Computer Systems

HEWLETT PACKARD

issue no. 3

Communicator 250

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

table of contents

from the editor's desk

We've Moved to California

by Ralph White

As you read this edition of the Communicator, you will notice a change in address for correspondence. The HP 250 has moved to California! A decision was made by Hewlett-Packard to consolidate all business oriented computer systems under one roof. Since the 300 and 3000 were located in the Bay Area, logic dictated that the 250 join those product lines there. This consolidation will allow us to readily share knowledge and support expertise across products.

Several people from Colorado with a great deal of 250 experience made the move to California and continue their high level of support. Other experienced Hewlett-Packard people as well as some bright new faces have joined the program. Both marketing and R&D are in full force in California and manufacturing will be moving soon.

Some changes have come with the move. One of these changes is covered in detail in the Marketing Bulletin section of this Communicator.

The move to California has gone smoothly and the 250 is thriving in the warm Bay Area climate. Support from the field and factory will continue at the high level you have come to expect from Hewlett-Packard. Your comments and suggestions about this magazine are always welcome. Please address your comments to:

Communicator 250
Hewlett-Packard
General Systems Division
19447 Pruneridge Avenue
Cupertino, California 95014

Structured Programming

by Gretchen Snowden

One of the new features introduced with the last operating system was structured programming. Those of you who are familiar with structured programming welcomed this addition to Basic on the HP 250. This article is intended for those who are not familiar with these statements.

The structured programming statements are a series of loops and branches which increase the readability of a program. Used throughout a system, these statements can increase the uniformity and hence ease of modification of the programs. The IDENT statement makes it easy to obtain uniform indentation. This also makes it easier to pick up a program and follow the logical flow.

Below is a program segment taken from the HP 250 demo that does not use structured programming. There are four sections in the original program, each demonstrating one of the four methods of accessing a data base. The segment shown demonstrates the direct method. The user is asked to enter a record number (line 1670). The number entered is verified as numeric (line 1690) and then a direct mode DBGET is done (line 1710). The condition code is checked for a valid entry and either the data found or an error message is displayed. The program then returns for another entry. The loop is exited by pushing the softkey marked EXIT. The subprogram Cur repositions the cursor.

```
1590 !
1600 !    Direct access
1610 !
1620 Direct:  OFF KEY 8
1630      ON KEY 8,"EXIT" GOTO Clrr
1640      DISP Clr$;
1650      CURSOR (4,5)
1660      DISP "DIRECTED ACCESS:";SPA(2);"Reads the entry at a specific
               address";LIN(1)
1670 Dirin:  INPUT "Enter the record number of the entry to be retrieved:
               ";In$[1,10]
1680      GOSUB Cur
1690      IF FNNum(In$,1) THEN Bad_rec
1700      Rec_no=VAL(In$)
1710      DBGET (Base$,"PRODUCT",4,S(*),Lst$,Buff$,Rec_no)
1720      IF (S(0)=13) OR (S(0)=12) THEN Deof      ! End of file?
1730      IF S(0)=17 THEN Dempty                   ! Empty?
1740      IF S(0) THEN Dberr
1750      DISP LIN(1);"Entry for record number";Rec_no;":";LIN(1)
1760      DISP USING Head5
1770      DISP USING Head2;Product_no,Prod_desc$
1780      DISP LIN(1)
1790      GOTO Dirin
1800 Deof:GOSUB Cur
1810      DISP "Record number ";Rec_no;" is out of the range of the set"
1820      GOTO Dirin
1830 Dempty: GOSUB Cur
1840      DISP "The entry for record number";Rec_no;"is empty."
1850      GOTO Dirin
1860 Bad_rec: GOSUB Cur
1870      DISP "Invalid record number, please re-enter"
1880      GOTO Dirin
1890 !
1900 !    Calculated access
1910 !
```

Listed below is the same program segment, rewritten to include three of the structured programming statements. The IDENT command has also been used to provide uniform indentation which makes the program easier to read. The first statement (line 1615) indicates that the four access methods are now handled with a SELECT . . . CASE statement based on which softkey is pressed. This is the second of the four methods. The next construct used is the LOOP . . . END LOOP (lines 1665 to 1885). This has replaced four GOTO statements in the old program and more clearly shows that the program is looping. In this case, the exit comes from the softkey.

The next construct used is the IF . . . THEN . . . ELSE in lines 1690, 1858, and 1880. If the entered number is numeric the DBGET is performed. Otherwise, an error message is printed and the END IF executed. This replaces a branch and GOTO, again making the logic easier to follow.

The final construct used is the SELECT . . . CASE following the DBGET. The old program checks the condition code for four values and then branches to a different program segment for each error. With the SELECT statement, no branching is necessary since the statements to be executed follow each CASE statement. The CASE ELSE statement handles all other values of the condition code.

This is a small example showing only some of the statements available, but illustrates the increased readability of a program that uses structured programming.

```
1590 !
1600 !    Direct access
1610 !
1615      CASE 2
1620        OFF KEY 0
1630        ON KEY 0,"EXIT" GOTO Clrr
1640        DISP Clr$;
1650        CURSOR (4,5)
1660        DISP "DIRECTED ACCESS:";SPA(2);"Reads the entry at a specific
              address";LIN(1)
1665        LOOP
1670          INPUT "Enter the record number of the entry to be retrieved:
               ";In$[1,10]
1680          GOSUB Cur
1690          IF NOT FNNum(In$,1) THEN
1700            Rec_no=VAL(In$)
1710            DBGET (Base$,"PRODUCT",4,S(*),Lst$,Buff$,Rec_no)
1720            SELECT S(0)
1730            CASE 0
1750              DISP LIN(1);"Entry for record number";Rec_no;":";LIN(1)
1760              DISP USING Head5
1770              DISP USING Head2;Product_no,Prod_desc$
1780              DISP LIN(1)
1790            CASE 13,12
1800              GOSUB Cur
1810              DISP "Record number ";Rec_no;" is out of the range of
                 the set"
1820            CASE 17
1830              GOSUB Cur
1840              DISP "The entry for record number";Rec_no;"is empty."
1850            CASE ELSE
1855              GOTO Derr
1857            END SELECT
1858          ELSE
1860            GOSUB Cur
1870            DISP "Invalid record number, please re-enter"
1880          END IF
1885        END LOOP
1890 !
1900 !    Calculated access
1910 !
```

'XREF' Utility Uses

by Miles Kehoe

One of the more useful utilities in the programmer's toolbox is a cross reference program which will provide information about all variables, functions, line labels, and constants within any program. Such a cross reference is invaluable in the support of applications written by someone else, and indeed, tracing program flow without a cross reference becomes quite a tedious task.

The HP 250 'SYSTEM' disc includes just such a program, 'XREF'. With it, you can cross reference any PROG type file and obtain as extensive a cross reference as you probably need. However, from time to time there are things you may want to modify in this run-only program, and an attempt has been made here to include solutions to some of the more common requests.

Performance

'XREF' is, at best, not a high throughput program, but considering its complex task, it performs acceptably. However, for those who use a 7906 disc drive on our HP 250's, there is a way to improve the speed of 'XREF'.

Keep in mind when using these suggestions, that 'XREF' will look to the current mass storage device for a file unless you specify a label or physical device. This also applies to files specified in a 'batch' input file.

First, let's make the 'XREF' program, its subprograms, and its data base reside on the 7906 disc. This can be done using two other utilities found on the 'SYSTEM' disc, 'ROUTIL' run-only program utility and 'DBMODS' modify IMAGE data base utility.

Start by inserting the 'SYSTEM' disc in one floppy drive and the desired platter in the 7906 disc. For the purposes of this article, assume that the label on the 7906 disc is 'CONLIB'.

Run the 'ROUTIL' program, and select the 'COPY' function softkey (SFK# 1), specifying the 'SYSTEM' disc as the source device and the 'CONLIB' disc as the destination device. Select 'XREF' from the list of RUN-ONLY programs presented, and 'XREF' and its associated subprogram will be copied to the destination device. You will see three ERROR messages during the copy: 'ROUTIL' cannot copy data bases, so the root file and both data sets will generate errors. Nonetheless, the program will complete its transfer.

Now that the program is copied, exit 'ROUTIL' and run 'DBMODS' which will permit us to move the data base. 'DBMODS' will first request a data base name, volume, and maintenance word: specify data base 'REF1' on 'SYSTEM', which normally has no maintenance word associated with it as supplied by HP. Once 'DBMODS' has accepted this information, it will display the main menu.

Since you want to move the data base, first use the 'SET' softkey (SFK#3). That will lead to a 'MODIFY SET' softkey (SFK#5), which will display both data sets associated with 'REF1' : 'SYMBOLS' and 'LINES'. Change the label on each set to the desired destination label, here 'CONLIB'. Once this change is made, exit to the main menu (SFK#8) and select the 'VOLUME' softkey (SFK#4). Move the root file (SFK#2) to the new (destination) label, 'CONLIB'. Exit 'DBMODS' and the change is made!

'XREF' will now run from the 7906 disc, should be somewhat faster, and will surely save head wear on your 'SYSTEM' floppy. You may want to leave 'XREF' on the 'SYSTEM' disc for backup purposes, but remember: 'REF1' data base does not exist on 'SYSTEM' any more, so running 'XREF' from the 'SYSTEM' disc will produce a 'Data base not found' error. 'DBMODS' has taken the liberty of moving the entire data base and purging it from the source volume.

In summary, then, perform the following steps:

RUN"ROUTIL"

- Copy"XREF" to the ",CONLIB" platter
- Be prepared to see three 'errors'
- Exit "XREF"

RUN"DBMODS"

- Which will move the Command Data Base and Root files.
- Move the sets using the 'SETS' softkey to the destination volume, here ",CONLIB".
- Modify the root file to reflect this change by using the 'VOLUME' softkey.
- DBMODS will have now moved both sets and the root file, and deleted the REF1 data base from the source (SYSTEM) volume.
- Exit "DBMODS".

You are now ready to run "XREF" from the new volume!

Capacity

Another question that has been asked concerns the capacity of 'XREF' and its data bases. Since 'XREF' provides each program cross reference separately from its included functions and subprograms, it will take a fairly substantial program to overflow 'XREF'. Nevertheless, there is something you can do.

The two data sets, used by 'XREF', 'SYMBOLS' and 'LINES', are preset to capacities of 511 and 1500 respectively. It would seem that by using 'DBMODS' you could change the capacities of these two sets and solve your problem. However, things are not always as they seem! The symbol table array in the 'XREF' program is fixed at 512, so increasing the 'SYMBOLS' capacity will only result in an error on running the program. In addition to variables, however, the symbol table will include names of labels, functions, subprograms, and constants. This means that a simple reference like 'Index = 1' will require a reference to the symbol '1', using one of your precious entries. By specifying not to include constants in the cross reference, you can effectively increase the number of symbols you can cross reference.

The number of lines, on the other hand, can be increased fairly easily simply by increasing the capacity of the 'LINES' data set. First, purge only the 'LINES' data set (set number "02"). Alter the capacity by using the 'MODIFY SET' key in 'DBMODS'. (Remember: the maximum number of lines any program can have is 9999, but even a large program may have no more than 2000 lines!) Exit 'DBMODS', recreate the data set, and you should be able to 'XREF' your large program.

Here is a sample of the steps involved:

```
DBPURGE "REF1";"2"        ! Purges only 'LINES'.
RUN "DBMODS,SYSTEM"       ! Modify the root file.
```

- Here make the changes of set capacity within DBMODS is described above.
- Exit DBMODS

```
DBCREATE "REF1";"2"       ! Recreate just 'LINES'.
```

Summary

These simple steps have enabled you to improve both the performance and capacity of the "XREF" utility by using "DBMODS" and "ROUTIL". Please notice that there are good reasons for not simply using the "XCOPY" binary command in moving the data base, mainly that "XREF" will not work when you are finished. In fact, you may have difficulty in purging "REF1" from its new destination if you should use "XCOPY"!

Notice also that "DBMODS" provides you some very powerful capabilities that should be used with extreme care! When in doubt, always have a back-up disc from which to recover. In a future issue of the "COMMUNICATOR", we will be looking at "DBMODS" and some of the other utilities in greater depth.

Changing a Program as it Runs!

by Miles Kehoe

A useful technique for a programmer would allow a program to be changed and re-stored based on input to the program as it is running. This would allow a set-up program, for instance, to specify certain IMAGE format statements to be used for different headers in REPORT WRITER. Another example is given below. This program could be part of a menu driven applications package that may run with any of several menu options. While you may store the options in a control file for reference, you may not wish to access that file each time you display the main menu. Using the features of HP 250 Business BASIC, you can indeed change a known portion of code within a program, re-store the changes on the disc, and eliminate any need to access a control file to discover your configuration. A similar technique could be used whereby a program accesses the control file to determine whether that program should update itself in memory and on the disc.

The program shown below prints a menu for the user to select desired applications. Here's what the program looks like on the disc:

```
1000 ! PROG1
1010 ! This program will modify itself using the 'LINK'
1020 ! command features.
1030 !
1035     DIM Buf$[80]
1040     ON ERROR GOTO Start
1050     STORE "BACKUP"              ! Make sure you have a copy
1060 Start:!
1070     ON ERROR GOTO Abort_job
1080     ON KEY #8:"EXIT" GOTO Abort_job  !The only way out
1090     PRINTER IS 8
1100     Flag=0
1110     PRINT "  "
1120 Entry:!
1130     IF Flag=1 THEN              ! Update disc file
1140          RE-STORE "PROG1"
1150          Flag=0
1160          PURGE "hp250$"
1170     END IF
1180     ON ERROR GOTO Abort_job
1190     GOSUB Main_menu             ! Offer main menu
1200     INPUT "Enter your selection)";Option
1210     IF Option=7 THEN GOTO Config
1220     IF Option=9 THEN
1230          PRINT "Othello game starts now!"
1240          PRINT "so I really have to go"
1245          STOP
1250     END IF
1260     ! Not valid at this time
1270     PRINT "  "
1280     PRINT "Please select another option for now."
1290     GOTO Entry
1300 Config:!          !This is where the work gets done
1310     ASSIGN #1 TO "hp250$",Status
1320     IF Status<>0 THEN
1322          CREATE "hp250$",10
1324          GOTO Config
1326     END IF
1330 !
1340 ! Want to change lines 1690 - 1760 into rems
1350 !
1360     FOR Index=1690 TO 1760 STEP 10
1370          Buf$=VAL$(Index)&"! New line "
1380          PRINT #1;Buf$
1390     NEXT Index
1400 !
1410 ! Let's update the PRINT statement at line 1710
1420 !
1430     Index=1770        ! Always
1440     Buf$=VAL$(Index)&"PRINT"&CHR$(34)&"9 - OTHELLO"&CHR$(34)
1450 !
1460 ! In order to get the quotes in the string, you have to use
1470 ! the CHR$(34) function: that is, the character with ASCII
1480 ! code of 34, or the double quote  (")
1490 !
1500     PRINT #1;Buf$
1510 ! Now remember the RETURN!
1520     Buf$="1780 RETURN"
1530     PRINT #1;Buf$
1540 ! Close the file
1550     ASSIGN #1 TO *
1560 !
1570 ! Flag is used to tell when to update the program on disc
1580 !
1590     Flag=1
1600 ! Now do the real work - bring the new program lines
1610 ! in from file 'hp250$'.
1620 !
1630     LINK "hp250$",1690,Entry
1640 Abort_job:!
1650     PRINT "  "
1660     PRINT "Program aborted. Re-load and view 'PROG1'"
1670     STOP
1680 Main_menu:! Lines after here subject to change!
1690     PRINT "Enter your selection:"
1700     PRINT "1 - Accounts Receivable"
1710     PRINT "2 - Accounts Payable"
1720     PRINT "3 - General Ledger"
1730     PRINT "4 - Order Entry"
1740     PRINT "5 - Inventory Control"
1750     PRINT "6 - Sales Analysis"
1760     PRINT "7 - Reconfigure software"
1770     PRINT "8 - Exit"
1780     RETURN
```

You will see that one of the menu options is to 'Reconfigure software', option 7. (All other options in the program as shown will not produce any results.) Here is an analysis of the program:

From 'Start' through 'Entry', all that is done is to set up a valid printer, ON ERROR condition, and set the variable 'Flag' to zero. 'Flag' is used at label 'Entry', a few lines down.

In 'Entry', a check is made to see if the disc version of the program requires updating based on the value of 'Flag'. The main menu is printed via the 'GOSUB' call, and the user is prompted for a selection. The only two options recognized at this point are '7 - Reconfigure software' and '9', not displayed on the menu at this time.

Upon selecting option 7, control is transferred to the 'Config' label. There, line numbers 1690 through 1760 are printed as remark statements (!) to a disc file named 'hp250$'. Line 1770 is printed to that file as follows:

1770 PRINT"9 - OTHELLO"

Notice that to print quote marks within a string the CHR$ function was used (line 1440), which basically says append the character with ASCII code 34 to the string. This, fortunately, is our desired character ("), and the task is accomplished. Of course, the RETURN statement must be included within 'hp250$', and the file is closed.

The real magic is accomplished at the end of the 'Config' paragraph: the LINK statement. LINK tells the operating system to bring a file of program statements into memory, and to begin overwriting existing statements at the first line specified (here 1690). Once the new file is loaded, all statements from the first line through the end of your program are deleted: that is why Main_menu is a subroutine at the very end of the program. The second line specifier, here the label 'Entry', is where execution is to begin once the new DATA file is LINKed into memory. Values of variables are not changed (as they would be using a GET command), and program execution continues as before.

This program, of course, is not offered as a solution to a real problem: it only gives you a feel for what can be done within your applications. You may find it helpful to review both the 'LINK' and the 'MERGE' commands in your BASIC Reference Manual to see what capabilities the HP 250 offers.

Editing Without the Editor

by Pat Ryan

The HP 250 interactive CRT is great for editing BASIC programs. Unlike most terminals, any line on the display can be changed and re-entered. This feature greatly improves the efficiency of program development and debugging. By performing a simple conversion, data string files can also be edited just like programs. To tell an editor to change line 20 and then type the change is not as easy as just changing what is seen on the screen. Except for some pattern matching commands, an editor is really not needed on the HP 250 if you are willing to perform a simple trick that will allow you to use the full interactive CRT to edit data string files.

The trick is to convert your data files to files that look like BASIC programs, edit the 'programs', and convert the files back to data form when you are done. This is easily done by making every line of your program a comment. The following algorithm, for example, will convert a data file to a BASIC format file simply by adding line numbers and an '!' to the start of every line in the file. It is assumed that Source_file$ is the data file and Dest_file$ is also a data file created at least as large as the source file. This is not a complete program, but is only intended to show the basic operations that are needed.

```
100 ON END #1 GOTO Fend
110 ASSIGN #1 TO Source_file$
120 ASSIGN #2 TO Dest_file$
130 L=990                ! Initialize line counter, start with 1000
140 LOOP
150    READ #1;Buf$      ! Buf$ dimensioned to hold largest string in file
160    L=L+10
170    Buf$=VAL$(L)&" ! "&Buf$    ! Add line number and '!'
180    PRINT #2;Buf$              ! Print the BASIC line
190 END LOOP
200 Fend: PRINT #2;END
210 ASSIGN #1 TO *
220 ASSIGN #2 TO *
```

Now, if a GET is done on the new file, the result will be a BASIC program that consists solely of commented lines. The program can be edited (line numbers changed, lines added or deleted, etc.) using all the features of the interactive CRT and BASIC program editing and RE-SAVE'ed. After the file is in final form, it can be converted back to its original form using an algorithm similar to the one above by deleting the line number and comment from each string while writing back into the original file. If the file changes in size after editing, simply create a new file that is close to the size needed.

If a program is used to make hard copy of your data, then the file need never be converted from its BASIC format. Simply modify the program to strip off the first six characters of each line (assuming numbering starts with line 1000) and print the remaining string. A program to perform the conversions outlined above has already been written and is available from the user contributed library.

Handling Binary Conversion

by Gretchen Snowden

When a new operating system is released, this often means that the binaries included with it have been updated. This may happen because of bug fixing, new features or inclusion of some streamlined coding. These changes are probably not obvious to the user. Since the binaries can be stored with a program, running with the new operating system will not necessarily mean using the current binaries. It is therefore generally a good idea to RE-STORE a program with the new binaries. The easiest way to do this is to SAVE the program which will strip off the old binaries. Then a LOADBIN will load the new binaries. The program is retrieved with a GET and then a RE-STORE is used to put it back on disc with the new version of the binaries.

If a user is unsure of whether or not a particular program has a binary, a similar procedure can be used. The program can be SAVEd which will strip any binaries. Then if the program is retrieved with a GET, any lines containing a binary call will be listed as an error. The binary needed can be found by checking the original listing. Fetching the line from memory will only yield an error message. Using these procedures will assure that all programs contain the latest and greatest binaries.

Design Objectives Used by MFG/250

by Loyd Nelson and Stacy Plemmons

One of the first tasks to be done when a system is being designed is to decide on the overall objectives of the system. This may be one of the more difficult tasks in the system design. An example of objectives used for the Manufacturing System are given below.

MFG/250 provides tools to help small manufacturers establish a solid first step toward material control. To accomplish this first step, MFG/250 was designed around four objectives.

The first objective is that the software should help users concentrate on manufacturing problems without having to worry about the computer. It should be quick and easy for the user to get to the desired feature. And, it should be as friendly (i.e. informative and forgiving) and as devoid of the computer command mystique as possible since it is aimed at the first-time-user small businessman. To meet this objective, the pack is menu driven and uses the HP 250 CRT softkeys extensively. Since manufacturers are usually separated into departments, the feature set is separated into six menus representing the five typical manufacturing departments and a common menu for all departments to use. The menus are all accessed via the softkeys, and often use the softkeys to accomplish the given task. This means the user can get quickly to the desired function without being concerned about the functions carried out by other departments. And once the user is within the desired function the softkeys help the user accomplish the given task.

The second objective of the pack is that data base modification is transaction driven. All activity changing information in the data base is handled by building a transaction which is then stored sequentially in a transaction data set. During batch update the transactions are executed and the data base updated according to the transaction execution routine. The transactions provide an accurate log of activity that has taken place. All functions generating a transaction also contain direct access to a transaction review and edit feature. This means that inadverent user mistakes do not require a backout to an already modified data base but instead allow the user to correct the mistake in the transaction (or deactivate the transaction if desired).

The third objective is that the user should be able to define several report requests and allow those requests to execute unattended. To accomplish this objective a report transaction can be defined and stored in a reports data set similar to the transaction data set. When the batch update is performed the report transactions are executed following the completion of the data base transactions. For convenience, the reports can also be run immediately. Also, it is assumed that certain reports will be run on a regular basis. The report transaction can be defined as a periodic report and can be invoked automatically on a daily, weekly (day of week specified) or monthly basis.

The fourth design objective is that the software code be modular and easily modifiable and expandable. Consistent coding standards are adhered to throughout with common subprograms providing standard utilities for user I/O, mass memory management, and data base access. A standard file naming convention has been defined so that the function of a particular file can be easily identified. Also, a set of pack support utility programs are provided to allow easy definition of new transactions to be added to the pack as well as providing a means of integrating the associated new programs into the pack. The MFG/250 Technical Documentation provides detailed information both on the design of the programs and conventions used in developing the pack. To review the technical documentation please contact your Hewlett-Packard representative and ask about product 45180-90040.

customer corner

Softkeys Used To Enter Password

by Gretchen Snowden

The program for this issue comes from Brad Kirley, the S.E. in the Rockville, Maryland Office. He did not like the idea of the password showing on the screen as it was being typed in. So he wrote this program segment which has the user push unlabeled softkeys in a certain sequence rather than enter a password.

Thanks for the contribution, Brad!

```
100  ! THIS PROGRAM ALLOWS A USER PASSWORD TO BE TYPED IN USING THE SOFT
110  ! KEYS ON THE HP250.  BY DOING THIS, NOTHING APPEARS ON THE SCREEN.
120  ! THIS SECURITY COULD BE ADDED TO ANY PROGRAM ACCEPTING PASSWORDS
130  ! WHICH APPEAR MOMENTARILY ON THE SCREEN.
140  !                                    BRAD KIRLEY, ROCKVILLE/BALTIMORE
150  !
160  !
170  !
180  ON HALT GOTO Exit
190  DISP "  "  ! CLEAR THE SCREEN
200  DISP "ENTER YOUR CODE"
210  BEEP
220 Setup: !
230       FOR I=1 TO 8             ! MAKES ANY SOFT KEY ON THE CRT WRONG
240       ON KEY #I GOTO Wrong
250       NEXT I
260  !
270  !
280 Keyset: !       THIS SETS UP THE CORRECT KEY SEQUENCE
290       READ A,B,C
300       DATA 11,15,13
310       X=0
320 One: !
330       FOR I=9 TO 16
340       ON KEY #I GOTO Wrong
350       NEXT I
360       ON KEY #A GOTO Two
370       WAIT
380 Two: !
390       FOR I=9 TO 16
400       ON KEY #I GOTO Wrong
410       NEXT I
420       ON KEY #B GOTO Three
430       WAIT
440 Three: !
450       FOR I=9 TO 16
460       ON KEY #I GOTO Wrong
470       NEXT I
480       ON KEY #C GOTO Success
490       WAIT
500       GOTO Exit
510 Success: !
520       Pass$="WHATEVER"
530       DISP "YOU HAVE SUCCESSFULLY ENTERED THE CODE."
540       ! LOAD/LINK/PASS ON TO THE MAIN PROGRAM
550       STOP
560 Wrong: !
570       X=X+1
580       PRINT "TRY # ";X
590       IF X>=3 THEN GOTO Exit
600       GOTO One
610 Exit: !
620       BEEP
630       DISP "YOU'RE GUESSING"
640       DISP "SO YOU'RE OUT"
650       BEEP
660         DEL 1,660        ! THIS LINE OPTIONAL
670       END
```

marketing bulletin

STARS and the HP 250

by Gretchen Snowden

The HP 250 is joining the STARS. STARS, which stands for Software Tracking and Reporting System, is a computerized bug tracking system. On January 1, 1980, the 250 joined the 300 and 3000 on this system. This will allow for tracking of known problems, enhancement requests, and documentation errors on an automated basis. In addition, the shift from the current manual system will enable the factory to better monitor the status of fixes for these problems.

Customers and S.E.s will now use the same service request forms used by the 300 and 3000 to submit problems or enhancement requests to GSD. Such requests will then be entered into STARS and will be tracked through Marketing and Lab. The System Engineer will receive notification upon receipt of the problem and when the solution has been determined.

The Software Status Bulletin is also printed by STARS in the same format as the 300 and 3000. This will provide consistency for those customers with more than one type of system. These changes should complement our continual dedication to superior support.

Users' Group Contributed Library

by Miles Kehoe

As this edition of Communicator/250 goes to press, we are preparing to deliver our internal "Contributed Library" of basic programs to the Hewlett-Packard General System Users' Group (HPGSUG) for their annual meeting in San Jose, February 24-29.

HPGSUG serves as the keeper of the HP 3000 Contributed Library, and now that the HP 250 has joined General Systems Division, plans are underway to organize a "HP 250 Users' Subgroup."

The Contributed Library is a program exchange forum where both HP employees and customers are able to share utilities, games, and other useful programs with HP 250 Users worldwide. You may find in this library many of those useful little utilities you never had time to write. In addition, by sharing your own useful programs, everyone benefits from a wide pool of resources.

The Users' Group acts independently from HP. Details of the final implementation of the Contributed Library have yet to be settled. Until then, any inquiries you may make should be directed to your sales representative. You will be kept informed through future articles in the Communicator.

A New Operating System!

by Gretchen Snowden

Operating System Revision 2.3 (Rev. F), which is being released at the time this Communicator goes to press, contains solutions to problems generally affecting European Systems. A list of the solved problems will accompany your system update package.

These changes will be completely transparent to most users of the HP 250, particularly those not using European keyboards. The next major release of the operating system should be available this summer.

Why HP Approved Media

by Rich Bassett

Even though Floppy Diskettes appear the same visually, they are in fact quite different. These differences in most cases prevent the successful use of non-HP approved media in the HP 250 Flexible Discs. Listed below are some of the distinguishing characteristics of floppy diskettes:

Characteristic	Comments
1. Single-Sided	Contains half of the storage capacity of a double-sided diskette
2. Double-Sided	Read/Write data from either side of the diskette.
3. Soft Media	Has a high Read signal amplitude, which lowers the retry and error rates. However, it is highly susceptible to media wear problems. Also, the media can wear onto the heads of the drive which will contaminate and cause wear on other diskettes.
4. Hard Media	Has a lower Read signal amplitude than the soft media, which will cause it to have a higher retry and error rate. Also, some media will cause the heads on the disc to wear.

We have evaluated most of the major suppliers of diskettes, and have found only one suitable vendor. Even with the approved vendor it is necessary for us to purchase diskettes to a specification that exceed those which they normally provide. This makes HP the only supplier of HP approved media.

The HP approved media is double-sided with a hardness specified at a level which will guarantee a reasonably long life[1] with low error rates. The use of any other non-HP approved media can cause one or more of the above problems.

[1]HP approved diskettes which are used regularly should be replaced approximately once every six months or before if signs of wear appear.