    CR,LF,'No File Name Specified or Improper Option','$'
;
ERRM2:    DB    CR,LF,'Specified File Not Found','$'
;
ERRM3:    DB    CR,LF,'Cannot Process Files with 0 or 1 Record(s)','$'
;
ERRM4:    DB    CR,LF,'File I/O Error, This Error Should NOT Normally'
          DB    CR,LF,'Happen, But the File is now Garbaged...','$'
;
;
;PROGRAM DATA STORAGE SECTION
;
OPTION:   DS    1                ;PLACE TO STORE COMMAND LINE
                                 ;OPTION CHAR
;
NR:       DS    1                ;NUMBER OF RECORDS TO SWAP
;
CNTR:     DS    1                ;ENCODE/DECODE LOOP COUNTER
;
CURR:     DS    1                ;CURRENT SWAP SECTOR
;
LISTP:    DS    2                ;LIST SCAN POINTER
;
LIST:     DS    128              ;LIST BUFFER
;
BUF1:     DS    128              ;DATA BUFFER 1
;
BUF2:     DS    128              ;DATA BUFFER 2
;
          DS    36
STACK     EQU   $                ;USER STACK AREA
;
;
          END
;
;
;+++...END OF FILE
```

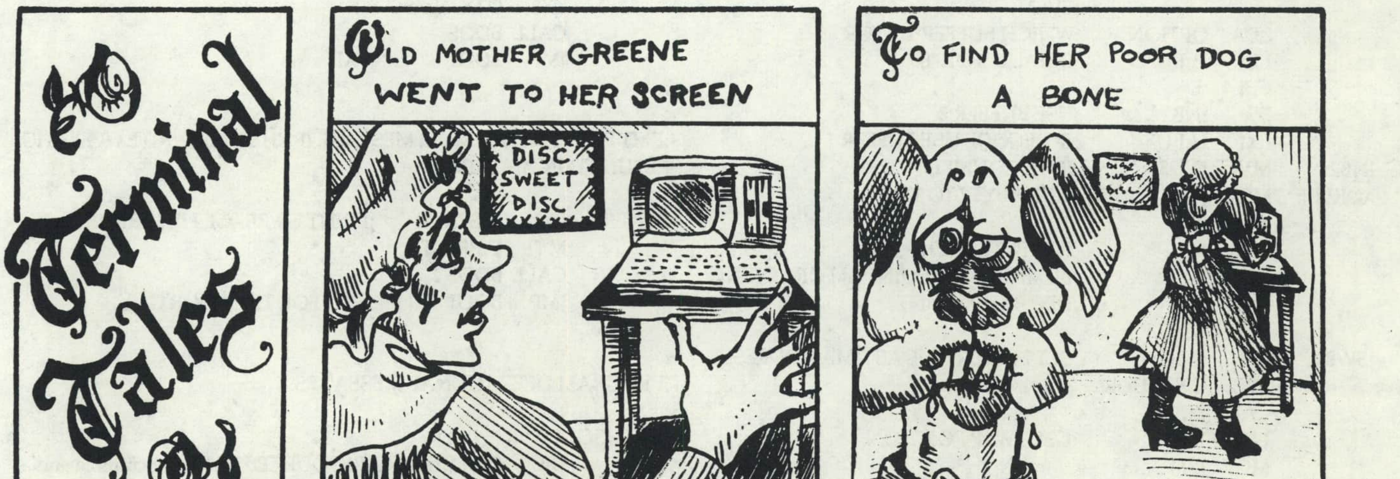
Figure 1. FILE CONTROL BLOCK DESCRIPTION

dr	f1	f2		f8	t1	t2	t3	ex	s1	s2	rc	d0		dn	cr	r0	r1	r2
00	01	02	…	08	09	10	11	12	13	14	15	16	…	31	32	33	34	35

where:

dr	drive code (0 - 16) 0 => use default drive for file access 1 => select drive A: for file access 2 => select drive B: for file access ... 16 => select drive P: for file access
f1...f8	contain the files name in ASCII upper case with high bits equal to zero.
t1,t2,t3	contain the file type in ASCII upper case with high bits normally equal zero. tn' denotes the high bit of these bit positions. t1' = 1 => Read/Only file t2' = 1 => SYS file, no DIR list
ex	contains the current extent number, normally set to 00 by the user, but is in the range 0 - 31 during file I/O.
s1	reserved for internal system use
s2	reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH system calls.
rc	record count for extent "ex," takes on values 0 to 128.
d0...dn	filled in by BDOS to indicate file group numbers for this extent.
cr	current record to read or write in a sequential file operation. Normally set to zero by the user upon initial access to a file.
r0,r1,r2	optional random record number in the range of 0 to 65535, with overflow to r2. r0/r1 are a 16 bit value in low/high byte order.

KIBITS


Terminal Tales
OLD MOTHER GREENE WENT TO HER SCREEN
DISC SWEET DISC
TO FIND HER POOR DOG A BONE
SPOT

Editorial (continued from page 2)
not data reduction. If you were to extrapolate the present forecasts for spread sheets you would conclude that within two years every man, woman and child on the face of the earth would be using VisiCalc or equivalent. We have reason to believe that this will not be the case. Generic packages will continue to play a dominant role supplemented by extensive use of graphics devices and peripherals such as the mouse.

The advent of the Lisa, introduced by Apple, marks an interesting development which will undoubtedly effect all micros. The concept is as simple as it is powerful. Instead of relying solely upon the keyboard the user instead "points" to objects called "ICONS" which are graphically depicted on the console screen. Then the user instructs the system by using metaphors to perform the various functions.

Suppose that you need a calculator. You simply point, using the mouse, to the ICON which is a graphical depiction of a calculator.

A calculator then appears, virtually lifesize on the screen. Using the mouse you point, i.e. move the cursor to the key on the calculator that you are metaphorically pressing. As various operations are carried out with the calculator the results are displayed on the calculator display.

There are of course many more sophisticated uses of such a machine but this simple example illustrates the concepts involved. The buzz words to remember are "Lisa", "mouse", "metaphor" and "icon". You'll be hearing and seeing these terms over and over again from now on.

Mice , or mices, are rapidly becoming available for all of the sixteen bit machines and together with VisiCorp's VisiOn (pronounced vis-e-on) will offer much of the same capability provided by Lisa for IBM PC's and IBM PC look-a-likes. Incidently of the many IBM PC clones that we have seen so far the Compaq is the closest to the IBMPC. Watch out when selecting an IBM PC look-alike. Most of them exhibit subtle incompatiblities which will loom up out of the swamp and bite you!!!

Micrologis offering an eight inch disk controller for the IBM-PC as well as a new board called Baby Tex which is a Z80 softcard for the Texas Instrument product called Pegasus which competes with the IBM-PC.

Apple IIe's are selling well. We saw an interesting offering at West Coast in the form of a seven slot motherboard for Apple products. The manufacturer promise a graphics board, keyboard, power supply, etc which will allow you to configure your own Apple clone.

There now exist a number of interesting boards for the Apple, e.g., an 8088 board, Digital Research's CP/M-3.0 softcard, etc. One wonders what the future holds for the Apple?

Similarly the offerings for the IBM PC are growing at an ever increasing rate.

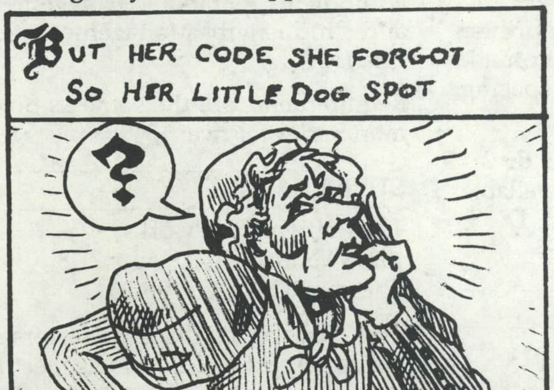
IBM's latest offering, the XT, is an interesting enhancement of the IBM PC. Basically the XT is equivalent to the IBM PC but includes a number of significant enhancements. Additional card slots are provided as well as a ten megabyte hard disk. The operating system provided is MS DOS 2.0 which has among other features a hierarchical directory. Those of you who have invested in the original IBM PC need not be intimidated, however, as there are not likely to be as many XTs in the near future as the original IBM PC.

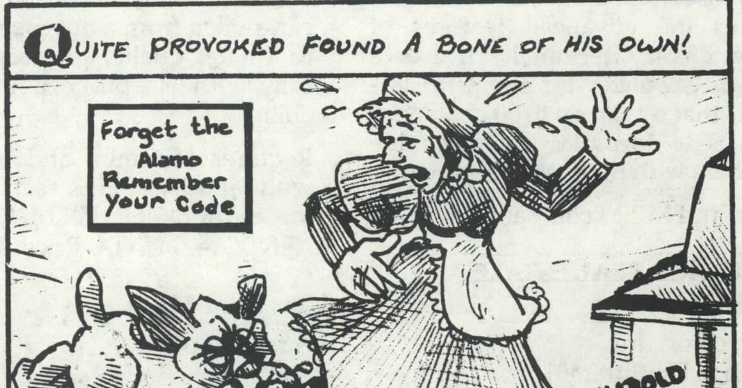
Those among you who are hardcore microcomputerists may be less than impressed by some aspects of the IBM PC but the wealth of software that is emerging will more then overshadow the alleged shortcomings suggested by the purists among us.

In the area of portables the Otrona is clearly the top of the line in today's market. This powerful machine is Z-80 based, very fast and offers substantial storage capacity. A printer port and communication port are provided which make hooking up the Hayes Smartmodem a piece of cake! The machine sports a five inch monitor which is a mixed blessing but the graphics provided more than offsets the small screen size objections. Provision for the use of an external monitor has been provided as well. The machine slides easily under an airplane seat and in fact one of the recent editorals was typed in a Las Vegas airport on an Otrona. (Airlines should provide outlets for future portable computer use in terminals).

The keyboard provided is the best we have tried on a portable and includes a number of important features such as the ability to vary printer and communications port baud rates, volume control for keyboard "click", screen intensity, etc. A screen dump feature is provided to permit dumping the screen to the printer, an alarm to permit setting times for appointments etc., and finally a calculator function is also supported.

If you are in the market for a portable machine give the Otrona serious consideration. You will find them available at prices which are astounding bargains for the value. The concept of the portable computer is not really appreciated by the general public but once you have used one you may well find that you won't travel without it!




Quite provoked found a bone of his own!
Forget the Alamo Remember your Code
Harold Black + Bess Powell
Garber '83

Product Status Reports

The new software products and new versions described below are available from their authors, computer stores, software publishers, and distributors. Information has been derived from material supplied by the authors or their agents, and *Lifelines/The Software Magazine* can assume no responsibility for its veracity. Software of interest to our readers will be tested and reviewed in depth at a later date.

New Products

THE WEDGE

Systems Plus, Inc.
1120 San Antonio Rd.
Palo Alto, CA 94303
(415) 969-7047

This electronic worksheet designed for ease of use contains lots of documentation including: quick reference cards, lessons cards, installation manual, an 80 page applications manual, and extensive HELP routines. It allows for split screen formating, insertion of rows and columns, format changes, and worksheet scrolling. The Wedge interfaces with most word processors, supports 52 columns and 400 rows, and utilizes the advanced features of many display terminals. It has a built-in calculator for use with formulas that can be up to 60 characters long. The formulas can combine numbers with multiple references.

CP/M and MP/M compatible.

QUIKCALC REAL ESTATE INVESTOR

Simple Soft, Inc.
480 Eage Dr. Suite 101
Elk Grove, IL 60007
(312) 364-0752

These financial templates are for use with Supercalc or Visicalc and are designed for novice users. They are made up of two parts: Individual Residence and Income Property. Functions include: conventional mortgages, balloon payments, variable rate mortgages, an interest only loan, complex financing structures, expense schedules, cash flows, tax benefits, and internal rate of return. The Individual Residence model shows all expenses year by year, disposable income after housing costs, investment requirements and tax benefits of ownership. The Income Property model shows all cash flows, a complete depreciation schedule (1981 ACRS depreciation), operating ratios, detailed investment basis schedule, projection of future sales price, and internal rate of return calculation.

Requires 64K.
Available for 8″ CP/M, IBM-PC, Apple, Xerox 820, Victor 9000, Kayro, and Osborne. Price $129.95

MemoPlan

Chang Labs
5300 Stevens Creek Blvd. Suite 200
San Jose, CA 95129
(408)246-8020

This concurrent word processing program has a split screen feature for work on two documents simultaneously, and has the ability to have up to five documents available at a time, sequentially with a keystroke. It automatically recovers documents following power outages and retrieves accidently deleted material. MemoPlan matches all printers and can switch from a dot-matrix printer to a letter quality printer automatically, with true proportional spacing ability.

Requires 64K min. and two drives with capacity of 150K each. Available for 8- or 16-bit CP/M, PC DOS, XENIX, and UNIX. Price: $195

Tax Mini-Miser

Sunrise Software, Inc
36 Palm Ct.
Menlo Park, CA 94025

This income tax planning software program allows the user to compute and compare the tax consequences of alternative tax strategies and to do projections for up to six years. It automatically computes the regular tax, income averaging, alternative minimum and preference taxes for each year or alternative, and points out which method yields the lowest tax.

Available for IBM PC and Apple.
Price: $295, annual updates: $50 to $75

Solomon Series I: General Accounting

Computech Group Inc.
Main Line Industrial Park
Lee Blvd.
Frazer, PA 19355
(215) 644-3344

This accounting package contains the functions: Accounts Payable, Accounts Receivable, General Ledger, Payroll, Order Entry and Invoicing, Cash Receipts and Dispursements, and Fixed Assets management. It is designed to be as foolproof as possible. Batch balancing techniques are used whenever dollar amounts have to be recorded, to double check the accuracy of the amounts entered.
Requires Z/80 system, CP/M, 64K, Two 8″ disk drives. Price: $2595

Solomon Series II: General Accounting with Job Costing

Computech Group Inc.
(for address see above)

This accounting package is the same as Solomon Series I except, for the addition of Operations Management and Productivity Management: Job Costing.

Requirements are the same as Solomon Series I. Price: $3495

SUPER ZAP

The Software Toolworks
15233 Ventura Blvd. Suite 1118
Sherman Oaks, CA 91403

This disk editor displays sectors in hex, octal and ASCII formats. Sectors may be accessed by file or by aboslute sector number. Data may be changed by moving the cursor to a byte and typing directly on the display.
Available on 8″ CP/M, Osborne, Heath/Zenith, and Xerox/Kaypro.
Price: $26.95

MicroTLX

Systems Plus
(for address see above)

This program turns your computer into a Telex. All you need is MicroTLX and a modem. MicroTLX provides automatic dialing, automatic answer, unattended operation and automatic retry of unanswered calls. You are allowed to send and receive Telex, TWX, Mailgram, International Telex, Cable and Telegram messages.

Requires CP/M and modem. Price: $150

New Versions

SAPANA-MAIL-TRACK-I

Version 1.1
Sapana Micro Software
1305 South Rouse
Pittsburg, KS 66762

This mailing list program comes with Sapana-LetterMerge program to print form letters. Each address entry can have the following items: Telephone Number (10 digits), Entry code (1-8 groups, 255 possible combinations), Last name and first name (total 30 chacracters), Company name (30 characters), Street address (30 characters), City (20 characters), State (2 characters), and Zip (9 digits). It keeps the mailing list in ZIP code order as you enter the addresses. It can warn you of duplicate entries and can handle up to 5000 addresses on a hard disk, 1100 on a single-sided disk and 2200 on a double-sided disk.

The Sapana-Mail-Track-I can handle 5 and 9 digit ZIP codes for domestic addresses. It allows foreign and domestic addresses in the same file. It searches and sorts across any of the items. Each label can belong to one or more of the eight possible groups and, can attach a 34-character long message with every label. It can print multiple copies (1-32767), one to four across. It can repeat items needed to be entered just once.

Requires IBM PC, 64K, and MS-DOS. Price: $29.95 – send $5 with the old version for the update.

POWER

Version 3.3
COMPUTING!
2519 Greenwich San Francisco, CA 94123

This utility now includes a password protection feature for sensitive data file. POWER lets the user sort disk directories in four different ways, subdivide each user area into eight sub areas to group programs and files, issue automatic Control C to CP/M, and optionally keep POWER and its menu functions in control of CP/M at all times. Other functions that have been enhanced include: Copy, Reclaim, Disk Test, Type Hex, Dump Hex, Read Track and Sector, Rename, Load, and Save. All of the functions are explained in the new manual.

Available for CP/M-80, CP/M-86, and MP/M. Price: $149 and $198 for MP/M – send $35 with original disk for update.

PAS-3 Update

Version 1.92 Medical:

1. A feature to print on patient's bill a message on whether the insurance company was billed for this charge was added in version 1.91. This new item was not range checked and it was possible to format the bill so an SB Error would occur. This has been corrected.
2. The ability to specify the destination drive for Utility items, moves patient to inactive disk, split the disk, and copy the data files was added in version 1.91. The drive input was not automatically paded with a colon. This has been corrected.
3. A new feature has been added to the Daily Charges and Receipts report. The user can now start the report at any page. This allows stopping the report and then restart printing at the point where the report was stopped. We will be adding this feature to all reports in future versions of PAS-3 Medical.

Version 1.74 Dental:

1. The CB-80 Version of PAS-3 Dental running under MP/M would give an EX Error when chaining from program to program if MP/M had files open on another drive. The CBASIC Version did not give this error but gave a warning message that disk reset denied because of open files. We have modified PAS-3 to prev ent this error EX when running on a hard disk. You must run MP/M on a hard disk system and not a floppy system for it to operate correctly with PAS-3 Dental in CB-80.

2. We have added the ability to start the Daily Charges and Receipts report at any page in Version 1.74 of PAS-3 Dental. This feature will be added to all reports in future Versions of PAS-3.

3. We have added two new options to the Pull Program. Option R allows the operator to view all recalls established for this acount. Option P gives the operator a print-out of the full account record for the patient. Both these options com up as a menu selection in Pull and are self explanatory.

T/MAKER-III DEMO's

There is a special version of T/Maker for CP/M80 which has no Save command and only permits printing on the screen.

There is a special version of T/Maker for the IBM which has no Save command and only permits printing on the screen.

OTHER NEW VERSIONS

1. MAGIC PRINT (NEC Spinwriters) v1.32s
2. PLAN80 (CP/M-80, CP/M-86, IBM PCDOS) v2.5
M/SORT v1.03
PLINK-II v1.16
MICROSTAT (for BASIC-80) v2.09d
muLISP-80 v2.15
muSIMP-80 v2.14,
Legal Time Accounting (UNIVAIR Series 9000) v1.09
FPL v3.0

Bugs

Bugs

The following bug exists in PANEL-PC and PANEL-86 version 4.01.

Symptom - The "LOAD ALL" option within the "BULK LOAD" facility in program RANDATA results in incorrect data being loaded onto the file.

Solution - Use the "DISPLAY ALL" option instead of the "LOAD ALL" option. This process performs a similar function, but it runs slightly slower because it displays each record as it is loaded. It also works correctly. The problem will be corrected in version 4.02 soon to be released.

LIFELINES™/The Software Magazine™
1651 Third Avenue, New York, New York 10028

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

[redacted]