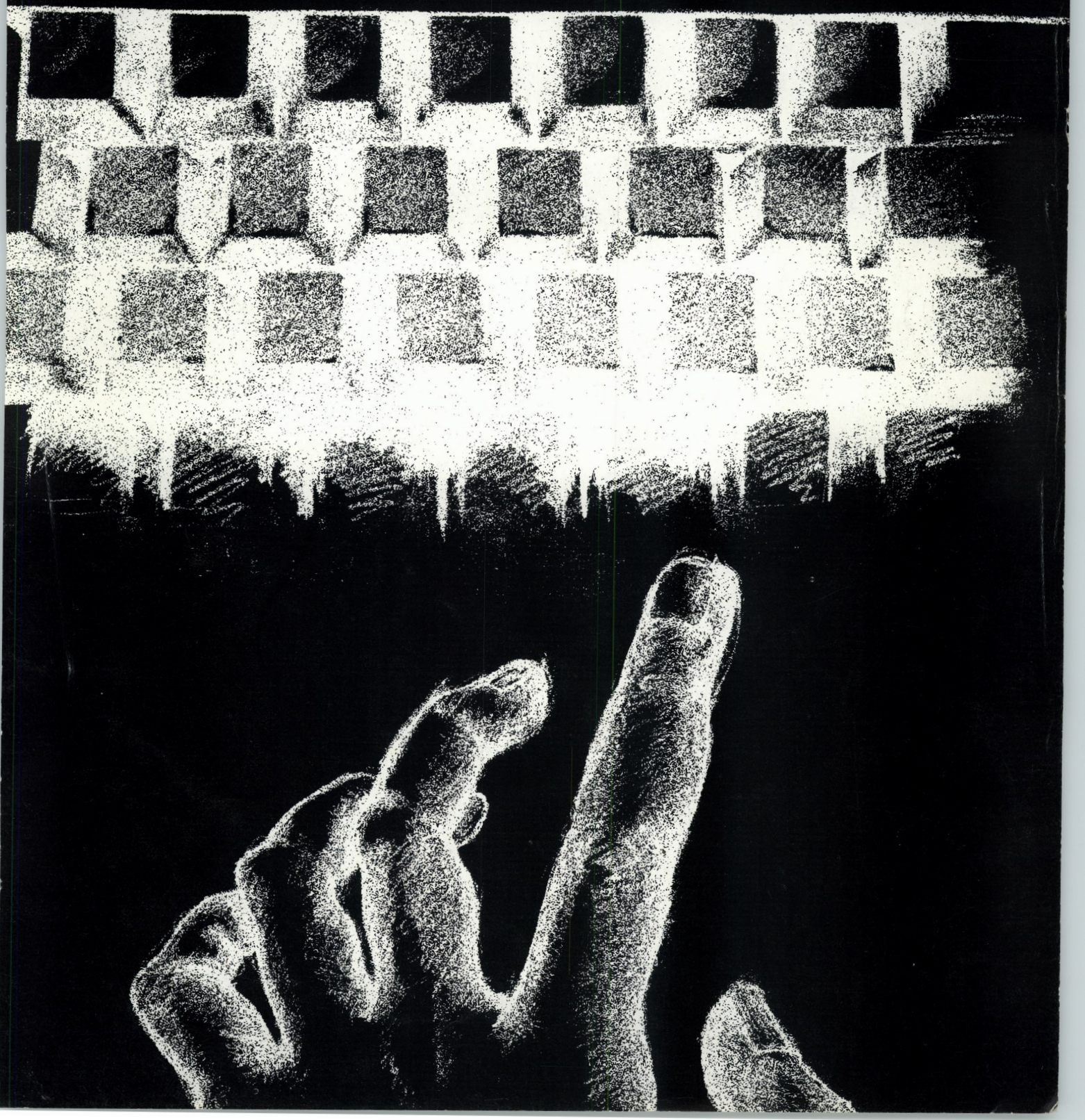
LIFELINES™

# The Software Magazine™

$3.00 February 1983 Volume III, No. 9 (ISSN 0279-2575, USPS 597-830)



### *It was inevitable.*

*In the beginning, there was the data base management system. Powerful, but only if you knew programming. Then came the program generator—anyone could use it, but why bother to generate poorly written BASIC programs? Now there's the best of both worlds with QUICKCODE™, the data base program generator.*

### *Power and ease of use.*

*Fox & Geller's QUICKCODE™ combines power and ease of use in one neat package. It writes consise dBASE II™ programs to set up and maintain any kind of database. You can run them as is or customize them in seconds. And you'll still have all the power of dBASE II™ at your disposal: query language, report generator, and so on. But just as important: you don't need to do any programming. Just draw your data entry form on the screen and you're in business. Typical time to set up a customer list or order file: 5 minutes.*

### *The Wordstar connection.*

*QUICKCODE™ also gives you the ability to transfer your dBASE II™ data into Wordstar®/Mailmerge™ files for word processing and form letters. So you can get the most from two software bestsellers: dBASE II™ and Wordstar®.*

*(Software dealers: DOUBLE YOUR SALES!)*

### *That's not all . . .*

*There are lots of other features, like form and report generation up to 132 characters wide, four-up mailing labels, three kinds of data validation, four new data types not found in dBASE II™ itself, data base keys, and menu generators. You really have to see it to believe it.*

### *It's your move.*

*Now it's up to you to take advantage of this latest development in software. Why waste any more time writing programs or paying someone to write them for you?*

*Fox & Geller's QUICKCODE™: $295.00.*

*QUICKCODE is now available for the IBM -PC with the Xedex Baby Blue Card.*

---

# QUICKCODE™

## The dBASE II™ Program Generator

*available at Lifeboat Associates*

## FOX&GELLER

*Fox & Geller, Inc.*
*P.O. Box 1053*
*Teaneck, NJ 07666*
*201-837-0142*

LIFELINES™

# The Software Magazine™



## Opinion



## CP/M® Users Group



## Product Status Reports



## Features



## Miscellaneous



 The single issue price is $3.00 for copies sent to destinations in the U.S., Canada, or Mexico. The single issue price for copies sent to all other countries is $4.30. All checks should be made payable to Lifelines Publishing Corporation. Foreign checks must be in U.S. dollars, drawn on a U.S. bank; checks, money order, VISA, and MasterCard are acceptable. All orders must be pre-paid. Please send all correspondence to the Publisher at the address below.

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

**The Best Is Yet To Be...**

The year end was marked by a large number of frenetic calls from people wanting to make their microcomputer purchase now, to get the best tax treatment. The IBM PC won out with virtually no competition. Even those who were uncertain about the advantage of sixteen-bits retreated to the rationale that if IBM produced it, the IBM PC will be popular for a long time to come, be well supported and offer excellent resale value.

The number of portable machines is on the rise and scores of pseudo IBM-compatible machines will undoubtedly emerge in the months ahead. Watch out when making your selection that the machine of your choice is in fact sufficiently close to the IBM design to preclude problems particularly with applications involving graphics and telecommunications.

Be sure to inquire whether or not a softcard is available for your machine should you need to run an application available only in an eight-bit configuration.

Don't expect worlds of difference in execution speed with a program that you have previously used in an eight-bit machine. In some cases yes... in many cases no.

As the year ended, Sony announced the release of the Sony Watchman. This exciting development is a small hand-held black/white television which is otherwise of a plain vanilla variety; plain vanilla that is except for one tiny (no pun intended) aspect – the implementation of a flat screen CRT. As you recall we have been writing about this fascinating piece of technology for some time. The screen size is approximately 300 mm measured diagonally and the resolution is of course excellent. But the interesting thing about this micro-television is the innovative flat screen cathode ray tube employed.

The glass envelope is designed with a flat surface which the observer views through to the back side of the envelope, which is coated with an electro-luminescent phosphor. The electron beam which paints the screen enters this chamber parallel to the viewing screen and is deflected towards the back of the envelope. This is roughly equivalent to viewing the television in your living room from inside the tube looking at the back side of the screen.

So what??? You are undoubtedly thinking "what's this got to do with micros???" The answer is that the truly portable computer is now within our grasp. Consider a machine with a flat screen CRT approximately five inches across a diagonal. The unit would be about one and a quarter inches thick and roughly the size of an eight and one half by eleven inch piece of paper. The memory employed would be CMOS to minimize power consumption. Bubble memory could be employed, but if memory density continues to increase, perhaps only RAM would be present. The initial system would have logical mass storage devices similar to floppies. Later a virtual mass storage device will be assumed as the last vestiges of floppies fade into the sunset. Floppies will undoubtedly be employed for some time to come, but will revert to the role of true peripheral devices.

The processor might well be provided by National Semiconductor with variable microcoding possible to permit it to emulate 8080, Z80, 8088, etc., depending on the software to be run. An integral modem will permit telecommunications at 300-1200 Baud. Additional ports will support printers and high speed I/O to mass storage devices. An auxiliary monitor will provide a large screen console display, if desired.

The price might well be under one thousand dollars and ultimately between one and two hundred dollars. The key here is that the Japanese are pushing the flat screen technology and are the world masters of high volume production of high technology devices. But the best part of all is that this notebook-sized machine will run all of the existing software with virtually no modification. Those of you wondering what the world of microcomputers will bring next need wonder no longer.

Texas Instruments is offering an information service called TEXNET Information Service, which provides more than 1200 programs and services. Included is UPI News Service, electronic mail, electronic travel service, educational programs, consumer aids, sports news, financial services, market reports, portfolio management, international document research, Legi-Slate, CompuStar, etc.

Using the TI-99/4A, all of the following are readily available: TI news, software exchange, voice chat (a library for owners of the TI voice synthesizer), phonetic dictionary, software directory, user's group, graphics library, music and sound library, submit (mechanism for users to submit programs). Thus the push is on to get home users linked to the outside world and a host of services and software.

It is rumored that the FCC has instituted some digital radio rule changes effective October 28, 1982, that relax the limitations on digital transmissions in the amateur bands.

The new limits will be as follows:

| Bands | Max Bit Rate | Max Bandwidth |
|---|---|---|
| 10m | 1200 | ? |
| 6 | 19.6K | 20khz |
| 2m | 19.6K | 20khz |
| 1.25+up | 56K | 50khz |

The new rules also allow non-ASCII codes to be used, provided that both stations in a contact are within the continental United States. This is exciting news, if true, and will hopefully mark the advent of repeaters which will permit widespread transmission of software on a nationwide basis.

The Z8000 has won a reprieve from total obscurity with the release of Olivetti's Z8000-based micro. Interestingly enough an 8086 softcard is reported to be available as well. This is the first time that a sixteen bit microcomputer has been available with provision for incorporation of an additional sixteen-bit microprocessor. Softcards continue to provide the bridges to the various vast reservoirs of software. (continued on page 17)

Feature

# Chart of Accounts Program Utilizing Access Manager

Bruce H. Hunter

## The Program

Since we have covered the majority of the available functions and had so much "fun" with them in two earlier issues of *Lifelines/The Software Magazine,* it is a good time to put them to use. The "Chart of Accounts" program below is a skeletal data base program to create and maintain a chart of accounts. A chart of accounts is similar to a mail list, but its purpose is to correlate the account numbers with the name and mailing address of the account. It is used in conjunction with a business system data base, and its portion of the data base would normally interface with the payables program of the system. Again, little effort has been taken to handle errors and bulletproof the routines, since it would have lengthened the program unnecessarily for the purpose of this article (and possibly obscured the clarity of the code). Besides, I have put in enough backwards gotos to invite plenty of criticism without adding more fuel to the fire.

When you are looking at this code, bear its rudimentary nature in mind, because you aren't going to see any of the polished error handling and exception processing routines or any other program essentials and niceties you would see were it a "real life" program. Also, I hope all you BASIC and Pascal programmers don't object to my PL/I source code. (PL/I is particularly appropriate for this article because the declarations of PL/I should help in defining the data types of the variables and parameters.) But remember, AM-80 is accessible from all three Digital Research compiled languages (PL/I-80, CB-80, and Pascal MT+). Even more intriguing is the ability of AM-80 to access the *same* data base from all three languages, so that a very effective data base system could conceivably be written in BASIC, Pascal and PL/I. With the program being structured "top down," let's start at the top. The declarations, precompiler replacements, and includes are essentially the same as the in-the-chart initialization program (in the October article), declaring the global variables and defining the set-up parameters. The first executable line of the program is a call to the procedure OPEN.

**OPEN** defines the one data file and the two index files. The data file consists of the record fields of the data structure declared in "data_buffer" at the top of the program. It coincidentally has a record length of 128 bytes, but the system will take a length of darn near anything. The file name is "chartdb.dat" and the OPENDAT routine opens it whether it exists or not. If it does exist, it is not overwritten. Instead, it is opened for update like any random or direct file. The program is constructed to automatically open the files on its (the program's) own opening, and they must be closed before exiting the program. If the program were prematurely closed by a power failure or a boot, the file directories would not be rewritten. The file would be corrupted.

There are two index files in use: one for the account name and the other for the account number. In a "real world" program for a chart of accounts, these two keys would normally be accessed. Both are opened as alpha keys in spite of the fact that the number key could have just as easily been a numeric. It was left alpha (type 0) to avoid cluttering the code with the necessary housekeeping to put the numeric data in reverse byte order and again reverse it on return. The account number key has no duplicate suffixes since we would not want to have duplicate account numbers. On the other hand, the name keys do have a duplicate key suffix since it is reasonable to assume that a name can be duplicated. The 11 byte key length will give a string length of 9 when 2 bytes have been subtracted for the suffix bytes. Because of the brevity of the key length, the names "Digital Research, Inc." and "Digital Equipment Corp." would be both stored as "Digital" (note the trailing blank is part of the key). Consequently, a duplicate key suffix is necessary. It also points out the elusiveness of accessing inexact names as opposed to ever so exact numbers.

## MENU

The program is menu-driven. Aptly, the first procedure is menu. The options available are:

Update
Search for an individual account number
Search for an individual account name
List all the accounts by number
List all the accounts by name
Delete accounts
Return the file(s) statistics
Close the files and return to the system

Since PL/I has no true case structure, we sneakily branch to the choice given by using subscripted labels. The branch now calls the pertinent procedures. In BASIC it would have been handled with an ON GOTO branching to a series of GOSUBs.

**UPDATE**

Update handles the pedestrian chore of inputting the record fields of the data structure "data_buffer", then verifying it before saving the information. The functions WRTDAT and ADDKEY write the data files and key files. Note the use of the function NEWREC to return the next available record number to the program. If a number of deletions had occurred, it would have taken the data record numbers of the deleted records. If there are no deletions, it gives the next available record number straight away. ADDKEY returns an update code (ud_code) to signal the successful or unsuccessful addition of a new index. If no addition has been accomplished, the code cryptically explains why.

**SEARCH_NUMBER**

The procedure for searching for a data record by number is short and to the point. Either an exact match is found for a key value or it is not. The function GETKEY searches for the target key, and it returns either the associated data record number or a zero to signal its failure to do so.

**SEARCH_NAME**

The procedure SEARCH_NAME is not as

(continued on next page)

straightforward as SEARCH_NUMBER. It is burdened by the fact that names are difficult to match exactly, so an exact match cannot be expected consistently. This procedure is lengthy and deserves explanation. Recall we have chosen a key length of 11 for the name key. A buffer KEYVAL is initialized with the contents of 11 spaces

```
                /*12345678901*/
    idxval = '           ';
```

IDXVAL is passed as an argument to the function SERKEY and will be overwritten, left justified with the key name. If found, the remaining 020hs will be padded to the right.

for example: "index

The key value to be searched for (or target key) is also passed to SERKEY. The routine will now truncate or pad the name to a length of 11. The function will return either a data record number or a zero to signify its failure to do so. Now the number returned is, in fact, the first key that the system can find that is equal to or greater than the target key. Were it "A", it would more than likely return the data record number of the first alpha key in the key file. Our program takes the data record number returned and passes it to the procedure PRINT_DATA which prints the record fields to the screen.

A "sub menu", the search menu, is now displayed to offer the opportunity of exiting the procedure, or searching forward or backward in ASCII order through the index file. Should a backward search be chosen, PRVKEY will return the preceding key, while NXTKEY will give the following key. The functions SERKEY, NXTKEY and PRVKEY join in a powerful trio to overcome the inexactness of name searching.

**LIST_ACCOUNT_NUMBER**

This procedure should have been named "list_data records_by_account_number," but that's a bit long, even if PL/I will take a 31 character variable or label name. The procedure takes no user input. Instead, it immediately sets about getting the first key in the number key index file. Being a B+tree key file, the keys have been placed in ascending order. Now they only have to be read sequentially to return them in sorted ascending order. The beauty of the B+tree now becomes more apparent. Once the first key has caused the corresponding data record to be printed, the program enters a loop to print the data in each successive key reference by means of the NXTKEY function. When NXTKEY returns a zero signaling the end of the index file, the loop is exited.

**LIST_ACCOUNT_NAME**

This procedure accomplishes the same function as LIST-ACCOUNT-NUMBER but accesses the name key file rather than the number key file.

DELETE

The deletion of a data record and its associated keys should and could be simple and straightforward except for the difficulty in the inexactness of name keys. In the procedure DELETE, the user inputs the number to be deleted. It is then used as an argument to GETKEY to call the data record to be displayed for verification. As usual, if a zero is returned by GETKEY, a scramble ensues to get another number or quit.

Next, the data record is read to get the account name to pass it to the DELKEY function to delete the index entry. In a real life programming environment, a much longer and discriminating algorithm would have to be devised to retrieve the name from the data record, justify and pad it, iterate the GETKEY routine, and either verify the correctness of the name key or search backward and forward to be sure to obtain the proper name.

The first deletion takes place removing the account number key. Next the data record is marked for deletion with RETREC. It will no longer be available for reading and will be overwritten by another record number when its number is picked off the returned record stack. Finally, the last DELKEY removes the name index.

STATS

Data base statistics are returned in this procedure. NOKEYS returns the number of keys in each of the key files. NMNODS yields the number of nodes. For a more detailed explanation, go back to the explanation of NMNODS in the main article. The size of the data file in records is returned by GETDFS.

PRINT_DAT

Print data is a simple procedure to print the data record to the screen.

ERROR

The error routine is a quick and dirty way to prevent the program from crashing on an AM-80 initiated error. Again, in a real world situation, a great deal more programming would be required to keep the program on its feet after encountering an accessing error. To keep from "re-inventing the wheel,"an error handling macro should be devised and separately compiled to be linked after the compilation of the main.

REBUILD

The file rebuilding procedure lurks here at the bottom of the code, hoping it will never have to be invoked. If the file has been corrupted, compromised, or otherwise clobbered by virtue of the directory not having been updated since it was opened, the data and index files are effectively unreadable. The first opening of the data file will detect this condition, and getting an error 70 from the function ERRCOD which has no arguments, it will pass control of the program to REBUILD. This procedure is almost identical to the opening functions, except that OPNDAT and OPNIDX have been replaced by OPRDAT and OPRIDX. The function reconstructs the errant directory entries to agree to the data base as it existed before it was crunched. It is a great deal easier to corrupt the file than it appears. I'm telling you this from hard experience. This is just about a mandatory set of instructions.

CONVERT

Sounds like there is a missionary hiding around the corner. BASIC's CHR$() returns a simple ASCII representation of a number. PL/I does not. The built in PL/I function, character(), returns an ASCII number padded with blanks to the left. Convert strips the blanks before returning the "number."

## Parting Comments

Well, that's the program. On the other hand, that is not all there is to Access Manager. A large number of routines exist that have not been covered, particularly for multi-user environments. Data locking and unlocking alone would be interesting to explore further. Perhaps one of these days when I get a chance to look at Bill Hogan's new Godbout MP/M 8-16 system that G & G put together for him last week, I will get a chance to check out AM-80 in a

multi-user environment.

Access Manager does everything I hoped it would and more, and it does it extremely well. It is well written and thought out. Even the documentation is above standard, and I have been informed by Digital that it is even being further clarified in its next printing. A 16 bit version is on its way, and with it, the new manual. A tip of the Hunter hat to DRI for a job well done.

## Author's Postscript

I hope you have enjoyed or at least have been edified by the series of articles on Access Manager. I have managed to get the good people at Digital to send me a copy of their much advertised and wildly anticipated utility Display Manager. This will of course be the subject of my next article(s).

```
/*********************************************************************/
/*********************************************************************/
/*******                    CHTDB.PLI                         ********/
/*******                 CHART OF ACCOUNTS                    ********/
/*******              program utilizing DRI's                 ********/
/*******                 Access Manager                       ********/
/*******                    written                           ********/
/*******                  Oct 8, 1982                         ********/
/*******    by Bruce H Hunter of HUNTER STRUCTURED SOFTWARE   ********/
/*******                 Copyright(C) 1982                    ********/
/*******                     <<*>>                            ********/
/*********************************************************************/
/*******   a program to create, update, delete, and search    ********/
/*******                chart of accounts files               ********/
/*********************************************************************/
/***                                                               ***/
/***      linkage note: link chtdb,am80pl/i.IRL[S,A],am80buf.IRL   ***/
/***                                                               ***/
/*********************************************************************/

chart:
    proc options (main);
    dcl
        data_buf_ptr    pointer,
        1       data_buffer,
            3   act_name        char(32) var,
            3   act_no          char (4),
            3   address,
                5       street  char(32) var,
                5       city    char(24) var,
                5       state   char(2),
                5       zip     char(5),
            3   principal       char(24);
                                /* -----
                                    123  byte total */

    %replace
        MAX_KEY_LEN by 48,
        NAME_LEN by 14,
        TRUE by  '1'b,
        FALSE by '0'B,
        CLEAR by '^1';
    %include 'am80extr.pli';
    dcl
        dummy  char(1),
        no_lock fixed,
        file_name char(NAME_LEN) var,
        (name_index, nbr_index, name_key) char(128) var,
        (n_buf, n_keys, n_sec, n_data_files, erropt, progid, time_out) fixed,
        (drn, file_no, recd_len, act_name_len, nbr_len, name_type) fixed,
        (nbr_type, name_dup, nbr_dup, act_name_key, act_nbr_key) fixed;

    n_buf = 3;  /* index file buffers (name & number) */
    n_keys = 2; /* index files */
    n_sec = 4;  /* 1024 byte index file record length */
    n_data_files = 1;
    erropt = 1; /* trap user errors */
    progid = -1;/* program assigned id number request, ignored
                   in a single user system */
    time_out = 0;/* background server delay also ignored */
    no_lock = 0;  /* file lock request (no file lock) */

    /*********************************************************
    initilize system
    */
    progid = intusr(progid, erropt, time_out);
    if errcod() ~= 0 then
        call error(1);
    if setup (n_buf, n_keys, n_sec, n_data_files) ~= 0 then
        call error(2);
        call open;

        menu:
            proc;
            dcl
                choice  fixed;
            put list (CLEAR);
    put skip edit(
        '          MENU',
        '          ****',
        '',
        '1'         update accounts (add records)',
        '2'         search for account by number',
        '3'         search for account by name',
        '4'         list accounts by number',
        '5'         list accounts by name',
        '6'         delete account',
        '7'         return statistical data on files',
        '8'         close files & return to menu')
        (18 (skip,col(24),a));
    retry:
        put skip (2) list ('^iinput choice');
        get list (choice);
        if choice < 1 | choice > 9 then
            goto retry;
        goto q(choice);         /* case */
        q(1):
        call update;
        q(2):
        call search_number;
        q(3):
        call search_name;
        q(4):
        call list_account_number;
        q(5):
        call list_account_name;
        q(6):
        call delete;
        q(7):
        call stats;
        q(8):
        call close;
        stop;
        /**/
        end menu;

    open:
        proc;
        dcl
            record_number fixed,
            reply char (1);

        put list (CLEAR);
        put skip(4) edit(
            '          Open Index and Data File',
            '          **********************')
            (2 (skip, col(20), a));
            /**/
        file_no = -1;   /* auto file no assignment */
        recd_len = 128;
        file_name = 'chartdb.dat';
        /* open data file (create or update) */
        file_no = opndat (file_no, no_lock, file_name, recd_len); /* note:
                        the file_no is optional in single user systems */
        if errcod() = 70 then
            call rebuild;
                else
                if errcod() ~= 0 then
                    call error(3);
        data_buf_ptr = addr(data_buffer); /* set pointer to buffer */
        /*
        open index files
        */
        name_index = 'name.idx';
        nbr_index = 'nbr.idx';
        act_name_len = 11;
        nbr_len = 4;
        name_type = 0; /*alpha key */
        nbr_type = 0;  /*numeric key entered as alpha */
        name_dup = 1;  /*add duplicate key suffix if necessary */
        nbr_dup = 0;   /*no duplicate account number suffix */
        act_name_key = opnidx(-1,name_index,act_name_len,name_type,name_dup);
        act_nbr_key  = opnidx(-1, nbr_index, nbr_len, nbr_type, nbr_dup);
        call menu;
        end open;

    update:
        proc;
```

Display Manager is designed to create some pretty wild screen formatting (including what looks to be some fair graphics) and will interface with any terminal. I am anticipating it will do for the front end of my data base business programs what Access Manager is doing for the data base.

I have been threatening for some time now to write a book on PL/I-80. It's turning out to be a tutorial, heavily laced with small easy-to-follow program examples. If you have anything that you feel you would like to see in the book please let me know through *Lifelines/The Software Magazine.*

The book is not intended to be "the definitive text on PL/I", but rather an easy (as possible) reading volume, showing what makes PL/I tick, how to make it run, bugs and error codes, programming tools, etc. I'm looking forward to your input.

(continued on next page)

```
dcl
    key_name char(9) var,
    (number, reply, ud_code, ud_code2) fixed;

put list (CLEAR);
put skip(4) edit (
    ´          Update´,
    ´          ######´,
    ´´,
    ´enter EOF to quit´)
    (4(skip, col(24), a));
do while (TRUE);
   oops:
    put skip list (´account name :´);
    get edit (act_name) (a);
    if act_name = ´EOF´ | act_name = ´eof´ then
        call menu;
    on error(1)
        begin;
        put skip list (´^g^i numeric input required  ´);
        goto redo;
        end;
   redo:
    put skip list (´account number :´);
    get list (number);
    if number < 1000 | number > 9999 then
        goto redo;
    act_no = convert (number);
    put skip list (´street :´);
    get edit (street) (a);
    put skip list (´city :´);
    get edit (city) (a);
    put skip list (´state :´);
    get list (state);
    put skip list (´zip :´);
    get list (zip);
    put skip list (´principal or contact :´);
    get edit (principal) (a);
    /**/
    put skip (3) list (´^i^i*Verification*´);
    put skip (2) list (act_name,´^i´,act_no);
    put skip list (street,´,  ´,city,´,  ´,state,´,  ´,zip);
    put skip list (principal);
    put skip (2) list (´ enter 1 for corrections :´);
        get list (reply);
        if reply = 1 then
            goto oops;
        drn = newrec (file_no, no_lock); /*
            returns the next available data record number */
        if errcod() ~= 0 then
            call error(3);
        if wrtdat(file_no, drn, data_buf_ptr) ~= 0 then
            call error(4);
        /* add key values to key files */
        ud_code = addkey(act_name_key,file_no,no_lock,act_name,drn);
        if ud_code = 2 then
            put skip list(´index value ´,act_name,´ all ready in file´);
        ud_code2 = addkey(act_nbr_key,file_no,no_lock,act_no,drn);
        if ud_code2 = 2 then
            put skip list(´index value ´,act_no,´ all ready in file´);
        end;/*dowhile*/
        call menu;
    end update;


search_number:
    proc;
    dcl
        record_nbr fixed,
        target_key  char (4) var;

    put list (CLEAR);
    put skip (4) edit (
        ´   Search by Account Number´,
        ´   ************************´,
        ´´,
        ´note: enter 0 to terminate search´)
    (4 (skip, col(24),a));
    do while (TRUE);
        retry:
        put skip (2) list (´enter number of account to be searched :´);
        get list (target_key);
        if target_key = 0 then
            call menu;
        record_nbr = getkey (act_nbr_key, file_no, no_lock, target_key);
        if record_nbr = 0 then
            put skip list (´no exact match found for ´,target_key);
                else
                call print_data(record_nbr);
        end;/*do while*/
    end search_number;



search_name:
    proc;
    dcl
        (reply, record_number) fixed,
        idxval char (11) var,       /* key value found in index */
        target_key char(128) var;   /* KEYVAL */

    put list (CLEAR);
    put skip (4) edit (
        ´   Search for Account by Name´,
        ´   ************************´,
        ´´,
        ´enter the first 11 characters or less´,
        ´of the account name to be searched´,
        ´´,
        ´enter * to end search´)
        (7 (skip, col(24),a));
    do while (TRUE);
        restart:
        put skip (2) list (´enter name to be searched :´);
        get edit (target_key) (a);
        if target_key = ´*´ then
            call menu;
              /* 12345678901 */
        idxval =´          ´;
        record_number = serkey
            (act_name_key, file_no, no_lock, target_key, idxval);
        if record_number = 0 then
            put skip list (´index out of range´);
            else
                do;
                call print_data(record_number);
                sub_menu:
                put skip edit(
                    ´       Search Menu´,
                    ´       ###########´,
                    ´´,
                    ´1  exact match found, return to main menu´,
                    ´´,
                    ´2  exact match found, continue to search´,
                    ´´,
                    ´3  index too large, return previous record´,
                    ´´,
                    ´4  index too small, return next record´,
                    ´´,
                    ´5  quit, return to main menu´)
                    (12 (skip, col(24),a));
                retry:
                get list (reply);
                if reply < 1 | reply > 5 then
                    goto retry;
                goto r(reply);
                end; /* do */
                /* case */
                r(1):
                call menu;
                /* */
                r(2):
                goto restart;
                /* */
                r(3):
                      /* 12345678901 */
                idxval =´          ´;
                record_number = prvkey
                    (act_name_key, file_no, no_lock, idxval);
                if record_number = 0 then
                    do;
                    put list (´record out of range ´);
                    goto sub_menu;
                    end; /* do */
                call print_data (record_number);
                goto sub_menu;
                /* */
                r(4):
                      /* 12345678901 */
                idxval =´          ´;
                record_number = nxtkey
                    (act_name_key, file_no, no_lock, idxval);
                if record_number = 0 then
                    do;
                    put list (´record out of range ´);
                    goto sub_menu;
                    end; /* do */
                call print_data (record_number);
                goto sub_menu;
                /* */
                r(5):
                call menu;
            end; /* do while */
    end search_name;
    list_account_number:
        proc;
        dcl
            idxval char (4),
            record_number fixed;

        put list (CLEAR);
        put skip (4) list (´^i^iAccounts Listed by Number´);
        put skip list     (´^i^i************************´);
        put skip(4);
          /* 1234 */
        idxval =´    ´;
        record_number = frskey (act_nbr_key, file_no, no_lock, idxval);
        if record_number = 0 then
            do;
            put skip list (´index file is empty´);
            call menu;
            end; /* do */
        call print_data (record_number);
        do while (TRUE);
            record_number = nxtkey (act_nbr_key, file_no, no_lock, idxval);
            if record_number = 0 then
                do;
                put skip list (´end of file´);
                put skip list (´^i press enter to continue´);
                get skip;
                goto exit;
                end; /* do */
            call print_data (record_number);
            end; /* do while */
            exit:
            put skip list (´enter any key to continue´);
            get list (dummy);
```

```
    call menu;
  end list_account_number;


st_account_name:
  proc;
  dcl
      idxval char (11) var,
      record_number fixed;

  put list (CLEAR);
  put skip (4) list ('^i^iAccounts Listed by Name');
  put skip list     ('^i^i**************************');
  put skip(4);
       /* 12345678901 */
  idxval ='          ';
  record_number = frskey (act_name_key, file_no, no_lock, idxval);
  if record_number = 0 then
      do;
      put skip list ('index file is empty');
      call menu;
      end; /* do */
  call print_data (record_number);
  do while (TRUE);
      record_number = nxtkey (act_name_key, file_no, no_lock, idxval);
      if record_number = 0 then
          do;
          put skip list ('end of file');
          goto exit;
          end; /* do */
          else
              call print_data (record_number);
      end; /* do while */
      exit:
      put skip list ('enter any key to continue ');
          get list (dummy);
          call menu;
     end list_account_name;

 delete:
      proc;
      dcl
          keyval char (11)var,
          (number, dat_rcd_nbr) fixed,
          answer char(1),
          numberx char(4) var;

      put list (CLEAR);
      put skip (4) list ('^i^iDelete Accounts');
      put skip (3);
      do while (TRUE);
          put skip list ('enter 0 to exit ');
          on error (1)
              begin;
              put skip list ('numeric input required ');
              goto oh_darn;
              end;
          oh_darn:
          put skip list ('enter number of account to be deleted :');
          get list (number);
          if number = 0 then
              call menu;
          if number > 9999 then
              goto oh_darn;   /*not a legitimate account number */
          numberx = convert (number); /*convert to string */
          put skip list('^i^iRecord to be deleted :');
          dat_rcd_nbr = getkey (act_nbr_key, file_no, no_lock, numberx);
          if dat_rcd_nbr = 0 then
              do;
              put skip list ('record number ',number,' not found/ retry');
              goto oh_darn;
              end;
              else
                  call print_data (dat_rcd_nbr);
          put skip (2) list ('^i Is this the record to delete? y/n ');
          get list (answer);
          if answer = 'y' | answer = 'Y' then
              do;
              /*
              read act_name and pass it to function to
              delete the name index record
              */
              if readat (file_no, dat_rcd_nbr, data_buf_ptr) ^= 0 then
                  call error (98);
                  else
                      do;
                      keyval = substr (act_name,11);
                      end;

              if delkey
                 (act_nbr_key, file_no, no_lock, numberx, dat_rcd_nbr)
                  ^= 1 then
                      call no_deletion(1);
              if retrec (file_no, no_lock, dat_rcd_nbr) ^= 0 then
          call no_deletion(2);
      if delkey
         (act_name_key, file_no, no_lock, keyval, dat_rcd_nbr)
          ^= 1 then
              call no_deletion(3);
      end;
  end; /* do while */

  no_deletion:
      proc(no);
          dcl
                    no fixed;

        put skip list ('deletion not sucessful at ',no);
        goto oh_darn;
        end no_deletion;
    end delete;

stats:
    proc;
    dcl
        (number_of_nbr_keys, number_of_name_keys) fixed,
        (number_of_nbr_rcds, number_of_name_rcds) fixed,
        data_file_size fixed;

    put list (CLEAR);
    put skip (6) list ('^i^i Data & Index Files Statistics');
    put skip list     ('^i^i ****************************');
    put skip (3);
    number_of_nbr_keys  = nokeys (act_nbr_key);
    number_of_name_keys = nokeys (act_name_key);
    number_of_nbr_rcds  = nmnods (act_nbr_key);
    number_of_name_rcds = nmnods (act_name_key);
    data_file_size = getdfs (file_no) - 1;
    put skip (2) list ('number of keys in act. number index file ',
        number_of_nbr_keys);
    put skip (2) list ('number of keys in act. name index file ',
        number_of_name_keys);
    put skip (2) list ('number of records in act. number file ',
        number_of_nbr_rcds);
    put skip (2) list ('number of records in act. name file ',
        number_of_name_rcds);
    put skip (2) list ('number of records in the data file ',
        data_file_size);
    put skip list ('enter any key to continue ');
    get list (dummy);
    call menu;
    end stats;


print_data:
    proc (dr_number);
    dcl
        dr_number fixed;

    if readat (file_no, dr_number, data_buf_ptr) ^= 0 then
        call error (99);
    put skip (2) list (act_name,'    ',act_no);
    put skip list (street,', ',city,', ',state,', ',zip);
    put skip list ('principal or contact, ',principal);
    return;
end print_data;


error:
    proc(location);
    dcl
        location fixed;
    put skip(3) edit
       ('^iError ', errcod(), ' at code location ', location)
          (a, f(4), a, f(3));
    stop;
    end error;



close:
    proc;
if clsdat(file_no) ^= 0 then
    call error(6);
if clsidx(act_name_key) ^= 0 then
    call error(7);
if clsidx(act_nbr_key) ^= 0 then
    call error (8);
stop;
end close;


rebuild:
    proc;
    file_no = -1;   /* auto file no assignment */
    recd_len = 128;
    file_name = 'chartdb.dat';
    name_index = 'name.idx';
    nbr_index = 'nbr.idx';
    act_name_len = 11;
    nbr_len = 4;
    name_type = 0; /*alpha key */
    nbr_type = 0;  /*numeric key entered as alpha */
    name_dup = 1;  /*add duplicate key suffix if necessary */
    nbr_dup = 0;   /*no duplicate account number suffix */
    put list (CLEAR);
    put skip (5) list ('Rebuilding corrupted data and index files');
    put skip list     ('**************************************');
    file_no = oprdat (file_no, no_lock, file_name, recd_len);
    if errcod () ^= 0 then
        call error (13);
    act_name_key = opridx
         (-1, name_index, act_name_len, name_type,name_dup);
    act_nbr_key = opridx (-1, nbr_index, nbr_len, nbr_type, nbr_dup);
    put skip (4) list ('files rebuilt - list for check of data content');
    end rebuild;


convert:
    proc (number) returns (char (4));
    dcl
        (number, position) fixed,
        (string, string_out) char (12) var;

    string = char (number);
    position = verify (string,' ');
    string_out = substr(string, position, 4);
    return (string_out);
    end convert;

end chart;
```

# LIFEBOAT HAS THE ANSWER FOR ALL POPULAR MICROCOMPUTERS*

## 8-Bit Software Available Today – New Additions Regularly

### System Tools
BUG and uBUG
DESPOOL
DISILOG
DISTEL
EDIT
EDIT-80
FILETRAN
IBM/CPM
MAC
MACRO-80
MINCE
PANEL
PASM
PLINK
PLINK II
PMATE
RAID
Reclaim
SID
TRS-80 Model II Cust. Disk
Unlock
WordMaster
XASM: 05, 09, 18, 48, 51, 65, 68, 75, 400, F8, Z8
ZAP80
ZDT
Z80 Development Package
ZSID

### Telecommunications
ASCOM
BSTAM
BSTMS
eZmail
muLink-80
RBTE-80

### Languages
ALGOL-60
APL/V80
BASIC Compiler
BASIC-80
baZic II
BD Software C Compiler
CBASIC-2
CIS COBOL (Standard)
COBOL-80
FORTRAN-80
KBASIC
JRT Pascal
muLISP/muSTAR
Nevada COBOL
Pascal/M
Pascal/MT
Pascal/M+
Pascal/Z
PL/I-80
Precision BASIC
STIFF UPPER LISP
S-BASIC
Timin FORTH
Tiny-C
Tiny-C TWO
UCSD Pascal
Whitesmiths' C Compiler
XYBASIC

### Language and Applications Tools
BASIC Utility Disk
DataStar
FABS
FABS II
Forms 2 for CIS COBOL
MAG/sam3,4
MAG/sort
M/SORT for COBOL-80
Programmer's Apprentice
PSORT
QSORT
STRING/80
STRING BIT
SuperSort
ULTRASORT II
VISAM

### Word Processing Systems and Aids
Benchmark
DocuMate/Plus
Letteright
MagicPrint
Magic Wand
Math★
MicroSpell
SMARTKEY
Spellguard
TEX
Textwriter III
WordIndex
WordStar
WordStar French
WordStar Customization Notes

### Data Management Systems
CONDOR
dBASE II
THE FORMULA
HDBS
Hoe
MAG/base1,2,3
MDBS
MicroSEED
T.I.M. III

### General Purpose Applications
CBS
Selector III-C2
Selector IV

### Mailing List Systems
Benchmark Mailing List
CBS Label Option Pak
Mailing Address
MailMerge for WordStar
NAD
Postmaster

### Financial Accounting Packages
BOSS Financial Accounting System
Financial Pkgs. (PTree)
Financial Pkgs. (SSG)
General Ledger Acctng (Univair)
GLector

### Numerical Problem-Solving Tools
Analyst
fpl
Microstat
muMATH/muSIMP
PLAN80
SigmaCalc
Statpak
T/MAKER II

### Professional And Office Aids
Apartment Mngmnt (Cornwall)
Datebook
Dental Mngmnt (Univair)
Dental Mngmnt-Family (Univair)
GrafTalk
Insurance Agency Mngmnt (Univair)
Legal Time Acctng (Univair)
Medical Mngmnt (Univair)
Medical Mngmnt-Family (Univair)
PAS 3 Medical
PAS 3 Dental
Professional Time Acctng (PTA)
Property Mngmnt Pkg. (Am. Soft.)
Property Mngmnt (PTree)
Sales Pro
Wiremaster

### Lifeboat After Hours
Backgammon/Gomoku

### Educational Tools
Torricelli Author
Torricelli Studio

### Disk Operating Systems
APPLI-CARD
BRIDOS
CP/M-80
MP/M
SB-80
SoftCard

### Hard Disk Integration Modules

*All 8-bit software requires SB-80 (or other CP/M-80 DOS) unless otherwise stated.

## NEW – 16-Bit Software Available for the IBM PC, plus. . .

### System Tools
Emulator/86
EM-80/86
PMATE-86
UT86
PANEL-86

### Telecommunications
ASCOM

### Languages
CB-86
CBASIC-86
Lattice C Compiler
muLISP/muSTAR
Pascal MT+-86
PL/I-86
PL/M

### Language and Applications Tools
PSORT
FABS PC

### Word Processing Systems And Aids
WordStar
MailMerge
MicroSpell
Spellguard

### Data Management Systems:
dBASE II
MAG/base1,2,3
T.I.M. III

### Mailing List Systems
Postmaster

### Financial Accounting Packages
General Ledger

### Numerical Problem-Solving Tools
Math PC
PLAN86
muSIMP/muMATH
SigmaCalc
Statpak
Product names are generally TMs or SMs of authors.

### Professional And Office Aids
Dental Mngmnt Sys. (8000 & 9000)
Insurance Agency
Investment Work Station
Legal Time Acctng.
Medical Mngmnt Series (8000 & 9000)

### Disk Operating Systems
CP/M-86
MP/M-86
MS-DOS (SB-86)–available for OEM license; available for end-user license for CompuPro (Godbout)

### Hardware
RAMcard

*All 16-bit software requires IBM PC DOS, SB-86, MS-DOS, or CP/M-86 as stated in Lifeboat literature.

## ALL-BIT™ Tools Available

### Books and Periodicals
APL–An Interactive Approach
Accounts Payable and Accounts Receivable-CBASIC
CBASIC User Guide
The Computer Glossary
The CP/M Handbook (with MP/M)
The C Programming Language
Crash Course in Microcomputing
Devil's DP Dictionary
Discover FORTH
DON'T (Or How To Care For Your Computer)
8080/Z80 Assembly Language Techniques For Improved Programming
Executive Computing
Fifty BASIC Exercises
General Ledger-CBASIC
Introduction to Pascal
Lifelines/The Software Magazine
Pascal User Manual and Report
The Pascal Handbook
The Pascal Primer
Payroll with Cost Accounting –CBASIC
Structured Microprocessor Programming
A User Guide To The UNIX System
Using CP/M–A Self-Teaching Guide

### Accessories
Break-Out Box
DC Data Cartridges
Diskette Drive Head Cleaning Kits
Flippy Disk Kit
Floppy Saver
MT25 RS232 Interface Breakout/Monitor
Vari Clean Cleaning Kit

***SEND FOR FULL SOFTWARE DESK REFERENCE WITH DESCRIPTIONS OF ALL THESE PLUS A WHOLE LOT MORE.***

**LIFEBOAT • 1651 Third Ave. • New York, N.Y. 10028**

**(212) 860-0300 • TWX: 710-581-2524 • Telex: 640693 (LBSOFT NYK)**

Feature

# The Fancy Font System

Charles H. Strom

## Introduction

Fancy Font is a package of programs that takes advantage of the excellent resolution available with the Epson MX-80 and MX-100 dot matrix printers. I was provided with two single density eight inch diskettes, one containing the program *PFONT.COM* and a number of pre-defined fonts to be used for printing documents, the other containing *EFONT.COM, CFONT.COM, HERSHEY.CHR* (see below) as well as a few more fonts. The software requires a minimum 48K CP/M 2.2 system, and is written in BDS C (note, however, that no source files are provided on the distribution disks.) This combination of software and hardware is capable of producing hard copy output of a quality and variety that surpasses all other printers in the price range consistent with a personal computer system. See *Figure 1* for a sample of some of the type styles and sizes supplied.

## The Epson Printer

The Epson MX-80 and MX-100 printers normally use a matrix of 9x9 dots to define a character. This is very satisfactory for draft printing, program listings, etc. but leaves a lot to be desired for letters, contracts, and other "correspondence" needs. The addition of a Graftrax or Graftrax-Plus ROM (read-only memory) upgrades the operation of the printer substantially, including among the many enhancements a graphics mode in which a resolution of 216 dots/inch vertically x 240 dots/inch horizontally is possible. This resolution is considerably higher than some of the other competing printers on the market; the Okidata Microline 82A, for example, allows a dot resolution of 66 x 60 in high resolution graphics mode. I have long thought that the graphics mode of the Epson could lend itself to the generation of custom typefaces of high resolution, but the labor required to do the job properly discouraged my efforts. The Fancy Font package has indeed admirably accomplished this task. I will proceed to discuss the three basic programs supplied; each in turn can be considered to be independent in its use.

## PFONT

PFONT.COM is the workhorse program in the Fancy Font package. It is basically a text formatter that has provision for the use of up to ten font definition files (*FILENAME.FON* files) in a document. These fonts may be chosen from the pre-defined sets supplied on the second Fancy Font distribution disk or those created by the user. More on the latter when we get to a discussion of the EFONT and CFONT programs.

Parameters may be specified either on the command line or interactively; thus:

```
A>PFONT TEXT.FIL +FO B:ROMN12 +SP 1
```

is equivalent to:

```
A>PFONT
>>TEXT.FIL +FO B:ROMN12 +SP 1
>>
```

These commands each will print the file *TEXT.FIL*, using the font file *ROMN12.FON* on the B: drive, which is a 12 point high Roman type style. (A point is 1/72", and "normal" type is usually 10 to 12 points, unrelated to the familiar 10 and 12 pitch typewriter print size.) The +SP 1 switch will force a line spacing of one inch between lines.

There are close to thirty parameters such as SP that can be used to format output when using PFONT, including functions to control the font file names, margin size, formfeed handling, subset of pages to print, pause between pages, etc. Fortunately most of these have default settings which can be used for "normal" documents until the user has familiarized himself with the intricacies of the program. The program achieves a satisfactory degree of user-friendliness in that entry of a "?" in response to a PFONT prompt will display relevent help information.

In addition to the PFONT parameters, there are a number of switches which can be imbedded in the text file to be interpreted by PFONT during hardcopy output. All of these formatting indicators are preceded by a backslash character (\) as a default or any other user specified character as a signal to PFONT that an indicator follows. Thus one must prepare a document with these switches in mind. It is not possible to use all of the features in WordStar, for example, to format a document and then use PFONT to print it. On the contrary, PFONT cannot recognize formatting commands from other formatters and unpredictable results would follow. In fact use of the WordStar document mode is only possible if the soft carriage returns and any other characters with the high bit set are filtered with the use of PIP with the Z switch or one of the public domain programs such as Keith Petersen's *FILTER.COM*. It is therefore easier to use a simple-minded editor such as Cromemco's screen editor to enter text destined for Fancy Font.

As mentioned above, up to ten fonts may be specified in a document. These are indicated by \f1, \f2, etc., in the text file and by specification when executing PFONT of which font name corresponds to which font number in the document. Other switches are available for text centering, underlining, justification, tabbing, etc. There is also provision for printing a character corresponding to a particular ASCII value in the font. This allows the printing of special characters, symbols, etc. This will be discussed more fully in the section dealing with CFONT. Thus, PFONT supplies a versatile print formatting capability tailored to the characteristics of the Epson printer. There are few if any features one could ask for that are not available under PFONT.

(continued on next page)

## EFONT

EFONT.COM allows the editing of a font. It is supplied in two forms on the distribution disk, both in a normal file and in a format that uses overlays (parts of the program reside in different disk files called up as needed.) EFONTO.COM, the latter version, is required for smaller TPA sizes and sacrifices some speed for compactness. The purpose of the edit program is to convert a font into a set of characters suitable for modification using a standard text editor. The normal definition of a character as a series of dots is converted to a series of asterisk characters in the text file. In addition to the asterisk representation, left and right margin spacing for the character (blank dot positions) and the relative height of the top dot in the character are represented. This latter information is used so that character baselines line up properly when using several fonts in a single document.

The commands present in EFONT allow the loading of a font for editing or inspection, printing one or more characters to the Epson in the expanded asterisk representation, modifying margins of one or more characters in the font, replacing one or a group of characters with their edit file specifications, etc. The use of this program required a little trial and error to understand the effect of the various commands; it is not as straightforward to use as PFONT. I was also hindered by the preliminary version of the manual that I was dealing with, but the finalized copy that I have just received goes into each command in sufficient detail to make me comfortable. I have my doubts that a non-technical person would be able to use EFONT (or CFONT for that matter) without considerable grief; this is not true for PFONT, however, and considering the large variety of fonts supplied with the software, I do not see this as a serious handicap.

## CFONT

CFONT.COM is surely one of the most useful and enjoyable aspects of Fancy Font. This program is based on the National Bureau of Standards publication No. 424, authored by Alan V. Hershey. The *Hershey database* consists of over 1500 character definitions of all sorts, including "normal" characters, italics, greek letters, mathematical symbols, Old English, game symbols, etc. It is from this database that the bulk of the fonts on the distribution disk were drawn. CFONT allows the user to define new fonts using the Hershey database as a pool on which to draw. The versatility of the program allows the mapping of any Hershey character to any ASCII value; thus one could easily define a set of greek symbols with "A" corresponding to alpha, "B" to beta, etc. One could even define an "A" in a text file to be printed as a "Z". I suppose one might define a font with random correspondence between the actual characters and those printed out with Fancy Font to be used as a simple cryptographic tool, providing the receiver had the same font definition available.

There are several commands callable in CFONT, to do the mappings from the Hershey set to the font, change scaling factors (character size), print a set of characters to the printer for inspection (this command did not work properly in my version), save the mappings as a FONTNAME.FON file, etc. Presumably one would define a suitable character set using CFONT, print the characters and after inspection, use EFONT to make small changes so as to improve the appearance of any characters that looked a little odd, and then save the font for future use. While not inherently difficult, this is a tedious process. I suspect that most users will be satisfied to use the supplied fonts, adding perhaps a character or two now and then as the need arises. This is in fact how I have used Fancy Font in the two months or so that I have had it. It is an important plus, however, to have the tools available to define complete sets of characters from scratch using either the Hershey Set or a text editor. The latter capability could allow the creation of custom symbols for letterhead, for example.

## Kinks

All is not roses, unfortunately. The biggest drawback to Fancy Font is the speed of printing. The normal mode instructs the Epson printer to make nine passes for each line of text for a typical 12 point font. Since we are using graphics mode, these passes are made unidirectionally. The resulting print speed is only about six lines/minute! Needless to say, one needs a lot of patience. The *RD* - (rough draft) switch is available to optionally specify fewer printhead passes for trial documents and makes life a little more bearable. There is also an option to use the hardwired Epson fonts, but many of the PFONT features are of necessity not available under this mode (justification and centering for example) because of the hardware restrictions when not in dot-graphics mode. There is also a mode to preview the output on the console, though this is obviously of limited utility for the same reasons. Even the use of a hardware spooler such as the Queue-IV with a 128K buffer is only a partial solution since a line of text is translated into many hundreds of bytes of data to be output to the printer when in high resolution graphics mode. These are the tradeoffs one must endure in order to get print quality similar to a Sanders printer at a fraction of the cost. I think this is a reasonable price to pay for the enjoyment of seeing such pretty type, though I cannot see running assembly language listings, for example, through the Fancy Font process.

## Summary

I presume that it is pretty obvious from this review that I am thoroughly pleased with the Fancy Font package of programs. Though I cannot in all honesty state that I am using it for all of my daily correspondence needs, it is a masterful job of programming that fully captures the capabilities of the Epson hardware in a most user-friendly manner and it represents microcomputer applications programming at its best. The current price of Fancy Font is $180 and it is available in 8" single density CP/M standard format as well as Apple, Xerox, and Osborne formats. Contact the vendor for information on availablility on other formats. Fancy Font is available from Softcraft, 8726 S. Sepulveda Blvd., Los Angeles, Ca. 90045, 213-641-3822.

Sample Fonts

This is Roman, 12 point, 18 point, 8 point

*Italics are also possible with the included fonts*

**Roman bold is another font useful for everyday documents**

This is a sample of a Sans Serif style in 11 point

**My favorite demonstration is to show people the**

# Old English font

# Feature

# Crosstalk

Davis A. Foulger

There is a lot to like about CROSSTALK, a communications program developed by MicroStuf, Inc. for microcomputers running under CP/M-80, CP/M-86 and SB-86. The documentation is excellent. The procedures for backing up and initializing the program are extremely easy. The program runs smoothly — more smoothly, in fact, than any other microcommunications program I have looked at on the IBM PC. Indeed, the program meets just about all of the basic criteria for a usable microcommunications program (as outlined in my article in the July *Lifelines/The Software Magazine).*

However, it is a rare package that couldn't use some improvement. I can recommend the program with great confidence to microcommunicators who want to use a single package and who don't want to have to think very much about how it works.

## A Little Background

CROSSTALK has come to the IBM Personal Computer from the CP/M-80 world. MicroStuf describes it as a Smart Terminal And File Transfer Program, a description which sums up the program about as well as any available.

Although the program offers the user about 39 commands, it provides little in the way of "'ideal" (see my article in the August *Lifelines/The Software Magazine*) features. Rather, it turns the microcomputer into a "'smart" terminal — smart largely because it makes use of the microcomputer's disk drives to store and retrieve information.

CROSSTALK takes advantage of the special features of the Hayes Smartmodem. When I received CROSSTALK for review, this was an unusual feature of the package that should have made the package particularly appealing to IBM Personal Computer owners, as the Hayes Smartmodem has proven a popular peripheral for IBM PC's.

## A Session With CROSSTALK

CROSSTALK is not difficult to use. Easily copied onto the diskette of one's choice, the program can be kept on a dedicated diskette with transcripts of conversations, or kept as one of several utility programs in a DOS toolkit. The program does not require a run time package. As a result, there is no need to call up BASIC or any other language before using CROSSTALK.

The XTALK command invokes the program; this feature can be altered by changing the name of the program file. When the program is fully loaded, a screen is displayed that shows CROSSTALK's current settings and lets the user start up the system. The default version of this screen is shown in Figure One.

For the user who knows how to read it, this screen provides ample information about system status. The display is cryptic, however, and does not seem to get any easier to read with time. The meaning of each of these settings is summarized briefly in Table 1, where the 23 commands of Table 1 are displayed along with the sixteen other commands that are supported by the system. (Incidentally, CROSSTALK users can utilize Table 1 as a 'quick reference', not provided in the documentation.)

Cryptic informativeness seems to be a consistent feature of CROSSTALK's user interface. All the information needed to make CROSSTALK user-friendly has been implemented on the system, yet the package is not really all that friendly. To run CROSSTALK you must read and understand the CROSSTALK manual, especially if you use CROSSTALK with a Hayes Smartmodem.

The problem is presenting information effectively. At some points CROSSTALK offers the user an overload of poorly organized information — the system parameters listed at the beginning of the program are a good example. At other times, there seems to be no information at all. CROSSTALK is not a menu-driven program, and while there are prompts, they are not adequate.

## Getting Help On Getting Help

At the beginning of the program the user is advised to 'enter commands — or "he" for help'.

Ideally, the HElp command should lead to a menu of options, one being an explanation of how to start microcommunicating with CROSSTALK. If you don't know how to use HElp, all you see after entering the HE command is a rather busy table of the commands available for use in CROSSTALK.

The commands are only minimally organized in the table. No hint is given about what the commands are, how to use them, or how to use the HElp command to find out what they are. Plainly, this first HElp screen could be written more helpfully.

The HElp command would most benefit from a menu-driven approach that guides the user through the various HElp options. Indeed, a prompt and menu approach should probably be applied to CROSSTALK rather liberally. This is particularly easy to implement on the IBM Personal Computer, where line 25 of the display has been given over almost entirely for use as a prompt and menu area.

A line twenty-five display of menus and prompts would be particularly nice given the program's use of the IBM PC's function keys (F1 to F10; see Table 1). The assignment of four of those keys — F1 to F4 — as user programmable keys is a particularly nice feature of CROSSTALK.

## Dialing Out

Once you know the command structure (or have figured out how to use HElp) it is easy to call another computer using CROSSTALK. If you are using a Hayes Smartmodem, the telephone number can be dialed directly from the keyboard of your computer. If you aren't, a single carriage return will take you into terminal mode. The user has fairly complete control of the configuration of the terminal and transmission characteristics, as can be seen in Table 1.

Commonly used configurations and the telephone numbers they are used with can be saved for reuse in command files. When Smartmodem is used, these command files allow you to dial the phone by simply specifying the command file involved when CROSSTALK is first started. To do this you might write XTALK SOURCE (assuming a command file called SOURCE) to start the program. The command file would be called and the number dialed automatically.

The user is also given two sets of commands for exchanging files with other computers. One set, the CROSSTALK Protocol commands, allows the reliable exchange of files between computers when both are using CROSSTALK. Protocol transfers move files between computers one block at a time, with sophisticated error checking performed along the way. Non-protocol file transfers will work when a connection is established with any computer.

Once the connection is established, movement between communications (terminal) mode and CROSSTALK's command mode is quick and easy. The Escape key moves the user into command mode. The return key takes the user back into communications. Some of CROSSTALK's commands allow a remote user to control the operation of CROSSTALK. The program is not an Electronic Bulletin Board program and will not run in an entirely stand-alone mode, but files can be saved and retrieved from a remote terminal without the intervention of the local operator.

## CROSSTALK and the Hayes Smartmodem

CROSSTALK's ability to take advantage of the Hayes Smartmodem should be one of its strongest features. The Hayes product does not require special software and the user should be able to control it from any microcommunications program. Operating the Hayes does require the user to learn a small program language, however. Thus, controlling the Smartmodem from software opens the possibility of building a much more user friendly microcommunications interface.

CROSSTALK's advantage should be made even greater by the fact that, despite the relative simplicity of adapting software to Smartmodem, very few microcommunications software packages do so. Indeed, some software developers I have spoken with are openly hostile to the idea of building Hayes' interfaces into their software.

This advantage has been nullified, however, by two features of CROSSTALK's support of Smartmodem. First, CROSSTALK requires that the Smartmodem be configured with non-standard switch settings that require the user to take the cover off Smartmodem and physically reset the Hayes switches. Second, CROSSTALK then takes complete control of Smartmodem, refusing to let the user control Smartmodem in any way.

The non-standard switch settings present CROSSTALK with a kind of double problem. For inexperienced users the resetting of Smartmodem's switches may represent a formidable task. Although resetting these switches is not difficult, it is not well-documented either.

CROSSTALK leaves its explanation of the resetting operation to Appendix F, where the user is faced with nothing more than a numbered list of the appropriate switch settings. Reference to the Smartmodem manual helps (pages 2-4 to 2-7), but a person unfamiliar with computer equipment (and deathly afraid of damaging a several hundred dollar investment) may require several hours to figure out how to do it.

The second problem with the non-standard switch settings will be experienced by users (like myself) who employ several different microcommunications programs; many programs will be unable to utilize Smartmodem in the CROSSTALK switch configuration. In testing CROSSTALK I found it necessary to change switch settings so often, at times, that I took to simply leaving the front cover of the Smartmodem off.

Users who never use any program but CROSSTALK will not find this to be much of a problem. For me, however, CROSSTALK's lack of word wrap and other "ideal" microcommunications software options make it a poor choice for some frequently used microcommunications applications.

MicroStuf's decision to tie up Smartmodem with non-standard switch settings was neither capricious nor designed to frustrate users. The choice is made because CROSSTALK does a very good job of controlling communication itself. It does not, for instance, need the Smartmodem to formally recognize incoming calls. It is quite capable of reading the telephone lines itself if put into answer mode. There are some very good reasons to give a microcommunications program the kind of control that MicroStuf has built into CROSSTALK, especially when the user's microcommunications system works with one of the less expensive direct connect modems. That control is wasted on Smartmodem, however, which loses capabilities under CROSSTALK's control.

Both of the objections I have raised to CROSSTALK's use of the Smartmodem can be met in software, however. First, it would be possible for CROSSTALK to make the switch settings electronically all by itself, once it knows it is dealing with a Hayes Smartmodem. This would greatly benefit the user who makes use of other microcommunications software packages, as it would allow Smartmodem to be reset to the generally preferable factory settings by simply turning the modem off. User access to the Smartmodem can also be affected from within software.

## Documentation

The documentation lacks a quick reference card, a loss that is made particularly acute by the software's shortcomings in the area of menus and prompts. It also lacks

(continued on next page)

the kind of clear conceptual organization that would be gained in the exercise of creating a quick reference card.

The documentation's biggest problems are in its attention to little things. The process of resetting the switches on the Smartmodem, for instance, should be clearly explained in the manual. Indeed, much more should be made of the need to reset those switches.

The manual suffers from inattention to some big things as well, particularly from MicroStuf's failure to clearly organize the commands and explain their relationship to one another. On the whole, CROSSTALK's documentation isn't bad. But it would benefit from some reorganization and lengthening.

## Overall Evaluation

CROSSTALK is a good microcommunications software package that many will find highly satisfactory. As already noted, there is a lot to like about CROSSTALK. The package would, however, benefit from some improvement, particularly in the area of making the program friendlier through the use of prompts and menus. The documentation could be better, too, and would be much improved by the inclusion of a quick reference card. Finally, the package's interface with the Hayes Smartmodem could be handled better.

My complaints here are really aimed more at the programmers at MicroStuf than at readers. Readers should, of course, know about the program's shortcomings before putting down hard currency for it, but the potential for improvements to the program seem more important, especially now, while the market for microcommunications is still young and the number of truly inexperienced users is still limited. I don't expect to make much use of CROSSTALK right now, but I like the underlying package enough to think that it will be much improved in future releases. I look forward to reviewing, and using, those releases.

**Figure 1 — CROSSTALK's Default Settings**

Current Parameters:

NAme

NUmber

Time : 8/20/82 01:06:55

| | | | |
|---|---|---|---|
| MOde | Originate | ATten | ESC |
| DUplex | Full | CApture | Off |
| SPeed | 300 | PRinter | Off |
| PArity | None | FIlter | On |
| Data | 8 | DEbug | Off |
| STop | 1 | LFauto | Off |
| WAit | 01h | SCreen | Off |
| BLock | 01h | TAbex | Off |
| FLow | Char | UConly | Off |
| WIdth | 1 | Command | ↑C |

enter command — or "he" for help

**TABLE 1**
**CROSSTALK's Commands**

**NON-Protocol file transfer commands (work with any comm program)**

| | |
|---|---|
| CApture | (F5) save received data to a buffer or disk file |
| WRite | save receive buffer to disk file |
| MEm | draws a graph of the capture buffer |
| REad | Send a file to the modem |
| SCreen | don't send line feeds when reading a file from disk |
| FLow | sets CROSSTALK to transmit by line or character. |
| WAit | sets waiting times in line or character flow modes |

**Protocol File Transfer Commands**

| | |
|---|---|
| XMit | Initiate protocol transfer of files |
| RCve | Expect to receive a protocol file transfer |
| RQuest | request file from remote system |
| BLock | sets size of data blocks (in 128K increments) |
| NO | no more files are coming |

**Command File Commands**

| | |
|---|---|
| LOad | loads and executes command file |
| NAme | a convenience feature for identifying command files |
| SAve | save a command file |
| LIst | (F8) list current configuration |
| NUmber | sets number to be dialed |

**Convenience Commands**

| | |
|---|---|
| DIr | displays disk directory |
| HElp | gets help |
| TIme | (F9) Displays time and date on screen |
| TYpe | (F7) Displays contents of receive buffer |

**Transmission Control Commands**

| | |
|---|---|
| SPeed | set transmission speed |
| STop | set number of stop bits |
| DUplex | sets duplex to full or half |
| MOde | sets modem to answer or originate |
| DAta | Sets the number of data bits used |
| PArity | odd, even, none |

**Terminal Control Commands**

| | |
|---|---|
| ATtention | sets local command attention character |
| COmmand | sets remote command attention character |
| WIdth | sets video display width at 40 or 80 |
| PRinter | (F6) turn printer on or off |
| BYe | hangs up and permits you to make another call |
| XDos | exits to DOS without hanging up |
| QUit | hangs up and exits to DOS |

## TABLE 1
## CROSSTALK's Commands (continued)

**Character Control Commands**

| | |
|---|---|
| DEbug | Identifies control characters |
| FIlter | sets CROSSTALK to discard received control characters |
| TAbex | Transmit tab characters as spaces to the next tab |
| UConly | Converts lower case to upper case |
| LFauto | add line feed after received carriage returns |

**Additional Uses of Function Keys**

| | |
|---|---|
| F1 ... F4 | User definable messages |
| F10 | Sends a Break signal to the remote computer |

## TABLE 2
## Facts & Figures

**Package:**
CROSSTALK 1.02

**Price:**
$195

**Systems Available For:**
IBM Personal Computer, CP/M

**Machine Requirements:**
64K, one disk drive, and a modem

**Protocols Supported:**
Asynchronous up to 9699 Baud; Hayes Smartmodem

**User Interface:**
Serial

**File Transfers:**
Both Transmission of Saved files and saving of received files. Two transfer modes are available. One will work when communications is established with any computer. The second, which provides special error checking and transmission control mechanisms, only works when CROSSTALK is used on both ends of the line.

**Special Features:**
None

**User Skill Level Required:**
Requires a reading of the user manual and, if the Hayes Smartmodem is used, a reading of its manual. Relatively easy to use, once that reading is done and the modem is properly configured. This may prove difficult going for the novice user, however.

**System Upgrade Policy:**
Users are notified of upgrades, when available.

## TABLE 3
## Qualitative Factors

| | Rating |
|---|---|
| **Documentation** | 4 |
| organization for learning | 3 |
| organization for reference | 2 |
| readability | 5 |
| includes all needed information | 4 |
| **Ease of Use** | 4 |
| initial start up | 2 |
| operator use | 5 |
| setting terminal parameters | 6 |
| **Microcommunications Features** | |
| terminal Control | 6 |
| transmission control | 7 |
| operation under program control | 1 |
| storing terminal parameters | 7 |
| automatic log-on features | 5 |
| **Support** | |
| for initial start up | 4 |
| for system improvement | 7 |

*Ratings in this table wil be in a 1-7 scale where:
1 = clearly unacceptable for normal use
4 = good enough to serve for most situations
7 = excellent, powerful, or very easy depending on the category

Feature

# Fullscreen Program Editors for CP/M

Ward Christensen & Tom Cochran

## WRAP-UP

**by Ward Christensen**

Here is a look back after nearly a year of testing various full screen editors. I often am asked: "which one did you like best?". There was no single one. I find I use the editors I have in approximately the following proportions:

Program editing:

| | |
|---|---|
| PMATE | 80% |
| MINCE | 15% |
| WordMaster | 5% |

Text editing:

| | |
|---|---|
| PMATE | 83% |
| WordStar | 10% |
| MINCE | 5% |
| WordMaster | 2% |

The editors have several things in common: All do preemptive scrolling, i.e. when you press a key, whatever scrolling was in progress is aborted if the key pressed would dictate that. Thus issuing four "scroll up" keys in rapid succession, will quickly show the final screen and skip the intermediate ones. Some of the other editors I reviewed didn't have this basic usability feature.

With the exception of WordMaster, all are very configurable. I have changed *all* of them to have nearly the same keyboard layout — at least for the major cursor movement, scrolling, and line editing keys. They differ only in the area of less frequently used functions: top of file, bottom of file, etc.

PMATE: The characteristics that earned it top slot, are: (1) good full screen processing, making use of hardware line insert-delete, erase to end of line, scrolling, etc; (2) the ability to do some text work, i.e. it has word-wrap and limited printing; (3) superb customizability so that every couple of months I add or change some features; (4) a powerful command macro language. Mike Olfe's column in *Lifelines* gives many examples of the abilities of PMATE macros. My own most recent macro adds columns of numbers in a text file — typical of the *very* generalized applications macros may be put to.

MINCE: A good general-purpose editor. I wouldn't want to be without it because of its multi-file and split-screen options.

WORDMASTER: It is still the fastest when it comes to making small changes, or when making multiple identical changes through a long file. Its lack of an "undo" key and lack of block tag-and-move (or tag-and-delete) limit my productivity with it as a general editor.

WORDSTAR: I use this as a fancy print program. I use it to print PMATE text files since PMATE's and WordStar's word wrapped lines are compatible.

## Revisions

In the last several weeks, I have received revisions of VEDIT, MR EDit, and PMATE. They all show improvements, although it appears that PMATE was the only one specifically enhanced in response to the comments I made in my *Lifelines* reviews — and I liked it to start with! (A review of the new PMATE follows this article.)

A quick installation of the new VEDIT showed the lack of scrolling interrupting had not been fixed, so I didn't look further. Similarly, the Mr EDit cover letter indicated this release had "minor improvements" over the previous release. Thus I was able to save my time by not even looking at it, what with bandages not being sufficient when heart surgery was called for.

## PMATE Revisited

**New Release, nice features**

**by Ward Christensen and Tom Cochran**

PMATE for CP/M-80 by Michael Aronson of Aox Incorporated, and sold by Lifeboat, has turned out to be my favorite editor. I reviewed version 3.02 in the May '82 issue of Lifelines. The newly released version 3.21 directly addresses many of the comments I made in that review.

The major enhancement is a full-featured repeat key. Like WordMaster, pressing the designated repeat key puts 4 in a counter. Subsequently pressing it multiplies the counter by 4. Thus repeat-repeat-cursor down will move the cursor down 16 lines. Like the more powerful MINCE repeat key, PMATE will also accept decimal digits after the initial repeat key press. Thus repeat-1-8 will put 18 in the repeat count, i.e. repeat-1-8-* will place a row of 18 asterisks in your file. Unlike MINCE, PMATE does not *show* you the repeat value. However, since the top line of the screen does have a field called "ARG', I could see (hint hint) a future release placing the repeat value in that field.

I originally complained that single-line scrolling — the ability to move the screen up or down one line without moving the cursor — was not supported. The alternative was to either scroll a page, or set "wander = 0" so the cursor can't wander off of the center line of your screen, thus indirectly scrolling a line at a time when you moved up or down. Release 3.21 added a line scrolling feature, so now if you set wander to something larger than zero, the cursor moves within that space without scrolling. A major change in PMATE is the means of adding your own instant commands. These are functions that you want to do with one or two keystrokes, but that were not pre-programmed into PMATE. For example, standard PMATE implements the line delete key as "erase to end of line if you are not at the beginning of the line, otherwise delete the entire line". If I had started with this, I would probably be quite comfortable with it. However, WordMaster got me used to separate "erase to end of

line" and "delete entire line" keys, so I am able to extend PMATE to support them.

The process used to consist of making modifications via an assembly source program. Now the number of such "user instant commands" has been cut back a bit, but you can place them into PMATE via the *editor* in full screen mode.

People who bought PMATE for the IBM PC after reading my review, were occasionally sorry they couldn't customize theirs to the degree I could the CP/M-80 version. Since this new manual refers to *all* versions of PMATE: CP/M-80, CP/M-86, MS DOS, or PC DOS, this new method of adding user instant commands should simplify user enhancements to these other versions.

Among the other improvements: many useful macros are now included with the disk, thanks to Mike Olfe of Lifeboat; scroll left and right features are supported; structured programming (and even assembler programming) is improved by addition of auto-indent, indent set, and indent increment or decrement commands; the full screen width line of "---" (line 3 of your display) is no longer redrawn when macros are executing, thus speeding up execution;

Tom Cochran (who also owned version 3.02) and I reviewed version 3.21, and came up with a few things we would still like to see:

(1) Scrolling is nice, but it doesn't continue after the cursor reaches the top or bottom of the screen. Thus you have to switch from "line scrolling" keys to "cursor movement" keys if you wish to scroll off the page. We don't think the cursor being at the top line (actually, the top limit of "wander") means we might not want to scroll up some more. This is a minor problem, and can probably be easily corrected;

(2) While not drawing the line of dashes improves speed during the execution of macros that display the screen, the entire status line is still updated, even though it doesn't change.

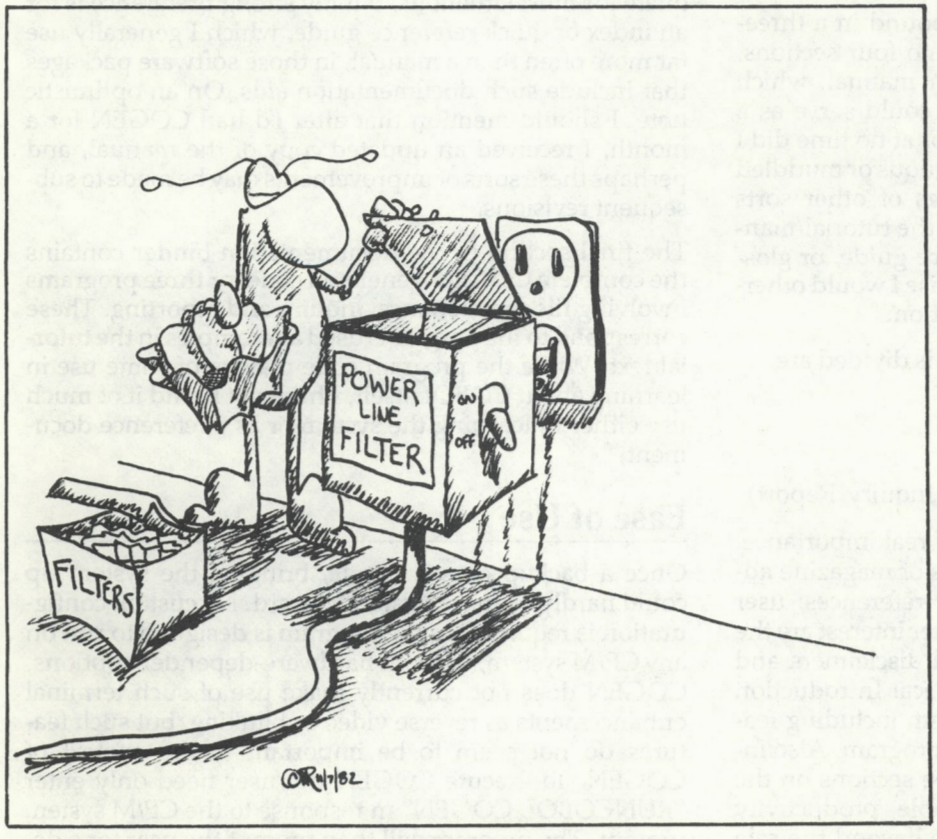
(3) If as required by some terminals, you have specified that data may not be placed in the last column of the last row of your screen, PMATE keeps addressing the cursor down there (actually on every keystroke). If your terminal uses a large block cursor, this is somewhat distracting. Similarly, the cursor flashes to update the column number on every keystroke. It would be nicer if it *never* went to the lower right corner, and if the column was only updated with a "¼-second delay" so if you were typing fast, it wouldn't keep jumping around.

(4) Under the new revision, the number of user instant commands appears to be limited to ten. A "real hacker" would like more than that.

(5) It would occasionally be nice to be able to horizontally scroll a file wider than 250 — (a limited, special need).

(6) A nice "frill" would be to have an indicator to show which buffers are in use, i.e., a line: T0123456789 being shown in the last 11 bytes of the line of dashes, or in the open space on that status line.

None of these significantly detract from the usability of PMATE. All in all, PMATE was very good, and the new revision makes it even better. ■




(Editorial continued from page 2)

Finally Allan Miller has produced an interesting book on CP/M called *Mastering CP/M*. It attempts to address the issue of writing a BIOS, discusses building a MACRO library, using BDOS for nondisk operations, reading/writing files with BDOS, and other less interesting topics covered countless times before. Ask for it at your local dealer or bookstore and if your examination warrants it, add it to your CP/M library. This is too technical a treatment to be of much interest to the casual reader/user. Unquestionably the best CP/M book to date is *Inside CP/M – A Guide for Users and Programmers with CP/M-86 and MP/M2*. This book, authored by David E. Cortesi, is a must for all serious CP/M users. If you can only afford one book on CP/M for your library this is the one. Beginners should read *CP/M Primer* by Stephen M. Murtha and Mitchell Waite.

These books are the best of what's available so look at Allan's book, read Murtha and Waite's treatment and get Cortesi's tome. You can forget the rest... The Best Is Yet To Be... ■

Feature

# Cogen – An Application Development Utility

Joseph Rothstein

COGEN (from Bytek, 1714 Solano Avenue, Berkeley, CA 94707) is an applications development utility which produces RM/COBOL program code for standard business applications: file maintenance, inquiries and reports. Through a series of steps based on menu selections and specifications, entered by the user in response to prompts, the utility produces copy files which can be incorporated into other COBOL source code, or combined by COGEN into complete applications programs.

COBOL is ideally suited to such a utility for two reasons: first, while not considered a "structured language" by many purists, COBOL nonetheless has a clearly defined program hierarchy and organization, employing certain segments of code which often vary little from application to application. Second, by intention of its designers, it is a business-oriented language. Consequently, a greater percentage of COBOL programs are of the variety at which COGEN excels than would be the case with a language such as BASIC or Pascal.

For this evaluation I used a homebrew S-100 system running CP/M, and though COBOL is not a language I use frequently, I was able to create useful, complete programs with a speed and facility I found impressive.

## Evaluation

**Documentation.** The documention is bound in a three-ring notebook divided by index tabs into four sections. The clarity of the tutorial text and user manual, which forms the bulk of the documentation, could serve as a model for all documentation authors, for at no time did I find myself confused or misled by ambiguous or muddled writing. However, several shortcomings of other sorts should be noted. Despite the quality of the tutorial/manual, the lack of an index, quick reference guide, or glossary of important terms tempers the praise I would otherwise lavish upon COGEN's documentation.

The four sections into which the binder is divided are:

1) Literature - Comments
2) Demo - Installation
3) User's Manual
4) Sample Programs (Maintenance, Inquiry, Report)

The Literature portion includes little of real importance. Most of the section is devoted to copies of magazine advertisements, news releases, customer references, user comments and such. Of somewhat greater interest are the User Agreement, containing the usual disclaimers and limits on liability, and a 13-page Technical Introduction containing a brief overview of the system including features, primary modules, and a sample program. Also included in the Technical Introduction are sections on the benefits of using COGEN and sample productivity benchmarks. These seem of little value beyond the sale stage, but may be of some interest to the novice user trying to determine what to expect of the package.

The Demo-Installation section consists of a single page, titled "Installation and Execution under CP/M". It describes the minimum configuration requirements, a brief admonition to copy the contents of the distribution disk to a working copy, and a few sentences on execution. While this may be entirely adequate for the experienced user of CP/M, the novice user generally needs a more complete and strongly worded backup procedure. No terminal- dependent features are described, and apparently none are part of this version of COGEN, though there are plans to include such features in subsequent releases.

It is in the User's Manual that COGEN truly shines. The 55-page combination tutorial/reference is a joy to read and a pleasure to learn from. Tutorial examples appear on left-hand pages, corresponding to reference manual entries on right-hand pages. Sample entries by the user are indicated in the tutorial by boldface characters enclosed in boxes.

Once beyond the tutorial stage, though, the experienced user may find the organization of the reference guide to be less than optimal. While all the needed information appears to be present, the only way to locate a particular subject is through the table of contents. This is probably adequate for most situations, but my strong preference is for an index or quick reference guide, which I generally use far more often than a manual, in those software packages that include such documentation aids. On an optimistic note, I should mention that after I'd had COGEN for a month, I received an updated copy of the manual, and perhaps these sorts of improvements may be made to subsequent revisions.

The final section of the documentation binder contains the complete COGEN-generated code for three programs involving file maintenance, inquiry and reporting. These correspond to the programs used as examples in the tutorial text. While the program code may be of some use in learning about COBOL itself, I have not found it of much use either in learning the system or as a reference document.

## Ease of Use

Once a backup copy is made, bringing the system up could hardly be more straightforward. No custom configuration is required, as the program is designed to run on any CP/M system, without hardware-dependent options. COGEN does not currently make use of such terminal enhancements as reverse video or blinking, but such features do not seem to be important in the context of COGEN. To execute COGEN the user need only enter "RUNCOBOL COGEN" in response to the CP/M system prompt. The program will then prompt the user for a de-

fault output drive, which may be the currently logged disk or any system disk drive.

COGEN uses menus and data entry screens to lead the user through the development of an application program. Four modules appear on the master menu: Files, Screens, Reports, and Programs. The Files module uses interactive data entry to construct the file select, description, and declaratives sections required of any COBOL program. The Screens module enables the programmer to design screens for use in COGEN programs or for incorporation into independently coded procedures written in COBOL. The Reports module enables the programmer to define report form blanks on the screen in an interactive fashion. Finally, the Programs module combines COGEN copy files into complete programs for either file maintenance, inquiries, or reports.

Code generated by COGEN may also be incorporated into other programs written in any version of ANSI COBOL-74 which includes the 'COPY' statement, though code created by COGEN to handle screen I/O may not necessarily be compatible with COBOL's other than RM/COBOL, since screen handling protocol is not part of the ANSI specifications.

Each of COGEN's modules deserves high marks for ease of use, with one caveat. Some knowledge of COBOL is required if the user is to understand what specification is required by the interactive dialogues. The user need not already be an expert COBOL programmer, but one who is new to COBOL should probably at least read an introductory text on the language before attempting to use COGEN. This is not a shortcoming of COGEN, which does not claim to be designed for the completely inexperienced user. Anyone with even a passing familiarity with COBOL should find this product a pleasure to use.

Only those field descriptors found in standard COBOL are included, so data entry error trapping is somewhat limited, though this is a result of COBOL itself rather than the application generator. For the same reason, there is no facility for creating online "Help Messages" to the user. Data entry is as idiot-proof as it is in any COBOL program – no more, no less. However, such features as screen overlays and split-screens provide facilities for powerful and flexible, if not necessarily error-free data entry.

The skilled COBOL programmer should be able to incorporate data files generated under other languages or programs into a COGEN-created program, provided that the data files can be described according to COBOL's rules for data file and record definition.

No file recovery utility is included with the package, making program and data backup completely the responsibility of the user.

File access, both in retrieving records and adding new ones, is fast and efficient. Whether the programmer specifies dynamic, random, or sequential access, COGEN programs can retrieve or insert one record of 1000 in just a few seconds. Dynamic record access by record key appears to be the fastest access method, but each access method acts within acceptable time limits – though the I/O speed of any particular system will depend on such factors as clock speed and disk density.

In general, while COBOL is not recognized for implementing facilities designed to make life easy on the end user – and COGEN is no different – COGEN itself deserves considerable credit for offering the COBOL programmer a tremendous improvement in the ease with which an application program may be developed.

## Support

The level of support and technical advice BYTEK offers COGEN users seems consistent with the professional standards evident in most other aspects of the product. BYTEK's telephone number is published (though not prominently) in their documentation. The company maintains a technical staff prepared to assist users of the package, and they are most accommodating in offering such assistance.

I spoke by phone with BYTEK's Dan Pines, author of the sample programs found in the manual, and found him to be most knowledgeable and helpful. I was particularly impressed by his statement that registered COGEN owners will continue to receive free updates to the package into the foreseeable future. This is a refreshing change from the usual practice among microcomputer software distributors (where $50 update fees are not uncommon), and goes a long way toward justifying a price which may, at first, seem expensive. I would gladly pay a higher initial purchase price in return for the sort of quality product, level of support, and update policy which BYTEK seems to provide COGEN users. As I mentioned earlier in this review, BYTEK sent me an updated version of their user manual shortly after I had received the original one; a disk containing their latest revision of the software, Version 5.2, was included as well. Enhancements and revisions appear to be made on a regular basis, and long-term product support seems assured.

## Conclusions

Stated briefly, any COBOL programming shop or professional COBOL programmer who does not have a system like COGEN should buy it without delay.

The hacker, software collector, or hobbyist will likely balk at COGEN's $950 price tag – steep by comparison to most microcomputer software. But to those for whom COBOL programming is a bread-and-butter pursuit, the package will quickly pay for itself in the coding time it saves. Programmer time is often the biggest single expense of a software enterprise, and COGEN has the potential to make a great contribution to increased productivity and cost-cutting in that regard.

On the other hand, COGEN does not itself seem an compelling reason to switch from another language to COBOL. No program generator can, alone, overcome the shortcomings of a programming language, and it's my opinion that COBOL has many shortcomings. That's no reflection on the quality or usefulness of COGEN, however – it is a professionally designed and implemented product, which should find receptive users among the COBOL community.

Moreover, I am pleased to see the emergence of program generators such as COGEN. We seem to be entering a new stage of software development, in which high-caliber

(continued on next page)

tools are becoming available to the professional programmer, regardless of which language or software environment he or she prefers to work with. In addition to COGEN for RM/COBOL, we have recently seen the introduction of such tools for MBASIC, dBASE II, and others. It's my hope and belief that this new generation of software products will raise the design and implementation of routine systems from the level of a black art to that of a lucid and comfortable activity.

COGEN is a professional product in the best sense of the word, and deserves serious consideration by any COBOL programmer.

**TABLE 1**
**Facts & Figures**

**Package or Version name:**
COGEN, Version 5.2

**Price:**
$ 30 (Manual only)
$100 (Demo system – applicable toward purchase price)
$950 (Complete package)

**Systems Available For:**
CP/M, OASIS, IMOS, RT-11, and IRX systems running RM/COBOL

**Required supporting software:**
CP/M Version 2.2 or later
A complete program development environment would also require a text editor or word processor, and the RM/COBOL modules

**Memory requirements:**
40K RAM

**Diskette capacity required:**
Two single-density disk drives minimum

**Utility programs provided:**
none

**Record size & type limits:**
Records may be fixed or variable length, to a maximum of 999 characters
File organization may be Index, Relative, or Sequential
Access mode may be Dynamic, Random, or Sequential

**Portability:**
Files created by COGEN may be incorporated into programs written in any ANSI COBOL-74 which uses the 'COPY' statement
Use of tun-time module for RM/COBOL requires payment of license fee and royalty to Ryan-McFarland, authors of RM/COBOL – call (408) 662-2522 for information and royalty schedule

**User skill level required:**
experienced COBOL user or professional programmer

**System upgrade policy:**
All registered COGEN owners will automatically receive updates from BYTEK as they are released

**TABLE 2**
**Qualitative Factors**

| | Rating* |
|---|---|
| **Documentation** | |
| organization for learning | 6 |
| organization for reference | 5 |
| readability | 7 |
| includes all needed information | 6 |
| **Ease of Use** | |
| initial start up | 6 |
| conversion of external date | 4 |
| application implementation | 6 |
| operator use | 4 |
| **Error recovery** | |
| from input error | 4 |
| restart from interruption | 7 |
| from data media damage | 1 |
| **Support** | |
| for initial start up | 6 |
| for system improvement | 7 |

*Ratings in this table wil be in a 1-7 scale where:
1 = clearly unacceptable for normal use
4 = good enough to serve for most situations
7 = excellent, powerful, or very easy depending on the category

**TABLE 3 – APPLICATION DEVELOPMENT FACILITIES**

| FUNCTIONAL PARTS | COMPLETENESS AND COMPLEXITY OF FACILITIES | | | |
|---|---|---|---|---|
| | LITTLE OR NONE | SOME | COMPLETE & COMPLEX | EASILY COMPLEX |
| INDIVIDUAL PROGRAM DEVELOPMENT | | | X | |
| INPUT TRANSACTIONS | | | | X |
| DATA MANAGEMENT | | X | | |
| REPORTS & QUERIES | | | | X |
| INTEGRATED SYSTEMS | | X | | |

Feature

# How To Be A Manipulator (of Z80 data)

Kim West DeWindt

As we continue on through the saga of the Z80 instruction set, we move into the realm of manipulation: Arithmetic, logic, all of the instructions that alter data. This is really the first level of data manipulation. Final data, information that people can use, has been altered from the computer's binary coding into 'real' numbers. Z80 instructions manipulate data, it is your program's duty to turn those ones and zeros into ASCII – or some format meaningful to the human eye. Though binary data does look good on a string of Christmas tree lights.

As before, I do not go into elaborate details about those instructions that are identical to the 8080 instruction set. This discussion centers on those instructions that are unique to the Z80. Z80 mnemonics are used, and when possible, the corresponding Intel mnemonics are shown for cross reference.

This general introduction is followed by a discussion of the Z80 arithmetic instructions, adding, subtracting, etc. Next in line are the logical instructions. This section ends with a comparison of the various rotate and shift instructions. If you have any questions, see me after class.

The mnemonics used throughout this tutorial are those shown in the Zilog Assembly Language guide. For a more detailed description of the addressing modes that are referenced in this discussion, see the earlier sections of this tutorial.

There is one group of eight bit operands that are used in the arithmetic and logical instructions. Rather than re-iterate them in each section, I am going to define them here and note which instructions use this set. The set of operands includes the general purpose eight bit registers (A,B,C,D,E,H, and L), the memory location pointed to be the HL register pair (known as memory location M, to all of you Intel groupies) and the memory location pointed to by one of the index registers (IX or IY). This indexed addressing is unique to the Z80. A fixed eight bit offset may be added to the contents of IX or IY at the time of execution. Throughout this section, I refer to this group of operands as the standard set. In addition, the terms 'accumulator' and 'A register' are used interchangeably.

In the mnemonic examples, the Zilog mnemonic is on the left, the Intel counterpart (if it exists) is on the right. A description of the opcode's function is directly under each example.

## Arithmetic instructions

The arithmetic instructions require that the accumulator be the destination register. There is no way to overide this fixed destination. These arithmetic operations are performed in the accumulator, might as well save transit time and leave the results where they are calculated. If the results of an arithmetic operation are needed elsewhere, you have to perform the calculation. Then (in a future instruction) send the information to its final resting place. (Also watch for next month's section on the Move instructions). Despite the limitations of a fixed destination register, the programmer does have a wide selection of operands to use as the source of data. These include the standard set of operands (mentioned previously) and the option of imbedding data in the opcode – immediate data.

To summarize: the accumulator must contain one byte of the data that is being used in the operation. The second byte of data can be in one of the general purpose registers (including A), in memory, or in the instruction stream. If the data is located in memory, the address of that data can be in the register pair HL or in one of the index registers. The results of the operation are stored in A, the original value of A is destroyed.

The arithmetic operations that can be performed are: Add, Add with Carry, Subtract, and Subtract with Carry. (In subtraction, the carry bit is used to indicate borrowing). In the examples below, the standard set of operands or immediate data may be used as the source (second) operand.

ADD A,B ADD B

Add the contents of the B register to the accumulator.

ADD A,6 ADI 6

Add 6 to the A register. Note that Intel uses a different mnemonic for the Add Immediate instruction.

ADC A,D ADC D

Add the contents of D and the value of the carry bit to A.

ADC A,10 ACI 10

Add to A the immediate data 10 and the carry bit. Again, Intel forces the programmer to remember another opcode.

SUB A,(HL) SUB M

Subtract the contents of the memory location, whose address is in HL, from A.

SUB A,35 SBI 35

Subtract from A the value 35. (Yet another Intel mnemonic)

SBC A,(IX+0) SBB no equivalent

Subtract the value of the carry bit and the contents of the memory location, whose address is in the IX register (offset = 0), from the accumulator. Intel calls this a Subtract with Borrow, and has no indexing mode – the user is constrained to using one of the eight bit registers or M.

SBC A,(IY+16) SCI no equivalent

Subtract from A the data stored at the address in IY, (add the offset of 16 to that address), and the value of the carry bit. There are eight different Intel mnemonics needed to match four Zilog mnemonics.

## Sixteen bit arithmetic

Unlike Intel, Zilog uses similar mnemonics for the sixteen bit arithmetic instructions. The opcodes are the same, only the operands have been changed to direct the data flow. During sixteen bit instructions, the register pair HL

(continued on next page)

corresponds to the accumulator, becoming the predefined destination. The arithmetic function is still processed in the accumulator, but partial and final results are stored in HL. For sixteen bit operation, the source operand must be one of the register pairs – BC, DE, HL, or SP. Sixteen bit instructions look like this:

ADD HL,BC DAD B
Add the contents of BC to the contents of HL (and another Intel mnemonic).

ADC HL,DE no match
Add the contents of DE and the value of the carry bit to HL.

SBC HL,SP no match
Subtract the contents of the stack pointer (SP) and the value of the carry bit, from the contents of HL.

Note that SUB (subtract without carry) is not used with sixteen bit operands. Since the Z80's internal data bus and the accumulator are only eight bits wide, all sixteen bit operations must be done in two steps. In a sixteen bit subtraction, the low bytes of the two operands are subtracted, then the high bytes are subtracted. The carry bit must be taken into consideration when subtracting the high order bytes, in order to record the effect of a carry out (borrow) from the low byte operation. The carry bit is always used in sixteen bit subtractions, and this is reflected by the use of the SBC opcode.

Two special arithmetic instructions require only one operand. These instructions are Increment, which automatically adds one to the operand, and Decrement, which automatically subtracts one from the operand. The operand can be any one of the standard set of arithmetic operands:

INC B INR B
Increment (by one) the contents of register B

DEC (HL) DCR M
DEC (IX+3) no match
Decrement the contents of the memory location whose address is in HL. With the Z80, the memory location can also be addressed by one of the index registers modified by an offset (3).

The sixteen bit version of Increment/Decrement can be used to alter any one of the register pairs, or one of the index registers. The opcodes are the same, just insert the appropriate register pair in the operand field:

INC BC INX B
Increment the contents of the register BC.

DEC IX DCX no equivalent
Decrement the contents of the index register IX. (No index registers in the 8080).

## Accumulator Arithmetic

There are a few arithmetic instructions that can manipulate only the accumulator. One of them, Decimal Adjust (DAA), is that little known opcode that mysteriously appears in most microprocessor instruction sets. DAA is used, after the addition or subtraction of BCD digits, to convert the contents of the accumulator into BCD digits. The mnemonic looks like this:

DAA DAA
Convert the contents of the A register into packed BCD.

BCD stands for Binary Coded Decimal, and it is a simple representation of decimal numbers in binary code. One nibble represents one decimal digit (from 0 to 9). Packed BCD means that two decimal digits are stored in one byte – one digit per nibble. For example, the decimal number 27 looks like this in BCD:

```
    2          7          ;decimal digits
  0010       0111         ;BCD digits
high nibble low nibble
```

All addition and subtraction instructions set or clear the half carry bit (H). The state of H tells the processor (and the programmer) whether or not there was a carry out from bit 3 (the most significant bit of the lower nibble), into bit 4 (the least significant bit of the high order nibble). DAA tests the state of this flag when converting the contents of the accumulator into valid BCD digits.

Complement (CPL) is another instruction that can only be used on data that is in the accumulator. To complement a binary word, the processor complements (toggles) the value of each bit. All the ones become zeros, and all the zeros are changed into ones. The result is known as the one's complement of the original number. The mnemonic for the complement instruction is:

CPL CMA
Complements the contents of the A register.

The third accumulator only instruction produces the negative value of the data of the A register. That is, it forms the two's complement of the data in A. The two's complement of a positive number is a binary representation of that value as a negative number. Conversely, the two's complement of a negative number, is a positive number. This is a signed format, where the value of bit seven (the most significant bit) represents the sign of the number. If bit seven is a 0 – the number is positive, if its value is 1 – the number is negative. In operation, the Negate instruction subtracts the value of A from zero. The mnemonic is:

NEG no match
Negate (form the two's complement of) the number in A.

## Logical Instructions

Actually, this first instruction falls into that gray region between math and logic. The Compare instruction functions like the Subtract instruction, subtracting the second operand (immediate data, or one of the standard arithmetic operands) from the accumulator. However, it is a non-destructive subtraction. The flags are toggled to reflect the results of the subtraction, but the data in the accumulator is not changed. Only the altered state of the flags reflects the fact that an operation occurred. Despite the fact that nothing changes in the operands, this is a very handy instruction. It appears in counting loops, and string search routines – for the very reason that it does not destroy the information that you are looking at, yet gives some indication of what it is. An example of the opcode looks like this:

CP B CMP B
CP 3 CPI 3

Compare (subtract) the contents of the B register (or the immediate data 3) from the A register – adjust the flags to reflect the results. Note that Intel requires a different opcode if the programmer wishes to compare immediate data.

The remaining logical instructions perform the software equivalents of hardware logic. In these instructions, as with Compare, the implicit destination is the accumulator. The second operand can be immediate data or one of the standard operands. Here are some examples (When Intel requires the use of a different opcode with immediate data, I have listed it directly below the basic form of the opcode):

```
AND (HL)        ANA M
AND 3           ANI 3 ;immediate
                data opcode
```

Logically AND the contents of A and the contents of the memory location pointed to by HL.

```
OR (IX+4)       ORA no match
OR 5            ORI 5 ;immediate
                data opcode
```

Logically OR the contents of A and the contents of the memory location pointed to by IX, offset by a distance of 4.

```
XOR C           XRA C
XOR 2           XRI 2 ;immediate
                data opcode
```

Logically eXclusive OR the contents of A with the contents of the C register.

## Block Compare

The Z80 has a special version of the compare instruction. Similar to the block move instruction, the Block Compare instruction can automatically address a block of memory. Comparing the contents of the current memory location with the contents of the accumulator, it will automatically search through the selected memory block. There are a number of conditions that will terminate the progress of the instruction. These include reaching the end of the memory block, or finding a match between data in memory and the accumulator data. Block Compare can move forward (up) or backward (down) through memory. It is extremely useful when searching through memory for a particular piece of information. and can be tailored to any number of sort, match, or compare routines.

Three pieces of information must be defined before using the block compare. The A register must contain the seed data – the data that you are looking for. The HL register pair must contain the starting address of the memory block that is to be searched. The register pair BC is a counter that may contain the number of times the compare loop should be repeated. If the instruction is structured to perform just one compare at a time (non-looping), the contents of BC is arbitrary. Once those three pieces of information are in place, the block compare can be directed in four ways. The opcodes are as follows (operands are implicit):

```
CPI                 no match
```

Compare A with the memory location at (HL), then increment HL, decrement BC and stop (ie, do not loop).

```
CPIR                no match
```

Looping version of CPI, continues until the contents of A matches the contents of the memory byte at HL, or until BC is zero.

```
CPD                 no match
```

Compare A with the memory location at (HL), then decrement HL, decrement BC and stop (ie, do not loop)

```
CPDR                no match
```

Looping version of CPD, continue until the contents of A matches the contents of the memory located at (HL), or until BC is zero.

In future sections discussing programming techiques, there will be examples incorporating the block compare instructions. For now, let's move on to the roundabout instructions, shift and rotate.

## Shift and Rotate

Shift instructions allow the programmer to shift data through an eight bit register or memory location, moving zeros into the empty bits that are left behind the shifted data. You are allowed to choose the direction that the data is to be shifted to. Each shift, left or right, will move all of the data in the addressed byte one bit to the left (or right).

Shift operations always include the carry bit as part of the operand. Like the poor relations that always seem to hang around, the carry bit is there to catch whatever is dumped out of shifted data byte. During a shift left, the high order bit (bit 7) is shifted into the carry bit. The low order bit (bit 0) goes into the carry bit during a shift right. Unlike rotate instructions, shift will dump the information that is shifted out of the carry bit into a silicon void, never to be heard from again.

The shift instructions back fill with zeroes. In other words, the Z80 puts a zero into the bit that is on the end of the addressed data byte. If the program calls for eight consecutive left shifts, the shifted data byte will contain all zeros after the last shift. (This is one very slow way to clear a register). This is what the mnemonics look like:

```
SLA B               no match
```

Shift Left (Arithmetic) the contents of the B register (bit 7 is shifted into the carry bit, bit 0 is filled by a zero).

```
SRL (HL)            no match
```

Shift Right (Logical) the contents of the memory location addressed by HL (bit 0 goes into the carry bit, bit 7 becomes zero).

In addition to shift left and shift right, the Z80 has an instruction called shift right arithmetic. In this case, the Z80 does not back fill with zeroes, but replaces the upper most bit (bit 7) with the value of the bit before the shift began. This is useful when dividing a signed number by two. A regular shift right would replace the value of the seventh bit (the sign bit) with zero, destroying the original sign of the data – and drastically altering results of the arithmetic operation. The shift instruction can be used with any of the eight bit registers, and with memory location that is pointed to by HL or one of the index

(continued on next page)

registers. Here is an example of this:

SRA (IX+7) no match

Shift right arithmetic the contents of the memory location pointed to by IX (plus an offset of 7). Bit 7 remains the same, preserving the sign of the original number, bit 0 goes into the carry bit.

After a shift instruction, the state of the carry bit can be tested to determine its value. It is a bit awkward, but by implementing the appropriate number of shift instructions, the programmer can effectively look at the value of any bit within the addressed byte, after shifting it into the carry bit. However, there is a much simpler way to do this. The Z80 instruction set includes a test instruction which can look at the state of any bit within the eight bit registers or in any byte of memory. I will cover the operation of these instructions in a future section.

## Rotate Instructions

Rotate instructions are used to move the bit pattern of the addressed byte through that byte and, optionally, the carry bit. The data can be rotated to the left or to the right. Data that is rotated out of one end of the byte is rotated back into the byte at the opposite end. If the carry bit is included in the rotate instruction, it takes the position of a ninth bit. None of the original data is lost, it is just repositioned. Rotate instructions can be used by the programmer to inspect different bits within a byte. The appropriate number of rotate instructions move the bit into the carry flag where a conditional test of the carry flag determines the state of the bit.

Any one of the eight bit registers can be rotated. In addition, the memory byte addressed by HL, IX, or IY can be rotated. This addressing flexibilty is in contrast to the 8080 rotate instructions which are limited to the accumulator. There are four basic mnemonics for the rotate instructions. They are:

RL A RAL

Rotate Left the A register, include the carry bit (bit 7 is moved into the carry bit, carry bit was moved into bit 0)

RLC C RLC (A only)

Rotate left circular the C register, move the value of bit 7 into the carry bit, and into bit 0. (The 8080 can only rotate data that is in the A register)

RR (HL) RAR (A only)

Rotate right the memory location at (HL), include the carry bit (bit 0 moves into the carry bit, carry bit was moved into bit 7).

RRC (IX+0) RRC (A only)

Rotate right the memory location at (IX), exclude the carry bit from the rotation loop.

Personally, I find these four mnemonics confusing. My first reaction to the mnemonic RLC is to assume that it includes the carry bit in the rotation, whereas RL merely moves bit 7 into the carry flag, but does not move the carry flag back into the addressed byte. However, Zilog chose to use the letter C to designate Circular. No doubt meant to imply that the rotation is contained within (circular to) the addressed byte.

The Z80 has a novel form of the rotate instruction that swaps information between the A register and the memory location addressed by the HL register pair. It is designed to swap nibbles - BCD digits, and affects the entire addressed byte but only the lower nibble of A. It is called Rotate Right (or left) Digit, and I have yet to find an appropriate use for it. The BCD digits of the addressed memory byte are rotated left or right, the displaced nibble goes into A, with the old contents of A filling in the blank nibble at the memory location - sounds confusing enough. The mnemonics look like this:

RLD no match

Rotate the BCD digits left (lower nibble of A goes into the lower nibble of the memory location, old lower nibble of the memory byte goes into the upper nibble of the memory byte, and finally, the old upper nibble of the memory bytes goes into the lower nibble of A. The upper nibble of A is not affected.)

RRD no match

Rotate the BCD digits right (upper nibble of the memory byte is rotated into the lower nibble of the memory byte, the displaced lower byte moves into the A register, and the old contents of A rotates to the vacant upper nibble of the memory byte. The upper nibble of A is not affected)

There must be some rational reason for the existence of this instruction. On the other hand, it might be an undocumented instruction that was 'found' and then promoted as a feature...

Enough speculation. In an upcoming issue this tutorial will discuss jumps, conditional jumps, and call.s These are the instructions that let you alter the program counter, and consequently, the flow of your program. If you thought unsigned shifts could throw off a program, wait until you see what a misdirected jump can do.

## Feature

# Another Chapter In The Continuing Saga . . . BASIC/Z

Jethro Wright

When I was offered this assignment, my first reaction was: "What? Another BASIC?". After all, software vendors offer such a wide selection of different "flavored" BASIC's that I wasn't sure whether the world was ready for another. In spite of this I took up the challenge and the following is my response to that challenge.

## The Package

BASIC/Z arrives from Systemation, Inc. on a single density 8" diskette which contains six program files, accompanied by a 250+ page reference manual. BASIC/Z is a native-code 8080/Z80 compiler for CP/M 2.XX computer systems. This environment is segmented into three parts: an executive, a run-time, and an installation component.

The BASIC/Z.COM and .OVL files provide the initial level, where programs can be created, modified, compiled, and displayed. These operations are facilitated by a set of nearly forty commands that cover virtually all the tasks necessary. There is even a Config command that allows an individual to tailor the system, to better utilize the machine running it.

RUN/Z.COM, invoked through the BASIC/Z executive module, loads your object file after it's been processed by the compiler. The file contains all the general-purpose routines for string-handling, arithmetic, and input/output that would be required by any standard program.

INSTALL/Z.BZO is an independent BASIC/Z program that modifies RUN/Z.COM to support certain specific I/O features found on most computers using the BASIC/Z package. These features can then be accessed or manipulated through special commands in the running program. The special keywords can activate video attributes, position the cursor, or take advantage of the editing capabilities found on all popular CRT terminals.

PATCH.BZO is another separate utility included to simplify the eventual issue of program maintenance. RUN/Z and BASIC/Z can be interactively updated by the user, to incorporate "bug fixes', as distributed by Systemation via its Software Information Bulletins.

The final program of the distribution set is TR/III.BZO, a special conversion utility that will convert programs written for BASIC/S – running under MDOS from Micropolis. This particular program will not be discussed in this article.

## BASIC/Z Itself

BASIC/Z.COM and its attendant overlay file require a minimum TPA size of forty Kbytes, a CRT terminal, a single disk drive, and CP/M 2.XX, for program development. Once loaded, it responds to a command set which is similar to – yet different from – the ubiquitous MBASIC system from Microsoft. However, BASIC/Z's designers have placed a greater emphasis on providing an interactive environment. The most prominent example of this emphasis is the automatic syntax checking feature. As each line is entered into its program buffer, BASIC/Z will inspect that line for proper syntax and immediately enable its edit mode if the line doesn't pass its test. When a source program is loaded from disk, all invalid lines will be deleted unless a special command switch is applied to the Load command. The same Load command switch will also include all necessary line numbers for programs created using a standard program editor.

The speed at which programs are displayed on the system console can be controlled. Since any key will temporarily stop the action, it's a much simpler matter to display a long program at a pace that suits the user. Listings sent to the printer are properly paginated, with the program's title and an optional program header printed at the top of each page.

Unfortunately, BASIC/Z does not support full screen editing and offers a line editor almost identical to that supplied with MBASIC. Unlike the product from Microsoft, BASIC/Z does include separate global search and replace functions, to supplement its intrinsic editing facility. Search strings can even have imbedded wild card characters.

An almost revolutionary enhancement is the Config command which modifies BASIC/Z.COM and .OVL, once again allowing the user to operate a system that is friendlier and more convenient to use. Certain console control characters can be re-defined, as well as the default parameters for both console and printer. You even have the option of enabling special interception of fatal CP/M BDOS errors – Bad Sector, R/O, and Select – if any occur while BASIC/Z is running. As stated above, BASIC/Z is a compiler, another feature accessible while in the executive mode. Its operation is simple to master, like all the other commands in BASIC/Z's executive repertoire. I was dismayed, though, when I found that no provision had been made for a subsequent link edit step. The linkage editing process promotes the development of pre-tested, relocatable, object libraries, so that writing programs becomes easier and less error-prone, over the course of time. That's my personal opinion, of course, though BASIC is often popular because of its suitability in providing "quick and dirty" solutions to many problems. So perhaps the lack of a linker will be good news to some few folks.

## Programmatically...

The BASIC/Z run-time environment is like that of other popular BASIC language products. RUN/Z.COM is a static, object loader/library similar to BRUN.COM for Microsoft's BASCOM. This program is explicitly invoked to load the user's compiled, object program so that a fixed

(continued on next page)

portion of memory is always inaccessible to user programs. The advantage of this arrangement is found in the special INSTALL/Z utility that can patch RUN/Z.COM to support direct cursor addressing, screen editing commands, advanced video attributes, printer page parameters, and fatal BDOS error trapping.

Some of BASIC/Z's keywords are compatible and syntax is the same as found in the other BASIC language processors. Variable names can have any number of alphanumeric characters. However, when it comes to actual data BASIC/Z takes a larger stride away from the crowd. First, there are Control variables that are single-byte unsigned values. Next, there are Control/D variables which are double-byte, 16-bit unsigned numbers. Integers, on the other hand, are packed-BCD entities that can be specified to be three to ten bytes long. Real numbers are also stored in BCD form and range in size from four to eleven bytes in length. Strings are allowed a length of zero to two hundred fifty bytes. If the default allocation for either integers, reals, or strings is inadequate, the value may be changed by way of the Sizes statement. Any data type may be part of a Common statement as long as the applicable type sizes match. Arrays are allowed, though without any memory-limited number of dimensions. Four dimensions for numbers and two dimensions for strings should be enough for most "quick and dirty" tasks, especially since specified arrays can be dynamically allocated via the Ddim and Erase statements.

Character I/O is performed through Print and Input statements when whole lines are being manipulated. Output can be re-directed to the printer or console exclusively, to both simultaneously, to a null device, or to a pre-declared spool file. Input from the console can be edited interactively with a series of nine cursor movement and editing keys. Instead of implementing a Print Using statement, BASIC/Z has a separate Fmt function that generates an edited string from the numeric argument passed.

Disk I/O under BASIC/Z takes some getting used to. Two primary types of files are recognized: random and sequential. Sequential files are very straightforward in that there is only one delimiter of a logical record: a carriage return/line feed sequence – and only one record type: a string type. Therefore, there is no limitation on the contents of a given record, as long as it doesn't contain a <cr/lf> pair. Random files get a bit hairier, since there are two sub-types to contend with: BASIC/Z random and Unfmt random. BASIC/Z random files have an invisible one-hundred twenty-eight byte header as record zero, that contains pointers to each record written to the file. These files can have logical records of any allowable size, but if larger than two-hundred fifty bytes, the Sizes statement must be explicitly involved to declare the maximum length. Clearly, this is the most flexible file I/O mode as a variety of options are available for retrieving the full range of data types processed by BASIC/Z. Unfmt random mode restricts the logical record size to one-hundred twenty-eight bytes. This mode is offered to establish portability between BASIC/Z and other language systems.

While line numbers are absolutely mandatory for a BASIC/Z program, an alphanumeric label can be equated to a line number, giving the illusion of structured BASIC program. These symbols can be the target of any BASIC/Z statement that can change program control, like Gosub <destination>, Goto <dest>, or If <condition> Then <dest> Else <dest>. In a similar vein, multi-line, user-definable functions are supported, as well as thirty-two assembly-language subroutines. BASIC/Z also has the full complement of iteration structures: the Do/Until, While/Wend, For/Next loops. Two potentially abusable, yet handy enhancements are the Push <dest> and Pop statements. These statements facilitate program stack manipulation, as available in assembly language programming, so that error handlers can more efficiently direct program flow control. Unfortunately, if these statements are used indiscriminately, a totally incomprehensible rat's nest of a program will result.

BASIC/Z has a few truly unique verbs like its Decr and Incr statements, that decrement or increment, respectively, a numeric variable. A feature on the verge of remarkable is the Sort statement, which will order any of BASIC/Z's array variables in either ascending or descending sequence. Then, of course, there are the special video mode statements. The user can select the reverse video, blink or normal video renditions or erase to end of line or end of screen. There is even a special two argument Tab clause for the Print statement that implements direct cursor addressing.

Finally, there's the subject of debugging, an area where most compilers fail, since few vendors supply the proper tools or enough data about the internals of their products to allow meaningful run-time debugging. While Systemation has decided not to "burden" BASIC/Z users with internal specifications on their compiler, they have at least included a special Debug statement that will monitor a selected range of line numbers – or the entire program – permitting greater programmer control in a real-time situation. There are 4 types of Debug statement, the most important of them being the single-step type that causes program execution within the desired range to execute one line, then dump the contents of up to four scalar variables on the console.

## Documentation

I received Version 1.01 of the BASIC/Z compiler so that only a preliminary manual was available for inspection. While bound in a nice, standard sized, three-ring binder, its contents definitely were not the last word in documentation. It was printed on an inexpensive dot-matrix printer, with no page numbers and therefore no index. A final production version is being prepared, that will hopefully remedy some of these deficiencies; it is reported to include BASIC/Z programming examples as well. This manual will be provided at no cost to present purchasers of the system. In spite of these comments the documentation is satisfactory for those already familiar with the idiosyncrasies of other BASIC language products.

## The Wrap Up

Throughout this article, I have often referred to Microsoft's BASIC language products, since they're among the most popular items on the software market. While I have my own

personal reservations about them, they are clearly the standard by which others are to be judged. Using BASIC/Z, I have written a label writer program that operates as well as a comparable program I have been using which was written with MBASIC. Since both are generally I/O bound, there are no significant parameters to distinguish either system as superior.

It is notable that BASIC/Z has been designed to acknowledge the existence of CRT terminals, something that other languages fail to address at all. Moreover the BASIC/Z user can programmatically set printer form length and width, so that all printer output will paginate properly. These factors go a long way towards simplifying the I/O interface, and thereby making programs easier to write.

BASIC/Z definitely has an adequate repertoire with which to perform in any situation where MBASIC would be used. Its enhancements certainly justify the conversion effort of someone who does not have a very big investment in MBASIC code. Number crunching, as required by many engineering and scientific applications would suffer, however, because BASIC/Z's real numbers are BCD values and not true binary floating-point numbers. Execution time for complex calculations could easily increase by a factor of ten due to this difference. I don't see this as being particularly significant to the majority of potential BASIC/Z users. Important also is the fact that there is no licensing agreement required for distributing compiled BASIC/Z programs. That alone says quite a bit for this product.

All things considered, BASIC/Z is worth the time and effort spent learning to master it. I find the unfamiliarity of some of its enhancements to be the only real drawback in using the system. I hope we continue to see it mature over the course of time.

**TABLE 1**
**Facts & Figures**

**Program Name:**
BASIC/Z – an enhanced native-code compiler for 8080/z80 CP/M 2.XX-based microcomputers

**Produced By:**
Systemation, Inc., P.O. Box 11, Richton Park, Ill. 60471

**Cost:**
$395.00 for a single-user license – no licensing required for distributing compiled programs

**Distribution Kit Consists Of:**
BASIC/Z.COM and BASIC/Z.OVL – a program development executive and compiler;

RUN/Z.COM – the BASIC/Z run-time environment;

INSTALL/Z.BZO – special configuration utility for RUN/Z.COM;

PATCH.BZO – BASIC/Z system updating utility;

TRIII/Z.BZO – MDOS BASIC/S to BASIC/Z conversion utility program;

**System Requirements**
40 Kbyte minimum TPA, a single disk drive, a CRT terminal, and a CP/M 2.XX-compatible operating system

**Reviewer's Comment:**
Good tool for one-of-kind, straight-forward applications where interaction I/O is important

## Feature

# The New PL/I

Bruce H. Hunter

There is a new PL/I, in two senses of the term "new." Digital Research's newest version of Subset G, and Subset G itself. PL/I has been the most powerful language available for microcomputers for some time now, and its newest version is more powerful yet.

Right now many of you may be thinking, "PL/I? Why on earth would anyone want to get involved with that gargantuan language!" Unfortunately, when most people think of PL/I, they think of the giant language of two decades ago, which has probably frightened off more people than it has claimed. The language in its full set is awesome, if not awe inspiring. When many software authors make reference to PL/I, they touch lightly on the subject, making an inside joke about its size and inferring that it would be a nice language to study if you had a lifetime to devote to it. Others simply say, "I tried it and its works." The language has suffered too long from ignorance of its syntax and abilities. It has suffered unfairly from writers not giving it the exposure it deserves. The time has come to put it in the hands of the public rather than discouraging them from using it.

Why get into PL/I in the first place? Well, this is what PL/I can do. It crunches numbers as well as FORTRAN, (and now better). It handles strings as well as BASIC, but with more finesse. It has an exquisite structure like Pascal, but even more flexible. It has the file handling ability and aggregate structure ability of COBOL, but ever so much faster and much much more compact. Code in PL/I, like Pascal, can be poetry, a thing of artistry and beauty. It creates functions and throws pointers around like C. Its ability to do data structures, arrays, structures of arrays, arrays of structures, and arrays of structures of arrays, etc., will match C's anyday (if, indeed, you ever wanted to do that sort of thing), but C is a mid level language and PL/I is high level. It interfaces with data base managers and screen formatters like a giant mainframe system. It creates and links to macros like only an assembler's language can. It does it all, and has a compiler that takes up less room than Whitesmith's C. Can it do everything? Almost. Is it worth the effort to learn a language this large? You'd be a fool not to learn it. Is there a version of PL/I available for micros and minis? Of course. It's called PL/I Subset G. And Digital is coming out with an incredibly exciting new version of Subset G soon which will really "blow you away."

A couple of decades have gone by since PL/I was first implemented. The language seemingly dropped out of the main stream while COBOL, BASIC, and a little more recently, Pascal have come into dominance. In the meantime, dedicated programmers needing a powerful and versatile language have encouraged the use of PL/I on minis and super minis and micros as well. The language was trimmed down from the figurative physique of a Russian weight lifter to that of a lithe long distance runner. ISAM and VSAM files were dropped. The functions that may have found a use once or twice a decade were dropped. So were some of the matrix functions and other "niceties" that were less than essential. An ANSI committee was formed for the standardization of PL/I's new subset. The committee (X3J1) standardized the new subset, and it was named Subset G. The committee is very active today, and so is the subset!

**Subset G** is easy to learn and easy to implement. It is still the most powerful language available today for micros and minis, and even more powerful in the sense that it is available! The subset can be considered to be an improvement over the original set in many ways. It is now truly portable, and has a sufficiently compact compiler to run on any 64 kilobyte system with a couple hundred K bytes on disk.

Dealing with specific applications for a computer language is essential for any programmer today in order to utilize the most suitable language for the purpose. Granted, PL/I is not a teaching language; BASIC and Pascal are ideal for that. FORTRAN was the language of choice for heavy duty scientific programming, but now PL/I will do it better. And believe it or not, PL/I will write business programs better than any language in existence that I know of. I don't say this lightly. I write business data base programs, and believe me, I searched a long long time for the best language I could find. PL/I Subset G was it. It serves very well as an engineering language, too. If you deal with systems languages predominately, then PL/M is more up your alley, but PL/I has most of the facilities of PL/M! So you can do calls to BIOS directly without having to revert to assembly language calls, and PL/I's ability to manipulate storage gives you power you may never have thought possible in a high level language.

Is it fast? Is it compact? If speed turns you on, a now famous benchmark program published in September '81 of Byte ('A High-Level Language Benchmark" by Jim Gilbreath) clearly shows PL/I's Subset G to be the fastest 8 bit language available. Faster than C, faster than FORTRAN, faster than RATFOR, faster by far than the BASIC compiler and very much faster than the Pascal or BASIC interpreter. The size of the code was more compact than any language tested other than RATFOR. It was over 300 times faster than COBOL and nearly three times more compact. If execution time has been slowing you down, perhaps you ought to take another look at PL/I.

Some of you may be looking for a language that is structured. For me it was a top priority. Well sir, PL/I *IS* structure, and it has the most versatile and varied types of structure available. It has blocks (program structures that can be entered from the top and exited at the bottom without deliberate transfer of control). It has procedure blocks (program structures that can only be entered by being called, and automatically transfer control to the next exe-

cutable line after the calling statement). It has functions as well (a form of procedure that is called and returns a value). And, all of these program structures can have global or local variables! Procedures can be external to the code (entry or public). The value of a variable can be passed to a procedure and operated upon with or without changing its value, by reference or by value. That is, it can change the value of the variable as it is stored or leave it alone. So if you like ALGOL-type languages, this is one of the best.

What PL/I can do with arrays is nothing short of unbelievable. It has two data structures. Not only the usual lists, arrays, and tables, PL/I has aggregate data structures as well. The aggregate data structure is a grouping of variables of the same and/or different types into a single data entity. It is similar to a Pascal Record or a COBOL Record. For example,

```
declare
      1 act_structure (3000),
            2 name character (32) varyng,
            2 account_number fixed,
            2 address,
                  3 street char (32) varying,
                  3 city char (20)varying,
                  3 state char (2),
                  3 zip fixed;
```

Any reference to the account structure by the name "act_structure" will deal with the entirety of the data above. It can be taken from or put out to disk file simply by

```
read file (act.dat) to (act_structure);
```

Now couple this power with the ability to put structures into an array. The declared structure above is a 3000 element array. The structure could just as easily have held an array. PL/I arrays can be moved in an entire "block'.

```
account_mat = act_structure;
```

This moves the entire 3000 element array "act_structure" to the 3000 element array account_mat. The bounds of the array may be negative, and arrays can start at any integer value. There are functions to return the high and low bounds of arrays, and the wonders go on and on.

PL/I's files are varied and powerful. All I/O is a file, and any device that can be patched into the operating system is addressable as a file. Files can be stream (sequential) or direct (random) and ample facilities are available for keyed files (files to carry the keys of other files). Keyed files provide the mechanism by which data files can be sorted and searched. Without keyed files, management of the data base is impossible. And PL/I can be tied to a number of data base managers, including Digital Research's new Access Manager. That means that the significant loss of ISAM and VSAM files from Subset G is no longer an inconvenience, because the ability to couple PL/I with a data base manager will buy back any loss in spades and then some.

Branching and decision-making in PL/I is an art form. The "if-then-else" is nestable to incredible depths and combinations. 'If-then-else-if" is easily implemented, and the use of "null else's" allow unbalanced nesting. PL/I has a "labeled goto" that will branch just about anywhere. The label can be a constant or a variable, and it can be subscripted. There is both a "do while" and an "iterative do" as well as an "iterative do while." There is even a "non-iterative do" to hold execution into a mini-block! But before going on any further about the many attributes of PL/I, wait until you get into the newest version.

## PL/I-80 version 1.4 and PL/I-86

Digital Research unquestionably feels that the future of micro-computers lies in the use of 16 bit processors, at least for the next decade. Their 16 bit laboratory has been hard at work for some time now expanding the Digital line into 16 bit. DRI's commitment to multi-user and networked systems also has been monumental. The micro is taking over the sacred ground of the mini and supermini. However, the world of 16 bit multi-user systems like MP/M-86 brings new problems as well as benefits. With data now open to multiple users at multiple terminals, how can sensitive information be protected? When a data base is open to multi-users, files are now vulnerable to both accidental and deliberate damage, and sensitive information can be read by people who shouldn't have access to it. It is clear that file and record locking has become a necessity.

Digital's recent language releases have been in both 8 bit and 16 bit versions. They have the features of file and record locking, and the newest version of PL/I is no exception. Files opened in PL/I now can be "locked", "shared", or "read-only". Files opened in the default mode are "read-only" and cannot be used by other users. "Shared", of course, will allow universal free access. Passwords can also be used. With the passwords come three levels of privilege: "read", "write", and "delete". With "read", you must use the password just to read the file. "Write" means the file can be read, but a password must be used to write to the file. "Delete" will allow free reading and writing, but the password is now required to delete the file. In addition, there are functions to lock and unlock individual records. Not only will these protect sensitive and privileged information, but they also protect a record from being accessed while being altered (updated) within a multi-user environment. The password is part of the file name:

```
A<B:MYPROG.COM;JBOND
```

File protection is handled in the environment section of the open statement, and the password is assigned in the title section.

```
OPEN FILE (F_1) DIRECT UPDATE ENV
(SHARED,P(D),F(128))
    TITLE ('B:INV.DAT;HIDE');
```

Here is a file that is opened for random update in the shared mode that will require a password ("hide") for deletion.

Another issue to be addressed is Intel's 8086 family of computer chips. We all know of the 8086 16 bit microprocessor. It is a part of Intel's third generation of processors, and there is a super chip to go with it called the 8087, an 80 bit word math processor. There are few adequate superlatives to describe the incredible power of this state-of-the-art number cruncher. The problem: creating software that is ready to make use of this phenomenal processor. The new versions of PL/I will emulate 8085 operation until the 16 bit version and the chip itself come together.

(continued on next page)

**SUPER NUMBER CRUNCHING** is now available on micros, thanks to the existence of math processors. PL/I is now ready for them. In the last version of PL/I-80 1.3, the maximum precision available for floating point math was binary 24 (16777216). For my business and engineering programing I never had any problem with that, but I have a friend who is a statistical programmer, and he was forever using FORTRAN for double precision when he would rather have been programming in PL/I. Now there is an answer to his problem. The newest release from Digital has double precision. This powerful data type has a precision of up to 53. No typo, that is really it, *53*! The exponent was a minimum of 38 for single precision before, but now it is a whopping 308. Good old Carl Sagan doesn't need to be limited to "billions and billions" anymore. He can come out with "billions *times* billions *times* billions. . . ."

Just for the record, Microsoft's FORTRAN-80, ANSI '66, has a double precision to 16 digits and an exponent of up to 38. What seemed phenomenal before now seems ordinary. That is the nature of our rapidly changing microprocessor technology. When future releases of PL/I integrate with a functioning 8087 to take full advantage of the 80 bit word, the precision and exponent will again become greater.

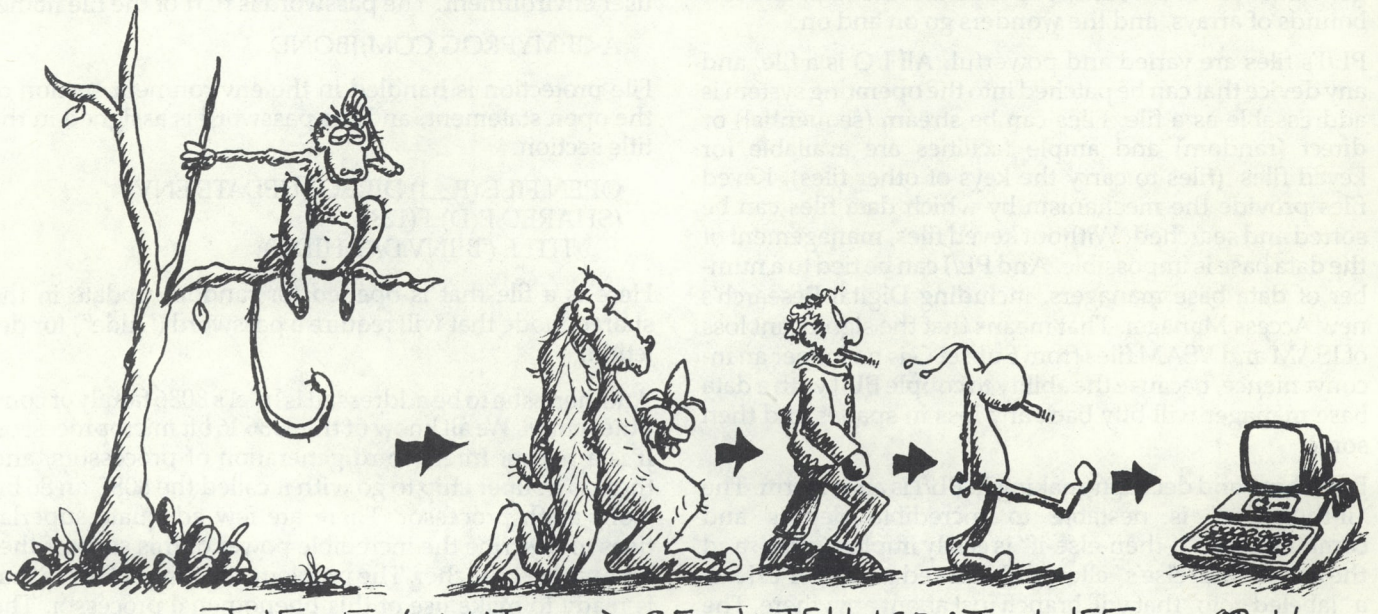
I have had the privilege of being one of the first consultants outside of Digital to read the advance documentation on the new versions. It is the most exciting technical reading I have seen in a very very long time. The first big surprise is it is good documentation. In fact, it is extremely good documentation! I have never been bashful about attacking the technical obscurity of Digital's manuals. (I did not coin the phrase "Digital has never been accused of hiring writers," but I certainly have used it enough.) But we are looking at a new Digital as well. Carmen Governale, the Product Marketing Manager of the Language Division, has insisted on and gotten the best documentation I have seen yet. It is extensive, it is clear, and it is well written. The language manual and programming guide account for a full inch and three eighths of paper, but there was no problem reading through the new language manual in a single evening. I am in the literal middle of writing a book on PL/I Subset G, called "A 'Basic' Approach to PL/I', and one of the things I have been trying to shoot for is CLARITY. That is the way this manual comes across, and it's great to see.

The first release of the new PL/I will be the 8 bit version 1.4. It has been in Beta test for some time now, and as of my last conversation with DRI, it is only waiting for the final documentation to be completed. The eight bit version 1.4 is an emulation of the 16 bit version, PL/I-86. This thoughtful feature guarantees full compatibility with both versions. Compile it to 8 or 16 bit, no difference, no changes in the source code.

Pictures! The documentation has pictures. Illustrations abound. Now you can really see a linked list. Data aggregates are real and visual. Memory and buffer maps are truly maps, not just descriptions. These additions ameliorate the learning of the material which gets you faster into actual programming. A glance at the Programmer's Guide shows not only the old programs that existed in the 1.3 version, but many new ones with more tables and pictures for clarity. You should find it very impressive and readily usable.

We find ourselves in a new era of micro-computing. Processors like the 8085, 8086, and 8088 have given us memory that is limited only by the availability of card slots in the card cage and our ability to pay for memory boards. Fully integrated multi-user systems are plentiful, with many below $10k. The 8085/8088 co-processor in my S-100 is old hat now. Pascal and BASIC interpretors/compilers that used to provide all the programming power 8 bit 64K machines could handle are now just not going to hack it. Today, serious programming applications will have to look for more powerful languages. PL/I is here and readily able to give that kind of power.

So, if you are a serious programmer or thinking about becoming one, what are you waiting for? Take a look at PL/I Subset G. You will sooner or later. Why not now? ■



# Volume 91, Catalogue & Abstracts

# CP/M Users Group

COMPILED BY: Ward Christensen

## Catalogue

DESCRIPTION: Microsoft FORTRAN programs:
(1) Spectrum analysis programs by Victor DePinto.
(2) Print formatting programs by Lawson F. Pierce for Microsoft FORTRAN and M-80.
See ABSTRACT.091 for details.

| NO. | SIZE | NAME | COMMENTS |
|---|---|---|---|
| | | -CPMUG.091 | CONTENTS OF CP/M VOL. 91 |
| | | ABSTRACT.091 | Abstract of volume contents. |
| | | FILES.CRC | CRC of all files on disk |
| | | CRCK.COM | CRC generator (CRCK ** F) |
| 91.1 | 26K | ARIT.COM | (1) Executable arithmetic functions. |
| 91.2 | 4K | ARIT.FOR | (1) Arithmetic functions. |
| 91.3 | 1K | ARIT#.SUB | (1) Submit file. |
| 91.4 | 2K | DISKIO.BAS | (1) Disk file I/O in BASIC. |
| 91.5 | 21K | FFT.COM | (1)Executable FFT program. |
| 91.6 | 27K | FFT.FOR | (1) 1024 point FFT. |
| 91.7 | 4K | FFT#.SUB | (1) Submit file. |
| 91.8 | 6K | HI.FOR | (1) DMP2 driver. |
| 91.9 | 2K | HIA.MAC | (1) DMP2 driver. |
| 91.10 | 1K | NAME.FOR | (1) Filename getter. |
| 91.11 | 4K | PLOT.FOR | (1) Plotter plot routine. |
| 91.12 | 9K | PRINT80.COM | (2) COM of PRINT80.FOR |
| 91.13 | 1K | PRINT80.DOC | (2) DOC PRINT80.FOR |
| 91.14 | 4K | PRINT80.FOR | (2) Compresses M80 PRN files |
| 91.15 | 4K | PRINT80.REL | (2) REL of PRINT80.FOR |
| 91.16 | 10K | PRINT81.COM | (2) COM of PRINT81.FOR |
| 91.17 | 6K | PRINT81.FOR | (2) Print FORTRAN and ASM files with page ejects and titles |
| 91.18 | 5K | PRINT81.REL | (2) REL of PRINT81.FOR |
| 91.19 | 1K | QCHEK.MAC | (1) Checks for abort (Q) command. |
| 91.20 | 17K | SPECTRUM.DOC | (1) Documentation. |
| 91.21 | 5K | TTYPLOT.FOR | (1) Printer plot routine. |
| 91.22 | 2K | TWIDDLE.FOR | (1) Generates twiddle factor table. |
| 91.23 | 8K | UTIL.FOR | (1) Utility program. |
| 91.24 | 1K | UTIL#H.SUB | (1) Submit file. |
| 91.25 | 1K | UTIL#T.SUB | (1) Submit file. |
| 91.26 | 25K | UTIL-HI.COM | (1) DMP2 version of UTIL.COM. |
| 91.27 | 24K | UTIL-TTY.COM | (1) TTY version of UTIL.COM. |
| 91.28 | 5K | WIND.FOR | (1) Window functions. |

(continued on next page)

## Abstracts

FILE NAME: (Note: All files submitted are part of the spectrum analysis software package.)

| FILENAME | COMMENTS |
|---|---|
| ARIT.FOR | Arithmetic functions |
| ARIT.COM | Executable arithmetic functions |
| ARIT#.SUB | Submit file |
| DISKIO.BAS | Disk file I/O in BASIC |
| FFT.FOR | 1024 point FFT |
| FFT.COM | Executable FFT program |
| FFT#.SUB | Submit file |
| HI.FOR | DMP2 driver |
| HIA.MAC | DMP2 driver |
| NAME.FOR | Filename getter |
| PLOT.FOR | Plotter plot routine |
| QCHEK.MAC | Checks for abort command |
| SPECTRUM.DOC | Documentation |
| SPECTRUM.TEX | TEX input file |
| TTYPLOT.FOR | Printer plot routine |
| TWIDDLE.FOR | Generates twiddle factor table |
| UTIL.FOR | Utility program |
| UTIL#H.SUB | Submit file |
| UTIL#T.SUB | Submit file |
| UTIL-HI.COM | DMP2 version of UTIL.COM |
| UTIL-TTY.COM | TTY version of UTIL.COM |
| WIND.FOR | Window functions. |

AUTHOR/SUBMITTED BY:
Victor DePinto
2627-148th Ave. S.E., Apt. 10
Bellevue, Washington, 98007

THIS PROGRAM IS PUBLIC DOMAIN BECAUSE: Submitted by author. I have checked with Microsoft, and they say it is ok to submit the COM files compiled by FORTRAN-80.

TO WHOM WOULD THIS PROGRAM BE USEFUL: Useful in the fields of Signal Processing, Statistics, Mathematics, Electronic Music, Physics.

BRIEFLY DESCRIBE THE PROGRAM FUNCTION: This is a package of programs which perform the FFT and inverse FFT and perform various arithmetical manipulations on signals as well as display the results.

WHERE IS FURTHER DOCUMENTATION AVAILABLE: Documentation is in the file SPECTRUM.DOC on this disk. Further technical information on the subject of Signal Processing may be found in textbooks such as *Digital Signal Processing* by Oppenheim and Schafer, available from Prentice Hall.

HARDWARE DEPENDENCIES: 48k total RAM. Standard 80 column CRT terminal. Printer with at least 80 columns. May optionally use the Houston Instruments DMP2 plotter connected to a serial port (CP/M PUNCH device).

SOFTWARE DEPENDENCIES: CP/M 1.4 or 2.x.

SOURCE PROCESSOR: Source code provided is for Microsoft FORTRAN-80, and MACRO-80. COM files are also provided. Compatible data files can also be written and read in Microsoft BASIC 5.x.

DOES THE SOFTWARE "DROP IN": Yes.

HOW EASY IS THE CODE TO MODIFY: Commented FORTRAN and Assembly source code is provided.

(2):

File Name: PRINT80.FOR,PRINT81.FOR
PRINT80 – Compresses M80 PRN files
PRINT81 – Prints FORTRAN and ASM Source files with page ejects and titles

AUTHOR:/SUBMITTED BY:
Lawson F. Pierce
2516 Sunnybrook Dr
Kalamazoo MI 49008

THIS PROGRAM IS PUBLIC DOMAIN BECAUSE: Submitted by author; author's approval

WHO WOULD THIS PROGRAM BE USEFUL TO: All CP/M users [who use M-80 orFORTRAN-80 from Microsoft]

BRIEFLY DESCRIBE THE PROGRAM FUNCTION:
PRINT80 – prints M80 files with choice of Headers or no headers, Filename in header, and compresses out surplus blanks so it will fit on a narrow printer.
PRINT81 – prints FORTRAN and ASM files with page ejection so that Subroutines can be paged separately. Ejection flags do not interfere with Assembler or Compiler
See PRINT80.DOC

WHERE IS FURTHER DOCUMENTATION AVAILABLE: PRINT80.DOC; Modified Extensively for better or worse from EDITM.FOR – CPMUSER 26.23

HARDWARE DEPENDENCIES: None

SOFTWARE DEPENDENCIES: STANDARD CP/M

SOURCE PROCESSOR: Microsoft FORTRAN

DOES THE SOFTWARE "DROP IN": yes

HOW EASY IS THE CODE TO MODIFY: Commented FORTRAN file

# Product Status Reports

The new software products and new versions described below and on page 00 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

### GENERAL LEDGER, FINANCIAL REPORTER, ACCOUNTS RECEIVABLE & ACCOUNTS PAYABLE

Information Unlimited Software Inc.

These are the first three members of IUS Inc.'s new six-program Financial Management Series for the IBM Personal Computer. General Ledger and Financial Reporter automate the bookkeeping process from posting individual transactions to producing up-to-the-minute income statements, balance sheets and financial reports. A built-in text editor and variable report generator allow users to produce both financial and other customized reports. Additional features include access to account or budget figures, audit trails, a flexible chart of accounts, departmental profit and loss statements, and year-end closing of revenue and expense accounts. The program also offers an extensive error detection feature to ensure data integrity and avoid costly mistakes.

Accounts Receivable (AR) automates the receivables function of any business, compiling customer statistics, calculating finance charges, generating customer statements and reporton the current company cash flow. Program capabilities include access to account information, comprehensive aged trial balances, choice of open-item or balance-forward posting, automatic finance charge calculations, detailed audit trails, standard reporting and customized statements and letters. AR also offers an extensive error detection feature.

Accounts Payable (AP) automates the vendor payment process. It compiles vendor reports, assists in taking advantage of discounts, calculates total purchase amounts and automatically prints detailed payment checks. In addition, it generates up-to-the-minute reports to help the user determine both immediate and future cash requirements. Features include access to vendor data including vendor statistics, open-item posting, automatic check-writing, detailed audit trail, comprehensive payable reports and customized checks and payment advices. Error detection is a standard feature.

Each of these programs is interactive with the other. All operate with the IBM PC under DOS. The financial programs are priced at $600 each or $1500 for the set of three.

### EZ-MAIL

Lifeboat Associates

Electronic mail through US Postal Service Ecom network for .26 cents/page. Requires a modem, and a modem control for asynchronous transmission. Recommend ASCOM, SMARTMODEM for asynchronous transmission. Recommend RBTE-80 for synchronous transmission. Cost is $149.

### THE TEAM MANAGER

Open Systems, Inc.

The Team Manager is a report writer and data formatter. It allows a user to create new reports using the data files from Open Systems' accounting applications. In addition, The Team Manager can reformat data from the accounting applications for use by popular word processing, spreadsheet and data base management systems.

The Team Manager will be available in Business BASIC running under CP/M-80, MP/M, CP/M-86, MP/M-86, MS DOS, PC DOS and XENIX. The product will be available for single- and multi-user configurations.

### HALO

Lifeboat Associates

Full color graphics for IBM PC routines callable from IBM or Microsoft BASIC, PASCAL or COBOL for drawing bars, boxes, lines, circles, arcs, rotating, moving, defining character sets and filling shapes with color. This is a full SIGGRAPH "CORE" standard graphics package, requiring IBM color graphics card for color or monochrome card, if no color is required. Cost is $150.

## New

## Publication

*CPM® Revealed* by Jack D. Dennon
Hayden Book Company, Inc.

The book describes in detail the potential of CP/M — the most popular operating system for microcomputers. It explains the technical aspects of CP/M including the console monitor, the system manager and the input/output driver package and the data structure of the CP/M disk. Detailed discussions of booting up, logging in, changing memory size, mapping disk space, calling programs, file handling and interfacing techniques are included. Also included are CP/M utilities and programming exercises that a reader can use with any CP/M based system. Available in paperback for $13.95.

(continued on next page)

# Books

# Books

*BASIC: Fundamental Concepts*, by Joseph C. Giarratano. Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1982. 198 pages (8½ x 11" paperback). $22.95.

*BASIC: Advanced Concepts*, by Joseph C. Giarratano. Howard W. Sams & Co., Inc., Indianapolis, Indiana, 1982. 214 pages (8½ x 11" paperback). $22.95.

As an instructor of BASIC programming classes for beginning and advanced students, I am always on the lookout for well-written textbooks to recommend to my students. Interestingly enough, although the marketplace seems to be flooded with BASIC texts, there are very few that I recommend wholeheartedly. In my opinion, such a text should aim to be either a tutorial or a reference book – it's hard enough to be a good example of either of those types without trying to be both. A tutorial should be clearly written, proceed in an orderly manner, and have lots of practical examples. A reference text should be concise, include examples, and information should be easily and quickly accessible to the user; the format of a reference text is extremely important.

With the above criteria in mind, I read the two new BASIC texts by Giarratano. Both are printed in large format paperback and are nicely laid out. Not surprisingly, both have the same general strengths and weaknesses. These are really Volumes 3 and 4 of a set of four – the second book begins abruptly right where the first ends, as if the books were originally a single volume that was dissected neatly by the publisher.

*Fundamental Concepts* is of the "tutorial" variety. It begins with an interesting history of BASIC and discusses the essential concepts and philosophy of the language. In particular, I liked the discussion of flowcharting and the emphasis on the need to learn structured programming techniques. I am constantly cautioning my students (and myself!) to avoid the "top-down" approach to programming which BASIC seems to encourage – the tendency to compose at the keyboard and to produce one-dimensional programs rather than thoroughly planning and organizing a program before starting to write code. Happily, Giarratano discusses this in the very first chapter, although I would have preferred even more emphasis on the importance of this approach.

In the second and third chapters, the book tells how to calculate with BASIC (in essence, how to turn your machine into an expensive calculator) and introduces many of the BASIC functions, emphasizing the mathematical ones. Mathematics-oriented readers may have been pleased, but I was puzzled to see things like arctangent in the third chapter of a beginning BASIC book.

The fourth chapter introduced the concept of creating entire BASIC programs, followed by chapters entitled Editing Commands, Input, Saving and Retrieving Software, etc. Loops, arrays, and subroutines were the last topics discussed in the book. Appendices included Error Messages, ASCII Codes, descriptions of booting up the appropriate machines (examples in the book were prepared on a DEC PDP-11 and an Ohio Scientific Challenger 2P). Finally, a comprehensive and well-organized index was included (one of my pet peeves is a technical book with a poor index, or none at all – this book passed that test with flying colors).

First, let me list what I liked about the book. The author is a talented writer, and has an enjoyable writing style; most of his definitions and descriptions are clear and concise. The exercise problems included in several chapters are particularly good, very appropriate for the subject matter covered in the respective chapters. The printing and format are pleasing to the eye – an important quality. And as I mentioned earlier, I liked the emphasis on program organization and the index.

Now to the faults, and there are some big ones. My major complaint with this book is its organization (or lack thereof). To be frank, it's awful – the book seems to flit from one topic to another in nearly every chapter, and several important definitions and discussions wound up hidden in rather strange places. For example, several interesting discussions of style (Programming Style, Software Maintenance, and Structured Programming) appear at the end of the chapter on "Loops". A separate "Style" chapter certainly would be justified, and could include similar discussion from other parts of the book. In addition, many of the discussions in the earlier parts of the book are really somewhat advanced topics to the average BASIC user, while some of the most essential definitions, ones that should appear early on and be very explicitly defined, are not found until the later chapters. In short, everything that should be in the book is there, and is well-explained, but the order of presentation is very poor.

Some of the definitions would benefit from more detailed examples, especially graphic ones. For example, PRINT formatting (TAB, SPC, punctuation, etc.) is discussed in various parts of the book, but nowhere are there graphic examples of what the various PRINT modes look like (how TAB differs from SPC, how the use of commas differs from the use of semicolons, and why). A set of sample printouts showing the effects of various print commands would have been very helpful.

In summary, the book has some good features and interesting discussions, and is well-written. Unfortunately, the organization is poor enough to cause me to recommend against its purchase by any BASIC user, particularly at the considerable price of $22.95. If the book was completely reorganized and the topics presented in a more logical and less confusing manner, it might be a different story. As it stands, however, the book is neither a good tutorial nor a good reference book.

* * *

*Advanced Concepts* is more of a reference text than the *Fundamental* book but it has many of the same strong and weak points. The first chapter discusses string functions (CHR$, ASC, STR$, etc.), beginning its discussion with nary a word of introduction (in the preface, the author points out that the *Fundamental* and *Advanced* books are two consecutive volumes of a four-volume set, but I

really think this volume should be more stand-alone). Subsequent chapters cover Exact Arithmetic Calculations, Accuracy and Precision (continuing the trend of mathematical emphasis), Files, and Direct Memory Access (POKE and PEEK). Then we're given the same Appendices that appeared in the first book, and the book ends.

Like its predecessor, the *Advanced* book suffers from poor organization. Giarratano continues his tendency to skip around from one topic to another, even though most of the individual discussions are adequate or better. In particular, the Files chapter is very poorly written, presenting that important topic in a very confusing manner. Again, a better-organized outline would have really helped a lot.

Unfortunately, this book has a broader and more important fault not shared by its companion volume: inappropriate emphasis on certain topics. For example, the chapter on exact arithmetic is an interesting discussion of ways to get a very high precision (40 digit numbers and the like), but consumes 58 of the book's 214 pages. Throw in the following chapter on accuracy and precision, and we have used up nearly half the book! While the existing chapters may appeal to the hard-core mathematicians among us, I can think of a number of more appropriate topics that could have been covered in more detail than they were (print formatting, for example). And lest any detractors think that I have a prejudice against higher mathematics, let me point out that I have a degree in math and *still* think too much emphasis was placed on it.

As must be obvious by now, I do not recommend this book. The price of the book (like its companion volume, a rather steep $22.95) could be much better spent elsewhere. Meanwhile, if I ever find the definitive BASIC text, I'll let you know.

* * *

*BASIC: Computer Programs for Business. Volume* 2. By Charles D. Sternberg, Hayden Book Company, Rochelle Park, New Jersey, 1982. 376 pages, paper, $13.95.

The microcomputer has had a profound effect on many segments of society. The little marvels have penetrated the entertainment, educational, and information markets, and more and more homes now house a family computer. Nowhere has the effect and future potential of computers been more strongly felt than in the small business sector, however. As computers have become smaller, more powerful, and less costly, more and more small businesses are finding them practical, cost-effective ways of increasing productivity and efficiency. Unfortunately, these remarkable tools are useless without proper software, and software development has typically lagged well behind hardware advances (can you think of any cases where a machine was designed around software, rather than the other way around?).

Charles Sternberg's second volume of *BASIC Computer Programs for Business* is aimed squarely at the small business sector. Like the first volume, it presents a variety of programs designed to appeal to a wide range of needs. The jacket of the book states that it was designed for "small businesses, microcomputer entrepreneurs, and independent consultants". Later in this review I'll tell you how each of these groups can benefit from this book.

A glance at the Table of Contents gives an idea of the subject matter covered: Marketing and Sales; Personnel; Administrative; Statistics; and File Handling (data base management). More than 60 complete programs are presented. And best of all, the organization, format, and usefulness of the book are top-notch, as becomes evident rather quickly.

I first encountered Sternberg's books at a local bookstore (the standard all-purpose kind, but one with a rich variety of computer books). I had been given a gift certificate for my birthday and was browsing through my favorite shelf with glee. I happened to notice Volume 1 of this set and began reading through it. My first impression was that the format seemed well-organized, easy to follow, and – to coin an already overused phrase – "user-friendly". I found a program similar to one I had been developing and started reading closer. Wow, a flowchart, and a record format, and a nice, concise description of the objectives and output of the program! And then a neat, readable listing of the program itself, with lots of REMs. Finally, a screen display example and a printed output list. Very complete!

Initial impressions of Volume 1 proved to be correct, and Volume 2 follows nicely in that tradition. The unwritten theme of the book seems to be that microcomputers have something for every small business. Many of the applications described in the book are potentially useful to a very broad range of businesses, and Sternberg has provided a lot of answers to the question "what can my computer do for me?".

Written in Altair Extended Disk Basic, the programs in the book can be adapted, with minor changes, to run on most other small business systems. For instance, the BASIC commands and statements are very similar to those in Applesoft or MBASIC. Sternberg has tried to avoid compatibility problems by eschewing uncommon BASIC terms and those which tend to differ widely between versions. In addition, subroutines are used extensively, particularly for those statements which may not be compatible; in those cass, the user need only replace the given subroutine with one more appropriate for his machine. A glossary in the Appendix lists the statements used, enabling the user to compare them with those used in other computers.

And here's how the three types of users mentioned earlier can make use of this book:

1. Small businesses. Most such users are unable to afford outside computer services, such as software consultants, and most do much of their own programming. The programs listed can be used as is or modified slightly. Furthermore, there are many useful ideas for use in other BASIC programs, allowing an inexperienced user to see various statements and techniques in the context of a working program.

2. Microcomputer entrepreneurs. By using the programs and ideas in this book as a foundation, these users can create "custom" applications programs for clients.

(continued on next page)

3. Independent consultants. Nearly any client's needs are served somewhere in this book. Again, slight modification can produce a useful program at a minimal cost.

Last but not least, the price is reasonable in comparison with many competing books. Computer books are, in general, quite expensive, many of them well over $20 in price (even for paperbacks!). There's a lot in here for $13.95.

Do I like this book? Obviously. Do I recommend it? Absolutely!

Reviewed by George H. Taylor

## New

## Versions

1. ASCOM ver 2.20
2. BOSS financial Acctg
   FULL SYSTEM & DEMO ver 1.15
3. BDS "C" Compiler ver 1.5
4. BSTAM for IBM PC DOS
5. dBASE II/PC
   FULL SYSTEM & DEMO ver 2.3d
6. GrafTalk ver 105
7. Magic Print ver 1.22
8. Micromotion - Forth 79 ver 2 for Z-80 CP/M & Apple Users
9. Precision BASIC ver 1.6
10. PRO-MAN (of American Software) ver 5.0
11. STIFF UPPER LISP ver 2.9
12. T/MAKER III, version 3.0

### Notice

The January issue was placed in the mail on January 5th. Due to changes in our production schedule, Lifelines/The Software Magazine, will be mailed the last week of the month preceding the current issue. This will take effect beginning with our March issue.

# KIBITS

# OOPS!

We inadvertently dropped a section of Charles E. Sherman's December 1982 MicroMoneymaker's Forum.

The missing material, an important part of his Quickey review, is printed below. It should have appeared on page 29, column two, after the paragraph which reads "Word-right and word-left tabs each get their own function key, and so do scroll-up and scroll down. Here again, HOME is an amplifier."

| | |
|---|---|
| HOME U | scroll up one screen |
| HOME D | scroll down one screen |

The repeat command key can be used with the scroll keys or up and down arrow keys to provide continuous scrolling.

It's all right there at your fingertips. I find this a lot easier to use, and to teach, than the way WordStar does things. *Quickey* also offers a big improvement in the file handling commands, which are all initiated by the FILE key:

| | |
|---|---|
| FILE U | update and continue editing |
| FILE E | update and end |
| FILE X | update and exit |
| FILE Q | quit, no update |
| FILE C | copy a file |
| FILE A | read a file in (Append) at the cursor |
| FILE R | rename a file |
| FILE * | delete a file |
| FILE P | print a file |
| FILE F | show file directory during edit |

In all the above examples, either upper or lower case characters have the same effect.

Personally, I think *Quickey* is an enormous improvement on WordStar, and offers innumerable advantages. By now you may ask, "Is there a catch?" Well, yes, a small one. At present, *Quickey* is only available for Televideo terminals and computers, although it will soon be implemented on others that have plenty of function keys.

Did I find anything to complain about? Hah! Need you ask? Do hackers have hangnails? *Quickey* on the Televideo is forced to make use of certain function keys that generate a three character code, and WordStar isn't quite fast enough to swallow them all when entered in rapid succession. Unfortunately, Raish Enterprises didn't catch this and assigned those keys functions which people do tend to jiggle quickly, such as character delete, line scroll up, line scroll down. Results? When overloaded, WordStar understands the first two characters as a command which it doesn't recognize, but treats the third as an input character which it writes into your text. This can scatter unwanted characters around like little mouse pellets. Not nice. If you refrain from impetuous key jiggling, this doesn't happen; however, I still recommend that you make sure you get an updated version which has fixed this problem.

Our apologies to Mr. Sherman & our readers for this error.

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

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