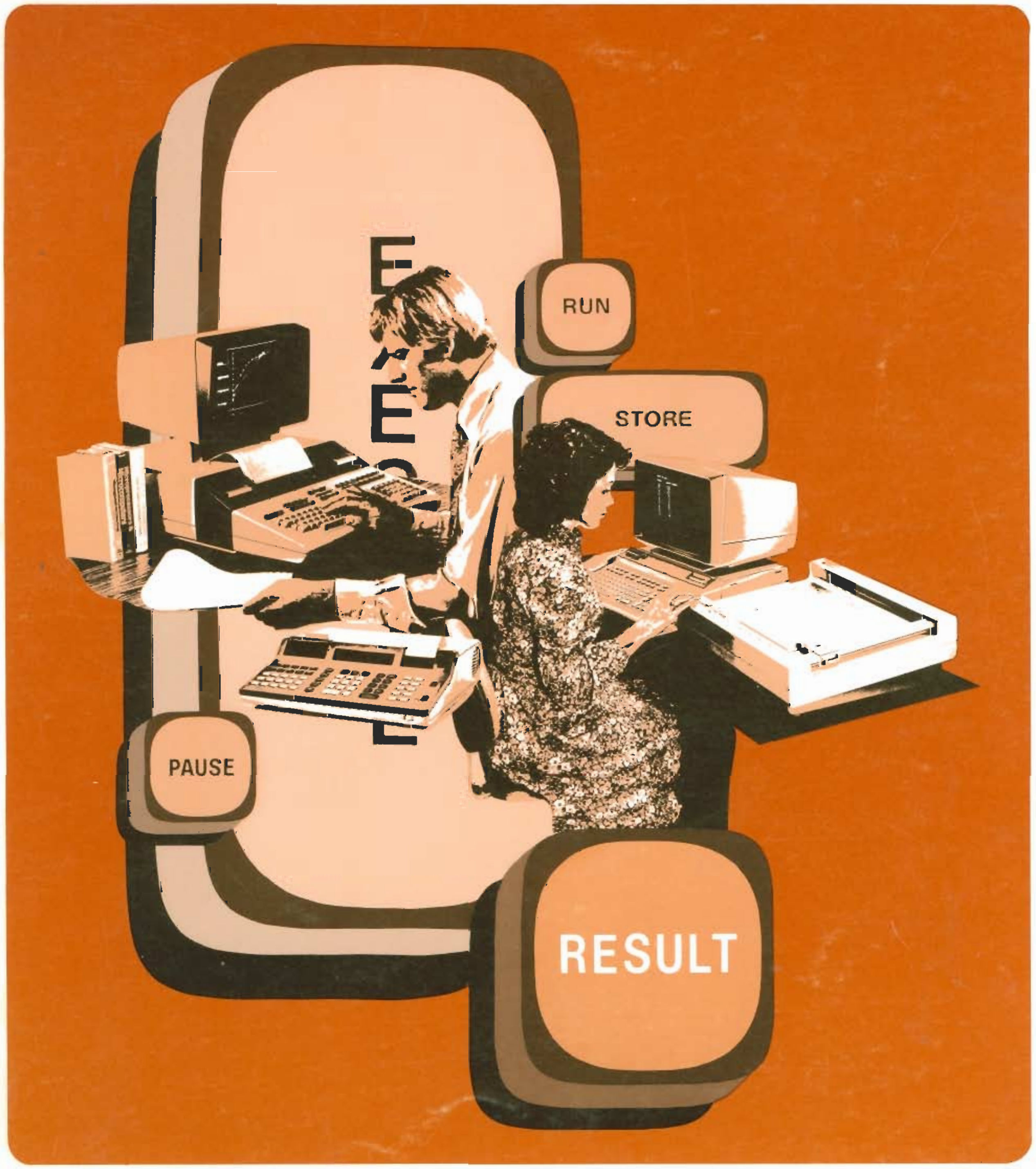
Programming Tips

For users of Hewlett-Packard 9800 Series Desktop Computers. Compiled and edited from Keyboard magazine.



Introduction

This book represents the continuing commitment of Hewlett-Packard to support all of our desktop computer users. In publishing "Programming Tips" in each bi-monthly issue of *Keyboard* magazine, we forward these ideas as they are generated. Now, in one compiled, edited presentation, you have access to all published Programming Tips for HP 9800 Series Desktop Computers. These Tips have appeared in *Keyboard* during the decade that we have published the magazine; the book is a "10th anniversary gift" to you.

The editing effort involved more than correcting typographical or other errors; duplicate Tips were combined or eliminated, and we were able to take mainframe improvements into account where they have eliminated the need for certain Tips; finally, some readers offered improvements to previously-published Tips, and these too, have been included in this book.

As is the case with all Programming Tips, these should help you to do some existing tasks with greater ease and effectiveness. They should also give you ideas about new things you can do with your computer, perhaps in areas of applications you had not considered before.

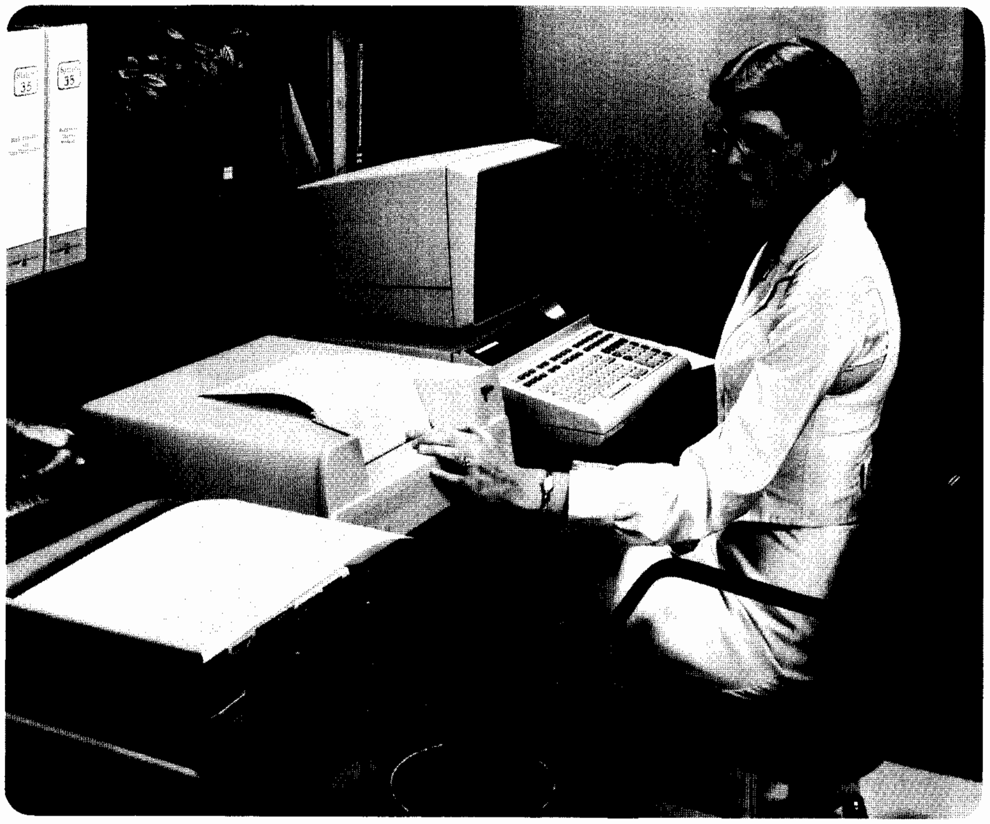
We hope you enjoy this book and reference it regularly. We have organized the Tips by mainframe to make scaning easier, and have provided an index as well. You can put the book up on your reference shelf as is, or separate the pages and keep them in a three-ring notebook, where you can add new pages from *Keyboard* as we publish more Tips.

Many people at HP's Desktop Computer Division have worked to help debug, update and compile these Tips. Others worked to typeset the copy, proofread all the material and present it to you in a visually appealing fashion. These people include: Hal Andersen, Ed Bride, Paula Dennee, Wendy Hart, Steve Hug, Brenda Hume, Donna Kimble, Martin Nielsen, Bill Sharp and Chris Stumbough.

Table of Contents

Section 1

9835 and 9845



Index

Section 1 - 9835 and 9845

Instant Success Using System 45A/B Graphics

by Donna Kimble, Hewlett-Packard Company, Desktop Computer Division

Any manager, engineer or secretary can create graphic outputs on the System 45 CRT and get printed copy from the internal printer. All you need is the Graphics Option and the Printer Option.

1. Type PLOTTER IS "GRAPHICS"
   Press EXECUTE
2. Type LETTER
   Press EXECUTE
   Type your message, using the display control arrows $\overset{\uparrow}{\underset{\downarrow}{\leftarrow\!\!\rightarrow}}$ to move the cursor to any part of the screen. To gain finer control over the cursor, press the SHIFT key and the display control arrows at the same time: this will cause the cursor to move 1/10th of a character space at a time.
   When finished typing your message,
   Press STOP
3. Type DUMPGRAPHICS
   Press EXECUTE
4. If you want another copy, repeat Step 3.
5. If you want to add more to your graphics, repeat Steps 2 and 3.
6. To draw a box around the graphics output in the CRT, add following Step 1:
   Type FRAME
   Press EXECUTE
7. If you want to create larger letters, insert before Step 3:
   Type CSIZE 6
   Press EXECUTE
   Go to Step 2.
8. If you want to letter vertically, at a 90 degree angle, insert before Step 3:
   Type DEG
   Press EXECUTE
   Type LDIR 90
   Press EXECUTE
   Go to Step 2.

You can also use this capability to add your own text to any completed plot when the output remains in the CRT. Simply begin at Step 2, *not* Step 1. Step 1 erases the previously stored graphics output.

The CSIZE statement specifies character size as a percentage of the height of the CRT. The standard is CSIZE 3.3; with CSIZE 10 the characters are 10 percent of the CRT height

The LDIR statement specifies rotation of the lettering as a counterclockwise angle from the horizontal. The standard is LDIR 0; with LDIR 180 the lettering would be upside down if the DEG (degrees) statement has been executed.

Minimize File Access Time

by Albert Brunsting, Ph.D., Solar Spectrum, Inc., Miramar, Florida U.S.A.

Minimizing file access time on a tape requires that the more frequently used files are located close to the tape's directory. As the tape is updated using HP's convenient file-by-name feature, some of the infrequently used files are relocated close to the directory. This set of file locations increases the average file access time, increases tape wear and reduces efficiency.

The following method minimizes these problems. After the tape is close to final form so that future updates and improvements on any one file are likely to be minor, rank the files according to the frequency of their use in the following manner: Let the files be named A1, A2, A3, . . . and B1, B2, B3, . . . where A1 and B1 are the most frequently used files, A2 and B2 are the next most frequently used files, and so forth.

Now place A1, A2, A3, . . . in the address locations 0-425 with enough space between them to allow for minor updates and improvementss. Likewise place B1, B2, B3, . . . in the address locations 426-852 with space between them. This can be done with a 5-record space between the files, for example, using the following technique:

1. Put the tape containing the current mixture of files, not in their optimum locations, in T14.
2. Put an initialized tape in T15, ready to record.
3. Enter: COPY "A1:T14" TO "A1" (EXEC).
4. Enter: CREATE "SKIP1", 5 (EXEC).
5. Repeat steps 3 and 4 for the files A2, SKIP2, A3, SKIP3, . . .
6. Say the last A file of length n (256 bytes/record) is located at address m. Enter: CREATE "SKIP", 425-m-n (EXEC). Now enter: CREATE "BLOCK", 1 (EXEC). This puts the data file BLOCK with just one record, at address 425. No files can now span the address locations 425-426, causing tape wear and loss of time because the tape drive moves the tape from one end to the other going from 425 to 426.
7. Now start on the second track with the B files at location 426. Enter: COPY "B1:T14" TO "B1" (EXEC).
8. Enter: CREATE "JUMP1",5 (EXEC).
9. Repeat steps 7 and 8 for B2, JUMP2, B3, JUMP3, . . .
10. Purge the files SKIP1, SKIP2, . . . and JUMP1, JUMP2, . . .

The repeat steps of 5, 9 and 10 are efficiently executed with the convenient RECALL key.

Now the files are in time-saving locations and have enough empty, adjacent records for minor updates using the HP file-by-name feature.

Label Centering

by Brad Miller, Hewlett-Packard Company, Desktop Computer Division

The LORG statement in the System 45 is a very powerful tool for lettering graphic output. However, several questions have come up when using LORG 5 (label centering). Labels do not appear to be centered! This is not because of problems with LORG 5 but rather because of the characteristics of the LABEL statement. The LABEL works like PRINT in that it puts literals (text) into 20-character fields. Therefore, LABEL "123456789012345" will be sent with 5 leading blanks and consequently not appear centered. The solution (as with PRINT) is LABEL "123456789012345"; the ; causes the literal to be sent "as is" and it will then appear centered! Happy labeling!

Better Label Centering

by Carl Johan Lamm, Lund, Sweden

I am sorry to say that Label Centering on the HP System 45 should not be done the way proposed in *Keyboard* 1978/3. Ending a LABEL statement with a semicolon will cause the field to be buffered (as well as being output). Filling the buffer will ultimately lead to a linefeed being output, as is seen by running the following program.

```
10 PLOTTER IS "GRAPHICS"
20 GRAPHICS
30 SCALE 0,20,0,20
40 FOR I=1 TO 10
50 MOVE I,I
60 LABEL "1234567890";
70 NEXT I
80 END
```

It is good practice always to use formatted output with LABEL. Change line 60 to LABEL USING "K"; "1234567890" and the program will run nicely.

READ DATA Error Recovery

by Bonnie Dykes, Hewlett-Packard Company, Desktop Computer Division

Sometimes, because of extreme use of a mass storage medium or adverse environmental conditions, the contents of a particular file may become "lost." That is, the desktop computer cannot successfully read a particular area on a tape cartridge or disc.

This READ DATA error recovery program recovers most of a "lost" file by trapping the READ DATA error and bypassing the unreadable portion of the storage medium.

The recovered data or program is recorded onto a storage medium specified by the user. The program is usable only on data files that have been SAVEd rather than STOREd.

```
10    ! THIS PROGRAM WILL HELP A USER
      RECOVER FROM A READ DATA ERROR IN A
20    ! PROGRAM FILE WHICH WAS SAVED AS
      DATA
30    COM A$(50)
40    DIM B$[160],E(10),File$[6],
      Mediumfrom$[10],Mediumto$[10]
50    PRINTER IS 16
60    PRINT PAGE,"READ DATA ERROR
      RECOVERY PROGRAM",LIN(1)
70    PRINT "NOTE:  The recovered program
      will be stored in a file called
      'Recvr' on the",LIN(1),TAB(8),
      "specified storage medium.",LIN(1)
80    PRINT TAB(8),"SOME LINES OF CODE
      WILL BE MISSING SO CHECK YOUR
      RECOVERED PROGRAM"
90    PRINT TAB(8),"CAREFULLY AND REPLACE
      ALL MISSING LINES.",LIN(1)
100   INPUT "ENTER NAME OF FILE WITH READ
      DATA ERROR",File$
110   INPUT "ENTER SPECIFIER FOR STORAGE
      MEDIUM CONTAINING FILE WITH READ
      DATA  ERROR",Mediumfrom$
120   INPUT "HOW MANY RECORDS DOES THE
      FILE WITH THE READ ERROR CONTAIN?",
      Records
130   INPUT "ENTER SPECIFIER FOR STORAGE
      MEDIUM ON WHICH TO STORE THE
      RECOVERED PROGRAM",Mediumto$
140   ASSIGN #1 TO File$&":"&Mediumfrom$
150   CREATE "Recvr"&":"&Mediumto$,
      Records
160   ASSIGN #2 TO "Recvr"&":"&Mediumto$
170   ON ERROR GOTO 230
180   E=1
190   ON END #1 GOTO 270
200   FOR N=1 TO 9999999999
210   READ #1,N;B$
220   GOTO 260
230   E(E)=N
240   PRINT N
250   E=E+1
260   NEXT N
270   READ #1,1
280   READ #2,1
290   E=1
300   ON ERROR GOTO Nextr
310   ON END #1 GOTO Out
320   READ #1;B$
330   PRINT #2;B$
340   PRINT B$
350   GOTO 320
360 Out: PRINT "RECOVERY COMPLETE"
370   BEEP
380   PAUSE
390   END
400 Nextr:    READ #1,E(E)+1
410   E=E+1
420   GOTO 320
```

Dynamic File Allocation System 35A/45A/45B

by Donna Kimble, Hewlett-Packard Company, Desktop Computer Division

In designing a system, there are times when the absolutes inherent in the language of the computer conflict with the changing reality of our problem. One such case is when we really don't know within, say, 10%, the amount of data storage space that will ultimately be needed for our application. Once a file is created with the CREATE statement, its size cannot be changed.

Let's say we're working with a mailing list for a new newsletter. If the newsletter is very successful, we might eventually need to keep track of 1000 names and addresses. If that happens, we would be happy to invest in storage space for the names, of course, but then it is also possible that the mailing list might not grow very soon. We don't want to permanently allocate space which may never be needed, nor do we want to permanently restrict the file size at the start of our system design.

In any application, we can choose a somewhat arbitrary space increment. Let's say for our mailing list we decide to increase the file size by 50 records at a time. When the first space of 50 records is full and we try to add the 51st entry, the program will automatically set up a second space of 50 records. Further, since our file is named "MAIL", we will identify the individual subsets of "MAIL" as "MAIL1", "MAIL2", "MAIL3", etc.

The following program lines show the technique required to numerically assign the names for five files.

```
FOR Subset = 1 TO 5
CREATE "MAIL"&VAL$ (Subset),50
NEXT Subset
```

The result of the above program would be "MAIL1", "MAIL2", "MAIL3", "MAIL4" and "MAIL5" created on the mass storage tape or disc. Later, we'll see how these files can be allocated as needed.

In a system, we would want to be able to check and see if the file already existed before creating it. The following program lines show the use of the ASSIGN statement return variable for checking to see if the file already exists. If the value of the return variable is 1, the file does not exist.

```
ASSIGN #1 TO "MAIL"&VAL$ (Subset), Checkword
IF Checkword = 1 THEN Nofile
```

When we print into a file, if there is no more space, we want to allocate more space. But if we are reading from the file, we need to have some way to determine, when we reach the end of a file, whether there are more file subsets or not. The following program lines determine whether a new file subset should be created depending upon a preset flag called Mode$.

```
IF Mode$ = "READ" THEN Finished
IF Mode$ = "PRINT" THEN CREATE
    "MAIL"&VAL$ (Subset),50
```

Whether reading from the file or printing in the file, when an end of file condition is detected, we want to return to the portion of the program where a new file is allocated. The following program line guarantees return to the Allocate subroutine whenever an end condition is detected.

```
ON END #1 GOSUB Allocate
```

In the following program, we allocate the first file subset at the beginning of the print routine and again at the beginning of the read routine. Part of the procedure for file allocation is to establish a return to file allocation when the end of file is detected.

This program allows keyboard entry of any number of data items, and then prints these entries from the file when data entry is complete. Of course, in actual applications, there will be other steps such as updating the file and sorting the records which are not shown here.

```
10     DIM Name$[30],Street$[30],
       City$[30],Zip$[10],Mode$[5]
20  Entry:    Subset=0
30     Mode$="PRINT"
40     GOSUB Allocate
50     LINPUT "Enter name.  To exit,
       enter END.";Name$
60     IF Name$="END" THEN Exit
70     LINPUT "Enter street address";
       Street$
80     LINPUT "Enter city and state";
       City$
90     LINPUT "Enter zip code";Zip$
100    PRINT #1;Name$,Street$,City$,Zip$
110    GOTO 50
120 Exit:     PRINT #1;"END"
130    INPUT "ENTRY COMPLETE.  DO YOU
       WANT A PRINTOUT?";A$
140    IF A$[1,1]<>"Y" THEN Finished
150 Printout:  Subset=0
160    Mode$="READ"
170    GOSUB Allocate
180    READ #1;Name$,Street$,City$,Zip$
190    PRINT USING Form;Name$,Street$,
       City$,Zip$,
200 Form:  IMAGE K,/,K,/,K,/,K,2/
210    GOTO 180
220 Finished:    ASSIGN * TO #1
230    DISP "PROGRAM COMPLETE"
240    END
250 Allocate:   Subset=Subset+1
260 Assign:   ASSIGN #1 TO
       "MAIL"&VAL$(Subset),Checkword
270    IF Checkword=1 THEN Nofile
280    ON END #1 GOSUB Allocate
290    RETURN
300 Nofile:    IF Mode$="READ" THEN
       Finished
310    IF Mode$="PRINT" THEN CREATE
       "MAIL"&VAL$(Subset),3
320    GOTO Assign
```

MAT SORT & MAT SEARCH System 35A/35B/45B

by Stephen M. Taylor, Hewlett-Packard Company, Desktop Computer Division

This programming tip tells how to use the MAT SORT and MAT SEARCH statements on mixed uppercase and lowercase strings and produce results as if all the characters were in the same case.

When doing comparisons on character strings where case is not a concern, the normal procedure is to use the UPC$ function, e.g., UPC$(A$)<UPC$(B$). However, when using the MAT SORT or MAT SEARCH statements defined by the System 35 or System 45 Advanced Programming ROM, this technique is not available. One could use a FOR/NEXT loop to go through and UPC$ all the strings in the array to be sorted or searched, but that would sacrifice the original form of the data.

Let's say, for example, that you have typed in data for a list of names that included ADAMS, WOTTEN, MACMAHON, ZHUKOV, and GREENOUGH. A short time later, another person adds names that include Thomas, Galerius, Cowley, Seward, MacKaye and Machiavelli. You typed names entirely in uppercase letters, while the second person added names with initial uppercase letters.

This is not a problem with a short list, as you can change them with little difficulty. However, if there are hundreds of names added to the list, it can be a problem if you try to sort them alphabetically. With a standard sort of the names above, they would print out:

```
ADAMS
Cowley
GREENOUGH
Galerius
MACMAHON
MacKaye
Machiavelli
Seward
Thomas
WOTTEN
ZHUKOV
```

The reason for this is that uppercase letters appear before lowercase in the ASCII character set, and receive priority. Even if both of you type with initial uppercase letters, the names MacKaye and MacMahon would apear before Machiavelli in a simple sort, because of the uppercase letters in the names.

The solution to this problem is to use the LEXICAL ORDER IS statement with an appropriately modified lexical order table, one in which the upper and lowercase letters have the same sequence numbers. The first program given below will create such a table from the 'ASCII' table on the cassette that comes with the Advanced Programming ROM. The second program shows how to set up the LEXICAL ORDER from that table. The third program, with output, shows the results of doing a MAT SORT with this LEXICAL ORDER.

Create Table

```
10  OPTION BASE 1
20  INTEGER Table(354)
30  ASSIGN #1 TO "ASCII"
40  MAT READ #1;Table
50  FOR I=0 TO 25
60  Table(100+I)=Table(68+I)
70  NEXT I
80  CREATE "UPC$LT",8
90  ASSIGN #2 TO "UPC$LT"
100 MAT PRINT #2;Table
110 END
```

You should note that with languages other than English, the integer table (Create table, line 20) would have to be larger. Designing the program to operate correctly would require referring to the Advanced Programming ROM manual.

Set Up LEXICAL ORDER

```
10  CALL Table_set_up
20  !
30  ! USER PROGRAM
40  !
50  END
60  SUB Table_set_up
70  OPTION BASE 1
80  INTEGER Table(354)
90  ASSIGN #1 TO "UPC$LT"
100 MAT READ #1;Table
110 LEXICAL ORDER IS Table(*)
120 SUBEND
```

MAT SORT With LEXICAL ORDER

Here we'll insert our quasi data base to demonstrate that the MAT SORT with this LEXICAL ORDER will disregard the case of the letters and print out the names in alphabetical order, preserving their original form.

```
10   DIM String$(10)
20   DATA ADAMS,WOTTEN,
          ZHUKOV,GREENOUGH,
                 MACMAHON
30   DATA Thomas,Galerius,
      Cowley,Seward,MacKaye,
                Machiavelli
40   MAT READ String$
50   MAT SORT String$
60   MAT PRINT String$
70   END
```

```
ADAMS
Cowley
Galerius
GREENOUGH
Machiavelli
MacKaye
MACMAHON
Seward
Thomas
WOTTEN
Zhukov
```

Transferring Data From 9830 to System 45

by Martin Nielsen, Hewlett-Packard Company, Desktop Computer Division

The 9830/31 to System 45 Translator package, part number 11141-10090, transfers only programs, no data. However, it is fairly simple to write a program to transfer a specific data format, providing you know a few simple things:

1. You need to have an I/O ROM (or binary) for the System 45.
2. You need the 98032 Opt. 30 cable for transfer from the 9830 to the System 45.
3. For input to the System 45, you must use the following sequence prior to any actual data transfer:

   S = <select code of cable (usually 12)>
   CONTROL MASK S;1
   WRITE IO S,5;1

   If you want to transfer data from the System 45 to the 9830, use CONTROL MASK S; 0 and WRITE IO S,5;0.
4. Read your data into the 9830 from the tape (or disc, or paper tape, or wherever it's stored).
5. Write the data from the 9830 to the interface cable as though it were a printer:

   R = <select code of cable (usually 1)>
   WRITE (R, *) <data list>
6. On the System 45, use an ENTER statement whose data list matches the output from the 9830.

For the 9830 example:

Line 30 reads the data from the tape cassette into memory.

Lines 40 through 70 send the data across the cable to the System 45.

For the System 45 example:

Lines 30 through 50 mark enough files to hold the data.

Lines 60 through 80 configure the interface for input to the System 45.

Lines 90 through 150 accept the data from the 9830 and print it on the tape drive.

Press RUN, EXECUTE on the 9830, and press RUN on the System 45. The System 45 will mark 51 tape files (0 through 50) and then will start accepting data from the 9830. The 9830 will be forced to wait at line 40 until the System 45 executes line 110.

To transfer data from the System 45 to the 9830, change the CONTROL MASK S; 1 in line 70 of the System 45 listing to CONTROL MASK S;0. Also change line 80 to read WRITE IO S,5;0. Then change all the ENTERs in the System 45 program to OUTPUTs, and change all the OUTPUTs in the 9830 program to ENTERs. Also change the sections accessing the tapes accordingly.

Example: 9830

```
10 COM A,B,CI,D$[50],ES[20,20]
11 S=1
20 FOR I=0 to 50
30 READ DATA I
40 WRITE (S,*) A,B,C,D$
50 FOR J=1 to 20
60 FOR K=1 to 20
70 WRITE (S,*) E[J,K];
80 NEXT K
90 NEXT J
100 NEXT I
110 END
```

Example: System 45

```
10 OPTION BASE 1
20 COM A,B,INTEGER C,D$[50],
   SHORT E(20,20)
30 FOR I=0 to 50
40 CREATE "F"&VAL$(I),7
50 NEXT I
60 S=12
70 CONTROL MASK S;1
80 WRITE IO S,5;1
90 FOR I=0 to 50
100 ASSIGN #1 to "F"&VAL$(I)
110 ENTER S; A,B,C,D$
120 ENTER S; E(*)
130 PRINT #1;A,B,C,D$,E(*)
140 DISP I; "TRANSFERRED"
150 NEXT I
160 BEEP
170 DISP "DONE"
180 END
```

Character Slant in a System 45A/9872 System

by Rita Wigglesworth, Hewlett-Packard Company, Desktop Computer Division

Characters drawn via the LABEL, LABEL USING or LETTER statements are defined within the System 45A Graphics ROM and cannot be slanted. To slant characters, use the 9872A as a printer and send HPGL slant and label instructions. An example is shown below. Use FIXED 4 format to send the tangent of the slant angle. If the tangent is so large as to require scientific notation, the plotter will generate an error when it encounters the "E". For reasonable slant angles, this problem should not occur.

To produce the "${}^{E}{}_{X}$" character, press the CONTROL key and the leter "C" at the same time.

This program is not needed for the System 45B because the slant function is included in the 45B Graphics ROM.

```
10 DEG
20 FIXED 4
30 PRINTER IS 7,5
40 FOR Angle=-70 TO 70 STEP 10
50 PRINT "SL";TAN(Angle)
60 PRINT "LBH^E_X "
70 NEXT Angle
80 END
```

Continuous Plots Using System 45A/B Graphics

by Donna Kimble, Hewlett-Packard Company, Desktop Computer Division

In applications where strip chart recorders have been used in the past, sophisticated outputs can be obtained using the System 45. To combine the power of the System 45 and its graphics capability with the familiar strip printer output, we can use the commands available in the System 45 Graphics ROM.

If I can present to you a sine curve, plotted continuously on the CRT and dumped to the internal printer, can you translate the technique to your own data? And can you add the labeling which you need?

As I asked these questions of my students who needed this kind of solution, I got an unqualified "yes" in response.

In writing this program, I used variables at lines 20, 30 and 40 so that you could see the interrelationships involved more easily, and so you could also plug in different values to be sure the basic concept works in a variety of cases. Initially, we will plot the number of cycles of the sine wave (Count = 3), where one cycle is 360 degrees (Cycle = 360).

We will dump the partial plot to the printer after each specified interval (Interval = 15 degrees). You could dump more often, say, after each 5 degrees, or less often, say, after each 180 degrees. The program listed on this page plots SIN(X)/X in a continuous form.

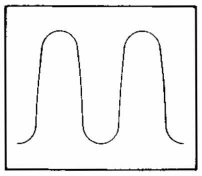
For a continuous plot, we need to visualize a transposed picture. The X-axis is normally assigned to the horizontal component with the Y-axis on the vertical. The origin of a plot (0,0) is normally in the lower left corner of the CRT.

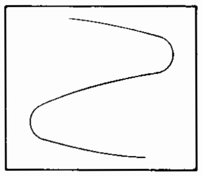
We will want to use the *width* of the paper output to represent the Y-axis, and the *length* of the paper output to represent the X-axis. This effectively rotates our picture 90 degrees so that the origin (0,0) can be in the upper left corner of the CRT. The SCALE statement at line 70 presents the CRT area not only rotated clockwise by 90 degrees, but with our new Y-axis scaled to meet the needs of the problem, where Ymin = −1 and Ymax = +1.

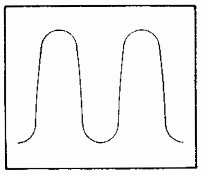
Once we have plotted a section of our picture to the CRT, the DUMPGRAPHICS statement at line 130 allows us to copy the picture onto the printer. This statement also allows us to vary the amount of information copied to the printer depending upon the portion of the CRT actually used for the plot.

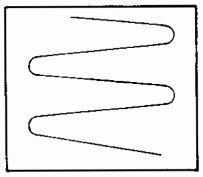
As we exceed the scaling for our next X-axis, we effectively offset our X-values to fit the original scaling by using the MOD function at line 100. For example, at 16 degrees we can correct our data to plot at 1 degree; 17 degrees corrects to plot at 2 degrees and so on. We reverse the XY coordinates in our PLOT statement at line 100 because the *dependent* variable, normally the Y-value, moves with the *width* of the CRT, which is normally thought of as the X coordinate.

```
10  DEG
20  Interval=15
30  Cycle=360
40  Count=3
50  PLOTTER IS "GRAPHICS"
60  GRAPHICS
70  SCALE -1,1,360,0
80  FOR I=0 TO Cycle*Count
90  IF NOT (I MOD Interval) THEN
    GOSUB Dump
100 PLOT SIN(I), I MOD Interval
110 NEXT I
120 END
130 Dump: DUMP GRAPHICS Interval,0
140 GCLEAR
150 PENUP
160 RETURN
```


Normal plot on CRT


Continuous plot on CRT


Normal DUMPGRAPHICS


Continuous DUMPGRAPHICS

9862 BASIC Language Drivers

by Dave Page, Hewlett-Packard Company, Desktop Computer Division

In answer to a number of requests, a set of BASIC language drivers has been written by Pierre Daubine of HP to allow using the 9862A with a System 45 or 35.

The drivers are simply a set of BASIC subprograms that can be appended to your main program and used with CALL statements. The only big drawback is that they don't label.

Here are the available subprograms:
CALL Scale (Xmin, Xmax, Ymin, Ymax)
CALL Penup
CALL Plot (X,Y,P) (P has the same meaning as in the System 45 graphics ROM)
CALL Move (X, Y)
CALL Draw (X, Y)
CALL Xax (Yvalue, Xticspacing, Xstart, Xend)
CALL Yax (Xvalue, Yticspacing, Ystart, Yend)

To use these subprograms, you must reverse the first six numbers in COMMON for use by the subprograms:
COM (Xmin, Xmax, Ymin, Ymax, Scalex, Scaley)

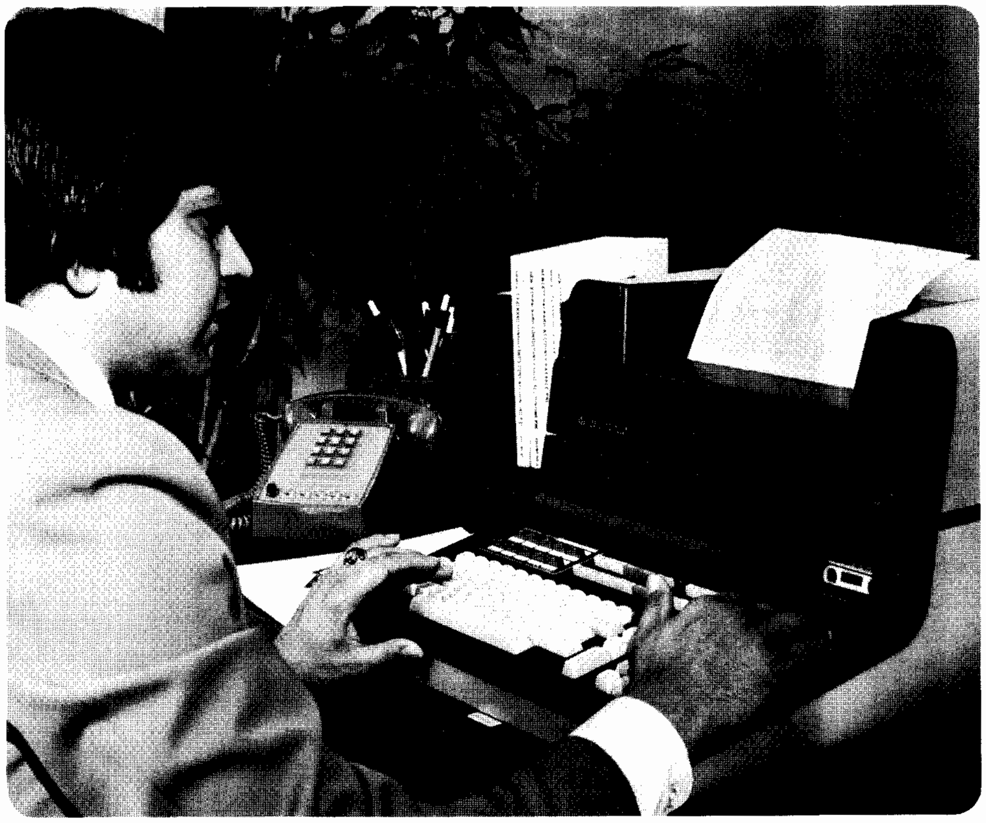
It is not necessary to set these numbers in the main program; the Scale subprogram handles that. It is necessary to declare the COMMON area in the main program, and furthermore, if there are other things in COMMON, these six must be the first ones.

```
5920 ! The following routines are
       available to drive the 9862 from
       the System45
5930 !      Scale (Xmn,Xmx,Ymn,Ymx)
5940 !      Penup
5950 !      Plot (X,Y,P)
5960 !      Move (X,Y)
5970 !      Draw (X,Y)
5980 !      Xax(Yintercept,Xtic,Xstart,
            Xend)
5990 !      Yax (Xintercept, Ytic, Ystart,
            Yend)
6000 SUB Scale (Xmn,Xmx,Ymn,Ymx)
6010 COM Xmin,Xmax,Ymin,Ymax,Echx,Echy
6020 Xmin=Xmn
6030 Xmax=Xmx
6040 Ymin=Ymn
6050 Ymax=Ymx
6060 Echx=9999/(Xmax-Xmin)
6070 Echy=9999/(Ymax-Ymin)
6080 SUBEXIT
6090 SUB Penup
6100 WRITE IO 5,6;20480
6110 WRITE IO 5,7;1
6120 SUBEXIT
6130 SUB Plot (X,Y,P)
6140 COM Xmin,Xmax,Ymin,Ymax,Echx,Echy
6150 Xplot=(X-Xmin)*Echx
6160 Yplot=(Y-Ymin)*Echy
6170 IF P=0 THEN P=1
6180 Control=28672
6190 IF P/2=INT(P/2) THEN Control=20480
6200 IF P>0 THEN Mopa
6210 CALL Pa(Control)
6220 CALL Mo(Xplot,Yplot)
6230 SUBEXIT
6240 Auto:CALL Mo(Xplot,Yplot)
6250 STATUS 5;Status
6260 IF NOT BIT (Status,8) THEN CALL
     Pa(28672)
6270 !
6280 Mopa:CALL Mo(Xplot,Yplot)
6290 CALL Pa(Control)
6300 SUBEXIT
6310 SUB Pa(Control)
6320 IF NOT IOFLAG(5) THEN 6320
6330 WRITE IO 5,6;Control
6340 WRITE IO 5,7;1
6350 SUBEXIT
6360 SUB Mo(Xmove,Ymove)
6370 P3=INT(Xmove/100)
6380 P4=INT(Ymove/100)
6390 P1=100*(Xmove/100-P3)
6400 P2=100*(Ymove/100-P4)
6410 STATUS 5;Status
6420 P5=BIT(Status,8)
6430 P5=8192*P5+256*NOT P5
6440 !
6450 WRITE IO 5,6;16384+P5
6460 WRITE IO 5,4;16*INT(P3/10)+10*
     (P3/10 -INT(P3/10))
6470 WRITE IO 5,7;1
6480 CALL Output (P1/10)
6490 CALL Output(P4/10)
6500 CALL Output(P2/10)
6510 SUBEXIT
6520 SUB Output(C)
6530 IF NOT IOFLAG(5) THEN 6530
6540 WRITE IO 5,6;P5
6550 WRITE IO 5,4;16*INT(C)+10*
     (C-INT(C))
6560 WRITE IO 5,7;1
6570 SUBEXIT
6580 SUB Xax(Y0,Xtic,Xdeb,Xfin)
6590 COM Xmin,Xmax,Ymin,Ymax,Echx,Echy
6600 CALL Penup
6610 IF NOT Xtic THEN Xax1
6620 FOR Xaxe=Xdeb TO Xfin STEP Xtic
6630 CALL Plot(Xaxe,Y0,0)
6640 CALL Plot(Xaxe,Y0-60/Echy,0)
6650 CALL Plot(Xaxe,Y0+60/Echy,0)
6660 CALL Plot(Xaxe,Y0,0)
6670 NEXT Xaxe
6680 CALL Penup
6690 SUBEXIT
6700 Xax1:CALL Plot(Xdeb,Y0,0)
6710 CALL Plot(Xfin,Y0,0)
6720 GOTO 6680
6730 SUB Yax(X0,Ytic,Ydeb,Yfin)
6740 COM Xmin,Xmax,Ymin,Ymax,Echx,Echy
6750 CALL Penup
6760 IF NOT Ytic THEN Yax1
6770 FOR Yaxe=Ydeb TO Yfin STEP Ytic
6780 CALL Plot(X0,Yaxe,0)
6790 CALL Plot(X0-40/Echx,Yaxe,0)
6800 CALL Plot(X0+40/Echx,Yaxe,0)
6810 CALL Plot(X0,Yaxe,0)
6820 NEXT Yaxe
6830 CALL Penup
6840 SUBEXIT
6850 Yax1:CALL Plot(X0,Ydeb,0)
6860 CALL Plot(X0,Yfin,0)
6870 GOTO 6830
```

```
6880 SUBEXIT
6890 SUB Move(X,Y)
6900 CALL Plot(X,Y,-2)
6910 SUBEXIT
6920 SUB Draw(X,Y)
6930 CALL Plot(X,Y,-1)
6940 SUBEND
```

Section 2

9830



Index

Section 2 - 9830

Random Number Generation

by Philip Dawdy, Lansing Community College, Lansing, Michigan

I have discovered a technique for "continual randomization" of random numbers generated from the 9830. Normally, the 9830 will generate a sequence of random numbers (via the RND(0) function) from a calculated seed ($2-\pi/2$) or another seed if specified by the user.

When programs are run, they are initialized before execution. This initializing process causes the random sequence to begin from the 9830's seed unless the program changes the seed. If the program uses the 9830's seed, or changes it using the same negative number within the parentheses of the function, the sequence will be the same every time the program is run.

The following method eliminates any need for an extra dummy entry and automatically generates a new sequence of random numbers each time the program is restarted. The only way the same sequence will result is from calculator turn-on restarts. When the calculator is first turned on, the sequence will begin at the same point, but each time the program is rerun a new sequence begins.

Lead the program with the following statement:

```
20 DISP RND(-0.12374536789-ABSRES 0.01)
or
20 DISP TAB32;RND(...
```

The above statement generates a seed for the random sequence and produces a new seed when the program is rerun. The reason is that the seed is calculated from the RESULT register, and it is the only register in the 9830 that is not made undefined when the program is initialized. In fact, the RESULT register is unaltered except during keyboard calculations and when SCRATCHA is executed. Another most unusual situation where RESULT is altered is in programs. Since the 9830 will not allow, for example, 30 RES = 5 x A, I was forced to find another method for storing values in RESULT via the program.

Experimentation told me that printing or displaying values are all stored in RESULT. The reason for the display of the random seed in the above statement should be clear to you now. RESULT will contain a different value each time the program is executed. If the calculator is used for keyboard calculations between runs, this alters the random sequence also, since the seed is then altered (another form of randomization).

To assure that RESULT does not contain a number from a print or display statement, a final random number is generated at the end of the program and placed in RESULT. The following statement should be placed before the program END (or STOP):

```
190 DISP RND0
200 END
or
190 DISP TAB32;RND0
200 END
```

To keep the program from starting with the same sequence every time the program is first loaded into memory, the user can do an arbitrary calculation prior to running the program (or just key in any random number and press EXECUTE).

Random Number Generation

by Professor Stanley Deming, University of Houston, Houston, Texas

I enjoyed the tip on random number generation on the 9830A by Philip Dawdy in Vol. 8, No. 1. Our laboratory makes use of randomized experimental designs and has a need for generating different variable-size sets of random numbers.

The program below generates a randomized set of a given number of values. The only major change in the original method of generating random numbers is to test for RES=0. If it is true, the program prompts the operator to supply a different seed.

```
10 DIM S[100]
20 IF RES=0 THEN 90
30 PRINT "PLEASE TYPE IN A NUMBER <1000"
40 PRINT "PRESS 'EXECUTE'"
50 PRINT "PRESS 'RUN'"
60 PRINT "PRESS 'EXECUTE'"
70 PRINT
80 GOTO 350
90 DISP TAB32;RND(-0.123456789-ABS(RES)*0.0003)
100 DISP "HOW MANY NUMBERS TO RANDOMIZE";
110 INPUT S3
120 IF S3=0 THEN 330
130 PRINT
140 PRINT "NUMBERS TO BE RANDOMIZED = ";S3
150 PRINT
160 REDIM S[S3]
170 FOR S=1 TO S3
180 S[S]=S
190 NEXT S
200 FOR S=S3 TO 1 STEP -1
210 S1=INT(S*RND(0)+1)
220 S2=S[S]
240 S[S1]=S2
250 NEXT S
260 FOR S=1 TO S3
270 WRITE (15,280)S[S];
280 FORMAT 10F5.0
290 IF S/10=INT(S/10) AND S=S3 THEN 310
300 PRINT
310 NEXT S
320 PRINT
330 DISP TAB32;RND(0)
340 DISP "END"
350 END
```

A Tip On Faulty Cassette Tapes

by Ian Collier, Hewlett-Packard, Melbourne, Australia

If a tape becomes unusable because of an oxide fault quite close to the start of it, then careful disassembling, turning the tape over and reassembling will make the majority of the tape usable again without having any differences discernible to the operator.

On the Efficiency of the POS Function

by Donna Kimble, Desktop Computer Division, Hewlett-Packard

Is September the ninth month? That question may surprise some people, because it seems universal that January is the first month, February the second, and so on. But the answer depends on who you ask. In the context of the fiscal year, September might be the third month or the eleventh month.

I came across a problem of this type recently, and I thought at the time that my solution was routine. But, being a fairly avid reader of other people's programs, I came across three lines buried in the middle of one program which put the entire situation in a new light. Here are those lines:

```
120 FOR P = 1 TO 10
130 IF C$(P,P) = A$(1,1) THEN 180
140 NEXT P
```

To make this situation a little clearer, I will quote other portions of the same program. 50 C$ = "0123456789" established ahead of this section a set of allowable numeric digits. And 180 V = VAL(A$) comes after.

A certain string, called A$, contains unknown data. The above section of the program was designed to avoid a non-numeric argument during conversion of the string to a numeric type data.

Consider the following line:

```
120 IF POS (C$,A$(1,1)) THEN 180
```

I believe this alternative line can replace the three lines previously used with no change in the function of the program except for a significant improvement in speed, as well as an improvement in the amount of memory used by the program.

All this brings me to that problem I had recently, in which there was quite a bit of confusion possible in referring to months by number. It seemed to me to be too risky in matters pertaining to dollars to arbitrarily decide that the user of my program should change his ways to conform to my standard. And no matter whether I decided to call January or November month one, I would be making such a demand on at least some of the managers in my department.

Using the String Variables ROM, I allowed instead that the answer to the questions pertaining to months could be the month name. In the program I could then with a clear conscience use whatever month number to refer to the month that I chose by using the POS function.

Because I had to include this facility in a number of programs, I have "optimized" it. You may find that with little modification it can be incorporated into your own programs. Here it is:

```
10 DIM M$(39)
20 M$ = "NOVDECJANFEBMARAPRMAYJUNJULAUG
   SEPOCT"
30 DISP "MONTH"
40 INPUT M$(37,39)
50 M=(POS(M$(1,36),M$(37,39))+2)/3
60 IF NOT M THEN 30
```

This program is optimized so that it requires only one string and takes the least amount of memory possible for variable storage. Also, it includes some protection against ambiguous answers from the keyboard. If a month is not recognized, the question is repeated.

It would have taken an incredible number of programming steps to accomplish this translation from month by name to month by number if the approach had been similar to that taken in the three lines at the beginning — but I have seen this done.

Just for fun, it would be interesting to see by how many steps the program could be expanded if Line 50 were replaced by some group of statements including a FOR and NEXT loop. Just for fun, of course.

Reading 5-Hole Telex Tape

by Lloyd Stott, Hewlett-Packard, Melbourne, Australia

This short program enables the 9830 to read 5-hole telex tape. Equipment required is the HP 9863A Paper Tape Reader, HP 11272B Extended I/O ROM, and HP 11274B String Variables ROM. Decimal equivalent codes for each shift mode can be worked out from Lines 30 and 40, where "T" and "5" are 1 in each case (not 2). Shift mode change characters are not included in the line or output (hence, Lines 160 and 190 to restore the actual character count). Line 70 logically ends a byte from tape with the expected 5 bits ($31_{10} = 37_8 = 11111_2$). I.e., the unwanted 3 holes of the 8-hole system are masked out. One is added to the masked character to give X.

```
1 REM:  PROGRAM TO READ 5-HOLE TELEX TAPE
   (L.STOTT/1.75)
10 DIM A$[75],N$[40],L$[40],T$[10]
20 T$="LETTERS"
30 L$=" T O HNM LRGIPCVEZDBSYFXAWJ UQK "
40 N$=" 5 9 #,. )4$80:=3+ ?'6%/-2 71(  "
50 A$=""
60 FOR I=1 TO 72
70 X=BIAND(RBYTE7,31)+1
80 IF X=28 THEN 150
90 IF X=32 THEN 130
100 IF X=9 THEN 130
110 IF T$="LETTERS" THEN 210
120 IF T$="NUMERALS" THEN 230
130 PRINT A$
140 GOTO 50
150 T$="NUMERALS"
160 I=I-1
170 NEXT I
180 T$="LETTERS"
190 I=I+1
200 NEXT I
210 A$[I,I]=L$[X,X]
220 NEXT I
230 A$[I,I]=N$[X,X]
240 NEXT I
250 END
```

Conserving Core Memory

by 1st Lt. Richard Virost, U.S. Air Force Environmental Health Laboratory, Kelly Air Force Base, Texas

I have found the following algorithm useful in conserving core memory when using large arrays in which each element is positve and has only four significant digits at most. The algorithm permits storage of such numbers in an integer array rather than a split or full precision array. The range of numbers that can be stored using this algorithm is $1x10^N<X<1x10^{N+9}$ where N is any integer — positive, negative, or zero — so that decimal numbers can also be stored in the integer array. To store X, use these steps:

```
10 DIM GI[10]
20 INPUT N
30 INPUT X
40 R=INT(LGTX)
50 IF R>N+2.5 THEN 70
60 R=N+3
70 G[1]=(-1)*X*10↑(3-R)+(R-(N+5))*10000
```

X is now stored in a coded form in G(1).
To recover X, use these steps:

```
80 Y=INT(G[1]/10000)+N+6
90 X=(-1)*(10)↑(Y-3)*(G[1]-(Y-N-5)*1000)
```

Calculations During Input

by A. de Faro Barros, GESPO, Porto, Portugal

For making some calculations during any input step without losing the trail of the program, I suggest the following routine:

• • •

```
110 DISP "NUMBER OF ITEMS (B)";
120 INPUT A
130 GOSUB 2010
140 GOTO Z OF 110
150 B=A
```

• • •

```
2010 Z=0
2020 IF A=9999 THEN 2050
2030 STOP
2040 Z=1
2050 RETURN
```

• • •

The input of a special number (e.g. 9999) throws the machine into calculation mode, returning to the same display line when required.

For not losing the results of your calculations and to keep a record of it, as soon as you enter into STOP (line 2020), press PRINT ALL (on), make your calculations, again press PRINT ALL (off), CONT, and EXECUTE. You now can contine programming.

S.F. Key Programs Used in Program

by K.S. Wilkinson, Wellington, New Zealand

To use Special Function key programs — defined by Math Pac or other cassettes — in programs rather than manually, first load the keys from the cassette, then store the required key programs one at a time in separate files on another tape (HP 9830A Operating and Programming manual, p. 6-7). Load the separately filed programs into the main memory in sequence, chaining them together. Replace the END statements with RETURN, and call the programs as subroutines. (Subroutines rather than functions must be used to pass several variables back to a mainline program.)

Eliminating RND Predictability

by Robert Campanini of BHP, Central Research Laboratories, Shortland, Australia

Each time the same seed is set into the random number generator (as is the case when the calculator is switched on or RUN is executed), the sequence of numbers which follows from the function RND is the same.

In applications which require new sequences of numbers each time a program is run (e.g., in generating systems of n random points in two dimensions), the following technique has been found useful:

1. To one of the SPECIAL FUNCTIONS keys of the program the following statements are assigned:
   10 R = RND0
   20 R = RND (−R)
   30 GOTO 20
2. Program variables are initialized by pressing RUN and appropriate SPECIAL FUNCTIONS key.
3. The SPECIAL FUNCTIONS key containing the statements in (1) is pressed.
4. After an arbitrary length of time the STOP button is pressed.
5. The body of the program is executed via the relevant SPECIAL FUNCTIONS keys.

The advantage of this technique is that it is parameter free, i.e., the sequence of random numbers produced is not determined by any input parameter.

Sorting and Pairing Numbers

by Andrew Zinn, Scott Wulfe, and Jack Ligon, Robert E. Lee High School, San Antonio, Texas

We have an idea for sorting four-digit numbers on the 9830 and, if desired, pairing them with alphanumeric data. This could be used in class ranking programs, for example.

First, all data must be in a similar range; i.e., between 0 and 1, 1 and 10, etc. Next, using the first four significant digits, enter the data element as a line number (3.459 would be entered as Line 3459). Type in some dummy statement (the program will never be run, so the statement sould be as short as possible to conserve memory), or, if alphanumeric data is to be sorted, the following statement, for example, would suffice:

2459 A$ = "JANE SMITH"

Press END OF LINE and continue entering data in this fashion. When all data are entered, merely LIST the program lines to print out the data elements in order from least to greatest. If alphanumeric data are included, executing REN 1, 1 will produce a listing of the data, in order from least to greatest, with the first line being 1 and all lines consecutive integers, when LIST is executed.

Obtaining Non-Keyboard Characters

by Robert J. Rahmann, Goonyella Mine, Queensland, Australia

Non-keyboard characters can be placed on Special Function keys of the 9830A very simply if you have an external plotter ROM. Even without the ROM square brackets can be entered. First ensure the ROM is in the central slot of the five ROM slots, then proceed as follows:

- Fetch a Special Function key — FETCH F_0
- Type in a legal array statement — 1A (1,1) = 1
- Press END OF LINE and ↓ (display viewing key)
- Edit the display 1A (1,1) = 1 to *(or *) and press END OF LINE.

For the rest of the characters, begin as above or use the brackets placed on the keys to enter, while in key mode,

(95,1)−1

The characters outside the brackets are unimportant. The first number inside the brackets is the ASCII code for the symbol required; 95 is the code for an underscore.

Press: EXECUTE
Press: RECALL
The display should read: (95,1) 1
Edit to place *⊦ on the key.

The non-keyboard characters, including line feeds, etc., can now be included in WRITE, PRINT, and FORMAT statements. Code 162 produces a quote on the display and the 9866A printer, but it does not terminate the quote field of a print statement or string variable assignment. In some programs, use can be made of the fact that the operator cannot normally enter these characters. They can be used, for instance, to separate substrings of keyboard characters within a string.

Obscure Uses of the RES Register

by William Zehner, Seascope Electronics, Lynn Haven, Florida

The thrifty 3-line program below causes the 9830A to display a running balance for use in balancing your checkbook, etc.

```
10 INPUT A
20 DISP A + RES;
30 GOTO 10
```

Its operation may not be obvious. The register called RES always contains the last number displayed or printed. Hence, line 20 is really equivalent to two operations that accomplish the running summation DISP A + RES; RES = (RES + A).

If you are short of variable names or memory, you can use the RES register for temporary storage. For example, to interchange the values of two variables without the use of a third temporary variable,

```
10 INPUT A,B
20 DISP A      (saves A in RES)
30 A = B
40 B = RES
50 GOTO 10
```

Another interesting use for the RES register is to pass a number from one program to another through a LOAD (file) operation. Unlike other variables, the value in RES is not altered by RUN, LOAD, INITIALIZE, SCRATCH, SCRATCHK, or SCRATCHV. It is only altered by SCRATCHA or by turning the machine off and on (both of which initialize RES to 0), or by any operation resulting in a number being displayed, printed or sent to a peripheral device. Hence, to pass the value of a variable, say X, from program A to program B, the last lines executed in program A should be

```
 990 DISP X      (saves X in RES)
1000 LOAD 22, 10
```

and the first line in program B should be

```
10 X = RES       (replaces RES into X)
```

Incidentally, I wish to thank Mr. Rahmann for his ingenious KEYBOARD tip on obtaining non-keyboard characters. I found it very useful in writing ALGOL programs in TEXT mode, because of the frequent need for square brackets [ ] and the \, which is used for multiplication.

Outputting Non-Keyboard Characters

by Dennis Eagle, Hewlett-Packard, Desktop Computer Division

There are times when it is desirable to be able to output characters which are not on the keyboard. For example, you might like to print 80 underscores across a page. If you had the underscore character, the problem could be solved by using a format statement in the following manner:

```
10 WRITE (15,20)
20 FORMAT 80"—"
```

Unfortunately, there is no underscore on the 9830A's keyboard. There is also no \, [,], line feed, or typewriter operations such as tab, backspace, etc. on the keyboard.

If you have a 9880 Mass Memory System, you can obtain the underscore and other non-keyboard characters as follows. First, execute the following instructions.

1. PRESS: FETCH
2. PRESS: f_0
3. TYPE: 1 DEF FNA (X)
4. PRESS: END OF LINE
5. TYPE: 2 STOP
6. PRESS: END OF LINE
7. END

Execute instructions 1 through 7 again, except that in instruction 2, press f_1. Execute instructions 1 through 7 for f_2, f_3, and so on up to f_9. Be sure that the two lines of programming are stored in every key. Next, key in the following program:

```
10 DIM A$[80],Q$[1]
20 X=34
30 DBYTE X,Q$
40 A$="1GETKEY CHARKY"
50 A$[8,8]=Q$
60 A$[15,15]=Q$
70 FILES XXX
80 PRINT #1;A$, "2END", "1RUN","1X=FNA1"
90 FOR N=0 TO 9
100 DISP "ASCII CODE";
110 INPUT X
120 IF X<0 OR X>127 THEN 180
130 DBYTE X,A$
140 A$[2]=A$
150 A$[1,1]="*"
160 PRINT #1;"1RUN","1DEL",A$
170 NEXT N
180 FOR N=N TO 9
190 PRINT #1;"1RUN","1""*"
200 NEXT N
210 PRINT #1;END
220 DGET"XXX"0
```

TYPE: SAVE KEY "CHARKY"
PRESS: EXECUTE
TYPE: OPEN "XXX',1
PRESS: EXECUTE
TYPE: SAVE "CHAR"
PRESS: EXECUTE
PRESS: RUN
PRESS: EXECUTE
ASCII CODE? will be displayed.
ENTER: 10
PRESS: EXECUTE
ASCII CODE? will again be displayed.
ENTER: 95
PRESS: EXECUTE
ASCII CODE? will be displayed. This time, terminate the program by entering 999.
ENTER: 999
PRESS: EXECUTE

The Mass Memory will make a few "clicking" sounds. 10 and 95 are the ASCII codes for line feed and underscore. If you press f_0 three times, ⊦ ⊦ ⊦ will be displayed. Although the character is displayed as a ⊦, it will print as an underscore. Now whenever you want an underscore, you can press f_0.

TYPE: PRINT "ABC
PRESS: f_0
TYPE: DEF"
PRINT: "ABC⊦DEF" will be in the display.
PRESS: EXECUTE
ABC_DEF will be printed.

As another example,

TYPE: 1 FORMAT 5"*⊦","TEST",5"⊦ ⊦"
(using the f_0 key for ⊦)
PRESS: END OF LINE
TYPE: WRITE (15,1)
PRESS: EXECUTE

Your printout should look like this:

```
*_*_*_*_*_TEST _  _  _  _  _
```

You can now type line feeds whenever you like. If you press f_5, J will be displayed. However, if the character is within a print or write statement, it will cause a line feed for the thermal printer or an index on the typewriter.

TYPE: PRINT "ABCJDEF" (using the f_5 key for J)
PRESS: EXECUTE

ABC
DEF will be printed on the thermal printer.

ABC
   DEF will be printed on the typewriter.

Pages F-6 and F-7 of the 9830A Operating and Programming Manual give all the ASCII codes and their corresponding outputs. The keyboard characters which can be stored are those with the following ASCII codes: 0 through 10, 12, 14 through 31, 91 through 96, and 123 through 127.

The CHAR program above permits you to enter up to ten of these characters into the keys. For less than ten characters, terminate by entering 999.

The characters are stored in the following sequence: f_5, f_0, f_1, f_2, f_3, f_4, f_6, f_7, f_8, f_9.

Null String Entry

by Dennis Eagle, Hewlett-Packard, Desktop Computer Division

If you have an HP 9830A with the strings ROM, there are times when in the Program mode, you would like to input a null or empty string. For example,

```
10 DIM A$[80],E$[80]
20 DISP "EMPLOYEE'S NAME";
30 INPUT E$
35 PRINT E$
40 IF LEN(E$)=0 THEN 70
50 A$=E$
60 GOTO 20
70 DISP "DONE"
80 END
```

To input a null string, enter a quotation mark (") and press EXECUTE.

The BASIC compiler uses quotation marks as indicators for the beginning and ending of strings, for example, A$="ABC". If there are no characters between the quotation marks (A$=" "), then the string is empty and its length is zero.

Use of Dummy Variables to Control Programs from the 9864A

by Dr. P.A. Burrough, School of Geography, University of New South Wales, Kensington, N.S.W., Australia

A common use of the digitizer is to measure lengths of curved lines on maps or charts, etc. Usually this is programmed in the form of a simple loop, with the digitizer in continuous mode:

```
10 D=0
20 ENTER (9,*)X,Y
30 ENTER (9,*)X1,Y1
40 D=D+SQR((X-X1)↑2+(Y-Y1)↑2)
50 X=X1
60 Y=Y1
70 GOTO 30
80 PRINT "DISTANCE="D
90 END
```

There are disadvantages to this method. To print out the result at the completion of the line, the continuous mode is switched off, the program must be stopped and recommenced by CONT 80, EXECUTE to print the result. This is time consuming and unnecessary, because by simple programming, the digitizer can be made to control the calculator. This is a great advantage if much digitizing is to be done.

Consider the following program:

```
10 D=0
20 ENTER (9,*)X,Y
30 ENTER (9,*)X1,Y1
40 IF X1<0 AND Y1<0 THEN 90
50 D=D+SQR((X-X1)↑+(Y-Y1)↑2)
60 X=X1
70 Y=Y1
80 GOTO 30
90 PRINT "DISTANCE ="D
100 GOTO 10
110 END
```

The line is digitized in continuous mode. At the end the continuous mode is switched off and a single double negative X, Y value is entered. This causes the line length, stored in D, to be printed via statements 40 and 90. After this the program returns to the beginning to measure the next line, all without need for control via the 9830 keyboard.

By varying the nature of the IF-THEN conditions, a whole range of controls over program operation may be obtained using only data from the digitizer, thus providing simple and flexible operation.

String Variables

by T.P. van der Zee, Eindhoven, The Netherlands

In many cases it is necessary to take the value of a string. If the string is non-numeric, Error 76 will occur. A technique has been developed to avoid this.

The first position in the string must always be declared as 0. The input must be given directly behind this position. Then by taking the value of the total string, no error will occur.

Example

```
230 DIM B$[10]
240 B$[1,1]="0"
250 INPUT B$[2,10]
260 A=VAL(B$)
270 END
```

If B$ (2,10) = "125" then A = 125.
If B$ (2,10) = "ABC" then A = 0.

If it is necessary to repeat the request for input in the second case (non-numeric argument), the next sequence applies:

```
370 DIM D$[10]
380 D$[1,1]="0"
390 INPUT D$[2,10]
400 IF D$[2,2]="0" THEN 440
410 A=VAL(D$)
420 IF A=0 THEN 390
430 GOTO 450
440 A=VAL(D$)
450 END
```

A RUN or INIT command erases the contents of the strings. To check whether or not one of these commands has been given, the following special test with the aid of T$ is suggested.

```
540 DIM H$[10],T$[2]
550 T$[1,1]="0"
560 IF VAL(T$)=1 THEN 600
570 DISP "YOUR NAME";
580 INPUT H$[1,10]
590 T$[2,2]="1"
600 PRINT H$[1,10]
610 END
```

As soon as H$ has been input, T$ (2,2) must be declared 1. Because a RUN or INIT command will also erase this information, the name is asked again after a RUN command. After STOP END CONT EXECUTE, the name will be printed immediately. With the aid of this string, it is also possible to check whether or not variables have been erased by a RUN or INIT command.

Available Memory

by Bob McCoy, Hewlett-Packard, Atlanta, Georgia

When the Model 30 memory has information in it and you need to know how many words of memory are still available, the Model 30 Operating and Programming Manual gives a key sequence LIST 9 9 9 9 EXECUTE to display this. A shorter and faster routine to get the same result is LIST followed by pressing any Special Function key.

More on Available Memory

by Professor Danial G. Maeder, Versoix, Geneva, Switzerland

Pressing LIST, any available Special Function key instead of 9999 — is worth much more than the little note lets one think. In fact, LIST, Special Function key, leaves the main program counter unchanged, whereas, after the conventional LIST 9999 one has to FETCH again the program line on which one had worked last. For someone who made it a habit to check the available memory after every program line change, the possibility of continuing the editing on the neighboring program lines without FETCH is very desirable. It also helps to save paper if one edits a program in the PRT ALL mode, by avoiding useless LIST and FETCH printing.

A "Keyboard Interrupt"

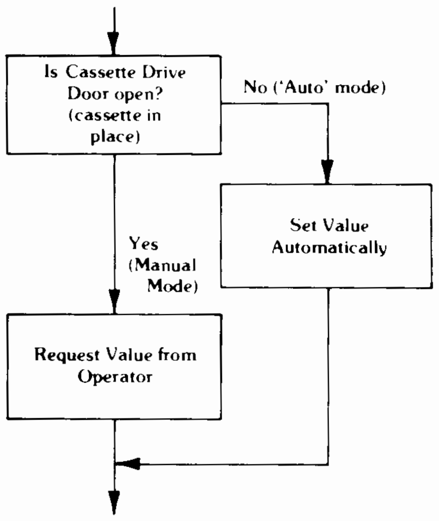
by Hewlett-Packard, Melbourne, Australia

Wouldn't it be nice to hit a key on the 9830A and cause a flag to be set? That is, to be able to change the course of a program while it is running from the keyboard? It can be done on the 9820A, 9821A, and 9825A, but there is no key to achieve this on the 9830A. Or is there?

If you have an 11272A Extended I/O ROM, try the following:

- Type in the program (see below).
- Put any cassette into the transport and close the door.
- Run the program. "999" will flash repeatedly.
- Open the cassette door.
- Enter a value manually in response to the display.
- Until the door is closed again, the program remains in manual mode.

```
10 V=999
20 IF STAT 10=8 THEN 50
30 DISP "ENTER VALUE";
40 INPUT V
50 DISP V
60 WAIT 1000
70 GOTO 10
80 END
```


Is Cassette Drive Door open? (cassette in place)
No ('Auto' mode)
Set Value Automatically
Yes (Manual Mode)
Request Value from Operator

Interrupt System

by David A. Ripley, General Dynamics, Albuquerque, New Mexico

If you have written programs containing nested or lengthy "DO loops" you probably know that there is no interrupt system for the 9830A as such. For example, you cannot alter your program flow any way short of stopping the program, changing a statement, and continuing from there. This forces the programmer to do one of two things: either display each result or wait for termination, assuming the loop is not "hung up."

The following sample may be useful to you as it allows a physical action on your part to cause branching. It uses the STAT (status) command found on pp. 3 - 4 and A-3 of the Extended I/O ROM manual to allow for such interrupts, i.e., by opening or closing the tape transport door. Only one command is needed to perform this function. See statement 120 of the example. Note that this statement assumes there will be a tape inserted into the transport and ready (not on clear leader). If the tape is on clear leader or if the transport is empty, a different value will be returned with "STAT."

Execution time can be drastically reduced for multiple calculations by this method as compared to displaying or printing each result.

Comparative Times

DISP Statement	STAT Statement
100 items approx. 20 sec.	100 items approx. 2 sec.

The STAT statement can be used at any time to allow branching by the simple IF statement. There are many other applications for this statement, such as printing totals, etc., without terminating execution.

Example

```
10 REM* SET UP FOR LOOP CALCULATIONS *
20 FOR I=I TO 1000
30 REM
40 REM** CALCULATION SECTION **
50 Y=INT(RND(I)*1000)
60 REM
70 REM** CHECK STATUS OF TAPE TRANSPORT **
80 REM** IF STATUS = 11 DOOR IS OPEN **
90 REM***** PRINT CALCULATIONS IF STATUS NOT
   EQUAL TO 11 (DOOR SHUT) **
100 REM*** STAT CHECK ASSUMING TAPE IN TRANSPORT
    AND READY **
110 REM
120 IF STAT10=11 THEN 140
130 PRINT "Y ";Y;" I" ;I
140 NEXT I
150 END
```

A Method of Inputting Variables

by A. de Faro Barros, GESPO, Porto, Portugal

Sometimes one needs to enter an array, many of whose elements are zeros. Instead of the time-consuming

```
1010 FOR J = 1 TO 50
1020 FOR K = 1 TO 9
1030 DISP "NUMBER";
1040 INPUT M(K,J)
1050 IF M(K,J) = 0 THEN 1070
1060 NEXT K
1070 NEXT J
```

• • •
use

```
1010 FOR J = 1 TO 50
1020 DISP "DEPENDENCIES OF" J;
1030 INPUT N(1), N(2), N(3), N(4), N(5), N(6)
     N(7), N(8), N(9)
1040 FOR K = 1 TO 9
1050 IF N(K) = 0 THEN 1080
1060 M(K,J) = N(K)
1070 NEXT K
1080 NEXT J
```

• • •

Continual digitation of 9 numbers can be partially avoided by entering

,0,0,0,0,0,0,0,0

onto a Special Function Key (f_9, for example). As an illustration,

- The display shows: "Dependencies of 5?"
- You enter: 3, 4, 15, 33, 45
- Press: f_9

Row 5 of the array now contains

3, 4, 15, 33, 45, 0, 0, 0, 0.

WAIT Within a DISPLAY

by Andrew Vettel, Jr., Steel Valley School District, Homestead, Pennsylvania

If a program contains a series of DISP statements followed by WAIT statements, it is possible to place the WAIT within the DISP as follows:

```
10 DISP "MESSAGE NO. 1" FNW2
20 DISP "MESSAGE NO. 2" FNW2
30 DISP "MESSAGE NO. 3" FNW4
40 . . .
999 END
1000 DEF FNW(X)
1010 DISP TAB32,
1020 WAIT 1000*X
1030 RETURN 0
```

The multiple line function, FNW(X), is constructed to take an argument that specifies the WAIT in seconds.

Monetary Formatting

by Bob McCoy, Hewlett-Packard, Atlanta, Georgia

When the output of your computation on the HP 9830A is in monetary units such as dollars, it is convenient to have the dollar sign preceding the figure, as well as having the digits grouped in threes separated by commas, especially when six or more digits appear to the left of the decimal. The routine shown below will insert the dollar sign and commas as required, according to the number of digits in the output. The input must be a minimum of .XX, and the routine requires the Extended I/O ROM (or appropriate DEXP command on the Mass Memory) and the String Variables ROM.

Example

```
$ 0.50
$-0.50
$ 0.02
$-0.02
$ 123.00
$-123.00
$ 123,456.00
$-123,456.00
$ 123,456,789.00
$-123,456,789.00
```

```
10 DIM A$[20],B$[20]
20 FIXED 2
30 INPUT A
40 OUTPUT (A$,*)A
50 FOR I=1 TO 20
60 B$[I,I]=""
70 NEXT I
80 X=LEN(A$)-2
90 A=17
100 B$[18,20]=A$[X-2,X]
110 X=X-3
120 IF X>4 THEN 150
130 B$[A-X+1,A]=A$[1,A-(A-X)]
140 GOTO 220
150 B$[A-2,A]=A$[X-2,X]
160 X=X-3
170 A=A-3
180 IF X <= 1 THEN 220
190 B$[A,A]=","
200 A=A-1
210 GOTO 120
220 A$[1]="$"
230 A$[2]=B$[A-X+1]
240 IF POS(A$,"-")=0 THEN 260
250 A$[1,2]="$-"
260 PRINT A$[1,21-(A-X)]
270 GOTO 30
280 END
```

Quick XREF

by Joe Armstrong, Hewlett-Packard, Desktop Computer Division

When debugging programs, it is often necessary to find the location of one or more variables (or even to see if a variable exists) within a given program. The usual procedure is to execute the XREF command found in the Advanced Programming I ROM. A printed cross reference of all the variables withing a program is printed. A significant amount of time and printer paper can be used during the normal debugging of a program using this standard XREF procedure. To cross reference only the variables you are interested in, simply define these variables in the first few lines of the program. Suppose you are interested in the following variables; A, B, C, D, A(10), A$. Simply key the following lines into your program:

```
01 A=B=C=D=0
02 A(10)=0
03 A$=" "
```

It is assumed that your mainline program starts at a line number greater than 03, and there is no common statement. Now execute the XREF command. The cross reference will print out the locations of the variables in the order shown above. After the cross reference is complete, press the STOP key to terminate the XREF command. NOTE: Be sure to delete the lines entered before saving your program.

ARCTG in the 0° to 360° Range

by Ing. Stanislav Milacek, State Res. Inst. for Machine Design, Bechovice, Czechoslovakia

The phase of a complex number from X and Y components in any range (e.g., 0 to ±180 or 0 to 360 degrees) can be calculated easily by the algorithm described in the sample program shown below. Note that the 'security' coefficient k = 1E-98 in Line 70 fits the Y/k value, which must be smaller than the numeric range of the calculator.

```
10 DEG
20 INPUT X,Y
30 PRINT X;Y;FNA1;FNA2;FNA3
40 GOTO 20
50 END
60 DEF FNA(I)
70 A=ATN(Y/(X+1E-98* NOT X))
80 B=180*(X<0)*(SGNY+ NOT Y)
90 GOTO I OF 100,110,120
100 RETURN A
110 RETURN A+B
120 RETURN A+B+360*((A+B)<0)
```

Example

```
  1        0        0        0        0
  1        1       45       45       45
  0        1       90       90       90
 -1        1      -45      135      135
 -1        0        0      180      180
 -1       -1       45     -135      225
  0       -1      -90      -90      270
  1       -1      -45      -45      315
```

Storing Alpha on a Data Tape

by John E. Barber, Cook Coggin Engineers, Inc., Tupelo, Mississippi

This routine is used to store alpha on a data tape without using an AP ROM. There are many ways to use this routine, but the example shown below uses an external cassette and stores the alpha in the first row of the array. With this method, your data tape can be marked in equal size files so it can be used to store more than one set of data. If all storage is alpha, the precision should be changed to save storage space.

Example

```
10 DIM T[5,40],A$[80],B$[40],C$[4]
20 A$="   #$%&'()*+,-./0123456789:;<=>?ABCDEF
   GHIJKLMNOPQRSTUVWXYZ     "
30 DISP "DATA FILE #"
40 INPUT D
50 LOAD DATA #5,D,T
60 DISP "NAME";
70 INPUT B$
80 B=LEN(B$)
90 FOR I=1 TO B
100 C$=B$[I,1]
110 C=POS(A$,C$)
120 T[1,I]=C+31
130 NEXT I
140 FORMAT B
150 FOR I=1 TO 40
160 WRITE(15,140)T[1,I]
170 IF NOT T[1,I] THEN 190
180 NEXT I
190 PRINT
200 STORE DATA #5,D,T
210 GOTO 30
220 END
```

Synchronization Between Timeshare and the 9830A

by Finn Hendil, Philips Elektronik Industri Ak/S, Copenhagen S, Denmark

When the 9830A is used as a terminal for a remote timesharing system having more than one fixed transmission speed, the speed is sometimes indicated from the 9830A by transmission of one specific character repeated several times.

In the TERM mode of the 9830A, the TRANSMIT function is terminated with a Carriage Return, which disturbs the proper synchronization to the timesharing system, and as there should be a time interval between the transmitted characters, we have found that this little program on one of the unused Special Function Keys gives a perfect synchronization each time:

```
10 FOR N = 1 TO 8
20 WRITE (4,*)"H";
30 WAIT 400
40 NEXT N
50 END
```

After the 9830A is in the TERM mode, the procedure is to press the key when the characters are to be transmitted and proceed in the usual way.

Right Justifying Input Strings

by Jordan Siedband, Harper College, Palatine, Illinois

The program given below will right justify input strings if the output device for the 9830A is a 9871A Printer. Any line width (M) could be specified. The one shown is 76 normal characters in length, or 7.6 inches of text. If the number of characters is less than 56 or (M-20), the machine does not right justify. This permits ends of paragraphs or tabulated data to print in their normal fashion. For finer line adjustments, the 20 could be replaced by 12, for example, Lines 500 - 600 are intended as the printing sub and could be used in any application when S0, M, and A$ are known.

```
10 DIM A$[80]
20 M=76
25 S0=15
30 DISP "TEXT";
40 INPUT A$
50 IF LEN(A$)>76 THEN 30
500 REM 9871 LINE JUSTIFICATION ROUTINE; M=MAX
    LINE LENGTH (CHARS)
510 REM S0=SELECT CODE OF OUTPUT PRINTER
520 L=LEN(A$)
530 IF L<M-20 THEN 590
540 A=INT(12*M/L)
550 X=12*M-A*L
560 FORMAT 20B
570 WRITE (S0,560)27,72,0,A,A$[1,L-X],27,72,0,A+1,
    A$[L-X+1],27,72,0,12
580 GOTO 600
590 WRITE (S0,560)A$
600 REM LINE PRINTED & JUSTIFIED IF LENGTH NOT
    <M-20
610 GOTO 30
620 END
```

String Comparisons

by Francois Martin, Tudor Engineering Company, Seattle, Washington

When a "yes" or "no" reply is input in answer to a program query and the reply is tested to determine program branching, as in the linc:

130 IF B$(1,1) = "Y" THEN 900

the user normally types in Y or YES without pressing the SHIFT key.

This is the expected reply, resulting in a "true" decision in comparing the Y's and proper branching to line 900, provided the Y in quotation marks above was also programmed in the unshifted mode. However, if the programmer held the shift key down while typing the Y in line 130, the test then compares Y (octal code 131) with y (octal code 171), so branching will not occur. The same difficulty occurs if the programmed Y was entered in the unshifted mode and the user inadvertently enters his reply in the shifted mode.

In all 9830A programs published by HP, alpha string characters are entered in the unshifted mode. An alpha YES reply in the unshifted mode then causes the expected program action. Users should remember that although the display and the printed 9866A output look the same for either shifted or unshifted letters, the calculator sees and compares different octal codes.

Using the Special Function Keys to Represent Data

by R.J. Carter, CSIRO, Clayton, Victoria, Australia

The method of using the 9830A Calculator's Special Function Keys to store sets of data and then to input those sets into a program as required is effective where:

- Some of several sets of data are required for each run;
- Sets of data are to be entered several times in each run;
- The same sets of data are entered in a different order for each run; or
- Sets of data are to be entered manually.

The method increases in value if combinations of the above are needed. In my case, the combination of all four applications occurred, and a saving of just over 1000 words in a 5100-word program was produced by the use of the method.

Data Entry to Special Function Keys

To put data on a Special Function Key, first enter the KEY mode by pressing the FETCH key then the desired fx key. If no information is on the key, KEY appears on the display. The data can be entered if an asterisk (*) is keyed in first. Key in the data, separating each number by a comma; then the key number, separating this from the data numbers by a comma. Complete the entry by pressing the END OF LINE key, which automatically exits the KEY mode. Pressing the FETCH and fx keys would now produce a display such as:

*0.7796, −3.584, 5.6883, −2,862,
−2.3719, 12.1878, −4.0823, 1.3468, 8*

The key number (or an identifying string) is included so that it may be later output as a check for the correctness of input data.

Twenty such sets of data may be entered onto the Special Function Keys and conveniently stored on the first file of a program cassette by using the command STOREKEY 0. Before each use of the program cassette, the data are restored to the Special Functions Keys by the command LOADKEY 0.

The method may be used even when there is already information on the key, but all of the previously recorded information is lost.

Data Entry to Mainline Program

The mainline program uses the following method of entering data:

```
510 FOR Z = 1 TO L
520 DISP "DATA & KEY: SUBSCRIPT"Z;
530 INPUT A[Z], B[Z], C[Z],...K[Z]
540 NEXT Z
```

Pressing the desired Special Function Key inputs the required data and automatically restarts the program. Omitting the second asterisk at entry of data to the keys allows data to be viewed at run time. In this case, pressing the Special Function Key inputs the required data and stops the program. The program is restarted by pressing the EXECUTE key.

Data may be entered manually if they have not been previously stored.

Single-Line Cross Reference

by Dennis Eagle, Hewlett-Packard, Desktop Computer Division

There are times when it is very useful to know where in a program a given line is referenced. In the following example, if you change Line 14 to 16, the program cannot be run because Line 14 is referenced in Lines 10, 12 and 30.

```
1  X=1
10 GOTO 14
12 GOTO X OF 14,20
13 REM
14 FORMAT 6F5.0
20 END
30 WRITE (15,14)
```

In a program of 1000 steps or more, it becomes very difficult to see all the lines referencing a given line. The following procedure permits a user to obtain a cross reference for a given line.

1. Change the line number of the line in question.
2. Type: 9999 GOTO 9998
   Press: END OF LINE
3. Be certain that there is not a Line 9998.
4. Type: REN
   Pres: EXECUTE
5. ERROR 44 IN LINE XXXX will be displayed, where XXXX is a line number in your program. The line in question in step 1, is referenced in Line XXXX.
6. Change Line XXXX so that it now references the new line number established by 1.
7. Go back to 4. and continue to perform 4. through 6. until ERROR 44 IN LINE 9999 is displayed
8. Type: DEL 9999
   Press: EXECUTE

As an exercise, key in the example program.

1. Change Line 14 to Line 16.

```
1  X=1
10 GOTO 14
12 GOTO X OF 14,20
13 REM
16 FORMAT 6F5.0
20 END
30 WRITE (15,14)
```

2. ENTER Line 9999

```
1  X=1
10 GOTO 14
12 GOTO X OF 14,20
13 REM
16 FORMAT 6F5.0
20 END
30 WRITE (15,14)
9999 GOTO 9998
```

3. If there is no Line 9998,
4. Type: REN
   Press: EXECUTE
5. ERROR 44 IN LINE 10 will be displayed
   Type: 10 GOTO 16
   Press: END OF LINE

```
1  X=1
10 GOTO 16
12 GOTO X OF 14,20
13 REM
16 FORMAT 6F5.0
20 END
30 WRITE (15,14)
9999 GOTO 9998
```

Repeat 4 and ERROR 44 IN LINE 12 will be displayed.
Type: 12 GOTO X of 16,20
Press: END OF LINE

The program will now be listed as:

```
1  X=1
10 GOTO 16
12 GOTO X OF 16,20
16 FORMAT 6F5.0
20 END
30 WRITE (15,14)
9999 GOTO 9998
```

When 4 is repeated again, ERROR 44 IN LINE 30 will be displayed. Change the reference in Line 30 from 14 to 16. Now when you attempt to renumber, ERROR 44 IN LINE 9999 will be displayed. Line 9999 purposely references a nonexistent line so that the program won't actually be renumbered. Delete Line 9999. All references to Line 16 have been found and changed.

Calculations During Program Execution

by Joe Armstrong, Hewlett-Packard, Desktop Computer Division

During the execution of a program, it is often necessary to make calculations to answer certain program-prompted questions. You can use the HP 9830A to perform these calculations by pressing either the up or down display keys (↑,↓) when the 9830 is waiting for an input. This will put the 9830 in the calculator mode. You can now perform any normal keyboard function.

Examples

1. To check the status of A: key in A and press EXECUTE.
2. To calculate A*B/C: key in A*B/C and press EXECUTE.
3. To change the status of D(1,5). Key in D(1,5) = T*5 and press EXECUTE.

You can return to the input step where you exited the program by pressing CONT and EXECUTE. The only disadvantage is that the original display message will be replaced by a ?. Simply remember what the message was and enter the response as usual.

NOTE: Use the 9830 as a calculator only. Do not attempt to edit, add, or delete any program lines during this procedure. To do so would cause the 9830 to lose its pointer to the step in memory where you exited the program.

Increasing Storage Capacity

by Philippe Kent, Lausanne, Switzerland

A substantial amount of memory may be wasted when storing large numbers of experimental results in full precision arrays. The values are often of (rather low) constant relative precision and/or of small dynamic range. The use of split precision arrays will double the storage capacity, but use of the following procedure will double that capacity again. The gain in access time, if the values are stored on the tape cassette, is also considerable.

The trick is to use a signed, biased logarithm of suitable base. The base and bias are chosen such that the log of the maximum absolute value encountered is 32767 and the log of the minimum absolute value is 1:

$\log_a(\max|V|) + b = 32767,$

$\log_a(\min|V|) + b - 1,$ or

$a = (\max|V|/\min|V|)^{1/32766},$

$b = 1 - \log_a(\min|V|)$ with

$\log_a x = \log_e x/\log_e a$

The absolute experimental value is then converted into its biased log signed as the original value and stored in an integer precision array.

Encoding is accomplished by:

```
10 DEF FNC (X)
20 N = 0
30 IF X = 0 THEN 50
40 N = b + c*LOG (ABSX)
50 N = SGNX*INT (N*(N>0) + 0.5
60 RETURN N
70 END
```

where b is the bias as above and c is $1/\log_e a$.

Decoding is accomplished by:

```
80 DEF FND(N)
90 X - SGNN*EXP (d*(ABSN - b))
100 RETURN X
110 END
```

where d is $\log_e a$.

For b = 16000, c = 2000 and d = 5E − 4, for example, numbers between ±0.0003357 and ±4374 as well as±0 can be represented, with an accuracy greater than 3 parts in 10,000, by an integer. The smaller the dynamic range, the greater the accuracy.

Properties of the coded value are such that zero will always convert to zero, an underflow results in zero, an overflow in recoverable Error 105 on attribution to the integer variable, and the relational expressions (<, =, >) remain valid between variables coded with the same base and bias. The relational expressions always remain valid when one side is zero. Direct multiplication and division may also be performed after separation of the sign, but checks on sign, overflow and underflow will nullify the gain in execution time compared to decoding-multiplication division-encoding.

Speeding Cassette Tape Access Time

by Thomas Krantz, Bermuda Division of Palisades Geophysical Institute, St. Davids, Bermuda

A decrease in cassette tape access time can be obtained by using a different file marking routine than the method described in the manual.

To illustrate the difference, two tapes were marked with ten usable files of 3000-word length. Figure 1 illustrates the marking method described in the calculator manual, which I will refer to as the "normal" method. The different marking routine, referred to as the "modified" method, is the same as the normal method, except that a minimal size file (4 words) is inserted before each usable file (see Figure 2).

The seventh usable file, File 7 of the normal tape and File 15 of the modified tape, was selected as the reference file. Time measurements with a stop watch were made between the execution of the LOAD 7, 10, 10 command and the appearance of the first line of a dummy program in the display.

Prior to the LOAD 7, 10, 10 (15 for modified tape), the tape was positioned using the LOAD, FIND and REWIND instructions. Two sets of FIND commands were used, since the normal tape was sensitive to the position of the tape prior to the FIND command — rewinding the tape before executing the FIND command, and positioning the tape to the end, File 10 for normal and File 20 for modified, before executing the FIND command.

In all cases except one, access time of the modified tape was faster than the normal tape. For the 31 measurements made, the mean and standard deviation are:

normal tape	M = 83.0 sec	sd = 30.9 sec
modified tape	M = 61.2 sec	sd = 24.7 sec

The modified method does have its drawbacks:

1. It takes about 6 minutes longer to mark the tape.
2. File positions are not in direct order, but this can be adjusted for by using a conversion statement where modified file number = (2 x normal file number) + 1.

Our conclusion is that the modified method of tape marking wins hands down. The methods used to test it are by no means complete, but are good enough to warrant using the modified method for a while to see how it works for you.

```
MARK 10,3000

TLIST

  0    0    3000  0       0      0      0
  1    0    3000  0       0      0      0
  2    0    3000  0       0      0      0
  3    0    3000  0       0      0      0
  4    0    3000  0       0      0      0
  5    0    3000  0       0      0      0
  6    0    3000  0       0      0      0
  7    0    3000  0       0      0      0
  8    0    3000  0       0      0      0
  9    0    3000  0       0      0      0
 10    0    3000  0       0      0      0
```

Figure 1

```
10 FOR I=1 TO 10
20 MARK 1,4
30 MARK 1,3000
40 NEXT I
50 MARK 1,4
60 END
```

Figure 2a

```
TLIST

  0      0    4    0         0      0      0
  1      0  3000   0         0      0      0
  2      0    4    0         0      0      0
  3      0  3000   0         0      0      0
  4      0    4    0         0      0      0
  5      0  3000   0         0      0      0
  6      0    4    0         0      0      0
  7      0  3000   0         0      0      0
  8      0    4    0         0      0      0
  9      0  3000   0         0      0      0
 10      0    4    0         0      0      0
 11      0  3000   0         0      0      0
 12      0    4    0         0      0      0
 13      0  3000   0         0      0      0
 14      0    4    0         0      0      0
 15      0  3000   0         0      0      0
 16      0    4    0         0      0      0
 17      0  3000   0         0      0      0
 18      0    4    0         0      0      0
 19      0  3000   0         0      0      0
 20      0    4    0         0      0      0
 21      0    4    0         0      0      0
```

Figure 2b

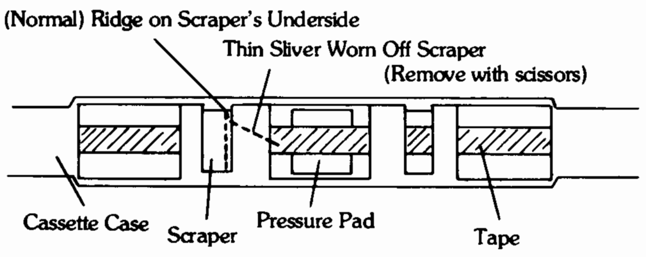
Recoverable Error 59s

by Ian Bird, I.T. Bird and Associates, Watson, Australian Capital Territory, Australia

Over 90% of the Error 59s I have experienced are recoverable without loss of data or program. Other users have verified this statement.

Recoverable errors appear to be caused by the tape wearing a very thin sliver of plastic off the tape scraper insert. This hair-like plastic fibre touches the read head and causes errors.

Cut the fibre with a pair of scissors and all is well. A magnifying glass could well be of assistance.


(Normal) Ridge on Scraper's Underside
Thin Sliver Worn Off Scraper
(Remove with scissors)
Cassette Case
Scraper
Pressure Pad
Tape

*A Magnifying Glass is Required to See the Fibre

Speeding Execution Time

by John Bidwell, Hewlett-Packard, Desktop Computer Division

The HP 9830 spends a significant amount of its execution time searching for variables in the symbol table. The deeper a symbol is in this table, the longer the search time. To achieve a saving in execution time of large programs with many variables, highly used variables can be put at the best location in the symbol table. The rules for where variables are in the symbol table are as follows:

Searched first (best execution time):	1. Last simple variable encountered during EXECUTION.
	2. Previous simple variables
	3. Common Statement (first variable searched first).
	4. 1st DIM Statement (first variable searched first).
Searched last:	5. Other DIMs (smaller line #'s are better).

Example

10 COM A,B,C
20 DIM D,E,F
30 Z=1
40 DIM G,H,I
50 X=1
.
.
.
1000 END

Symbol Table

Searched first:	X	simple variables
	Z	
	A	COM
	B	
	C	
	D	1st DIM
	E	
	F	
	G	last DIM
	H	
Searched last:	I	

In the example, if no new variables were encountered, 'X' would remain at the top of the symbol table and should be highly used (as FOR LOOP, counter variable, etc.). 'Z' should be the next most highly used variable, and so on. Therefore, by finding the most highly used variables in an existing program and initializing them at the appropriate point in the program, a significant saving in execution time can be achieved.

Signaling the End of a Program

by David M. Kuchta and Rona J. Newmark, Case Western Reserve University, Cleveland, Ohio

In the course of our programming on a Hewlett-Packard 9830A, we have developed the following method of signaling the end of a program, or a particular segment of program, such as a lengthy routine followed by an input by the operator. By inserting

10 A=9↑9↑9↑9↑9↑9↑9↑9↑9↑9↑9↑9↑9↑9↑9

at the end of a segment, the calculator will emit its characteristic beeping sound for ten seconds. Since this results in a recoverable error (error message 100 — numeric overflow), program execution can be continued at the next line. Each time the expression is raised to another power of 9, the machine will beep for another second.

Aligning Printed Headings

by Jack L. Gehrs, Tico Office Equipment and Supplies, River Forest, Illinois

When I wish to align a printed heading with a succeeding formatted output, I fill the WRITE line with numbers that equal each format statement first. I then go back and fill in the necessary heading.

Example

```
10 FORMAT F8,0
20 WRITE (15,10)"8765432187654321876543218765432187654321"
20 WRITE (15,10)" CODE #  CODE #  CODE #  CODE #  CODE # "
   (second writing)
30 FOR I=1 TO 5
40 WRITE (15,10)I;
50 NEXT I
```

I have found that filling the words first and then going back and removing the numbers remaining with the space bar to provide spaces is the easiest procedure.

Transferring Program Files Between Two Mass Memory Units

by Ing. Giuseppe Barzagli, S.I.M., Bologna, Italy

This tip requires a 9830A, String Variables ROM, 9880B Mass Memory System and related Mass Memory ROM.

I've found the following program useful to transfer a program file between two mass memory units without changing the name of the program file and without renumbering the lines. This is very important in cases of transferring many files of the same program without using the platter duplicate procedure; the program could not otherwise find the right subsequent files of the right line numbers.

Alternately, it is dangerous and tedious to use the keyboard command sequence UNIT, GET, UNIT, SAVE.

The program operates by building a new program with the ordered sequence of UNIT, GET, SAVE, and the right program file names and numbers. This program is immediately executed and, when the transfer has been accomplished, the control is passed again to the main program, COPRG, which is always resident in Unit #1. In Unit #1 is needed the data file COPRG of 1 record, too, as an intermediate storage for the generated program.

Lists of the main program, COPRG, and of the transfer program, PRGNAM, between Unit #0 and #1 are given.

```
100 REM *** FILE COPRG ***
110 DIM C$[80],A$[6],N$[4],O$[1],P$[1],D$[1]
120 FILES COPROG
130 Z=34
140 DBYTE Z,D$
150 DISP "PROGRAM NAME";
160 INPUT A$
170 DISP "FIRST LINE NUMBER";
180 INPUT N$
190 DISP "FROM UNIT";
200 INPUT P$
210 DISP "TO UNIT";
220 INPUT O$
230 PRINT A$;"    F.L.N. = ";N$;"    FROM
      U. = ";P$;"    TO U. = ";O$
240 C$="1 UNIT "
250 C$[LEN(C$)+1]=P$
260 PRINT #1;C$
270 C$="2CHAIN"
280 C$[LEN(C$)+1]=D$
290 C$[LEN(C$)+1]=A$
300 C$[LEN(C$)+1]=D$
310 C$[LEN(C$)+1]=","
320 C$[LEN(C$)+1]=N$
330 PRINT #1;C$
340 C$="3 UNIT "
350 C$[LEN(C$)+1]=O$
360 PRINT #1;C$
370 C$="4 SAVE"
380 C$[LEN(C$)+1]=D$
390 C$[LEN(C$)+1]=A$
400 C$[LEN(C$)+1]=D$
410 C$[LEN(C$)+1]","
420 C$[LEN(C$)+1]=N$
430 PRINT #1;C$
440 C$="5 UNIT 1"
450 PRINT #1;C$
460 C$="6 GET"
470 C$[LEN(C$)+1]=D$
480 C$[LEN(C$)+1]="COPRG"
490 C$[LEN(C$)+1]=D$
500 C$[LEN(C$)+1]=",100,100"
510 PRINT #1;C$,END
520 DGET"COPROG"
580 END

RUN
PROGRAM NAME? PRGNAM
FIRST LINE NUMBER? 1520
FROM UNIT? 1
TO UNIT? 0
PRGNAM    F.L.N = 1520    FROM
    U. = 1    TO U. = 0

LIST

1 UNIT1
2 CHAIN "PRGNAM",1520
3 UNIT 0
4 SAVE "PRGNAM", 1520
5 UNIT 1
6 GET "COPPRG", 100,100
```

Avoiding VAL Function Errors

by William J. Zehner, Seascope Electronics, Inc., Lynn Haven, Florida

When writing programs that perform several distinct but related functions, it is sometimes useful to arrange for the operator to branch to various routines from a command/data input statement by using specifically designated alpha commands. Using this technique, the input variable must be a string name, and if, after looking through a set of defined alpha commands, the calculator finds no recognizable match, it should assume that numeric data is present. At that point we can utilize the VAL function to extract the numeric from the input string.

The difficulty with this procedure is that if the operator misspells or accidentally uses an undefined string, the tests for defined commands will be failed, an attempt will be made to take the VAL of a nonnumeric argument, and an Error 76 will result. The accompanying program illustrates one nice way to get around this problem. Beginning at line 110, the first character of the input string A$ is compared with each of the digits 0 through 9 contained in the check string C$. If any of the 10 digits is found in A$(1,1), the program branches to exercise the VAL function. Otherwise, an Invalid Entry message is flashed, the input rejected, and the program returns to the input statement to give the operator another try. A String Variables ROM is necessary for this program.

Example

```
10 REM STRING/VALUE WILLIAM ZEHNER 1/31/75
20 REM TO INSURE THAT A STRING HAS A LEADING
30 REM NUMERIC VALUE BEFORE ATTEMPTING TO
35 REM EXECUTE THE "VAL" FUNCTION.
40 DIM C$[10],A$[80]
50 C$="0123456789"
60 DISP "ENTER COMMAND OR DATA";
70 INPUT A$
80 IF A$="STOP" THEN 3000
90 IF A$="ANALYSIS" THEN 2000
100 IF A$="*" THEN 500
110 REM TO CHECK FOR NUMERIC DATA
120 FOR P=1 TO 10
130 IF C$[P,P]=A$[1,1] THEN 180
140 NEXT P
150 DISP "INVALID ENTRY. RETRY"
160 WAIT 3000
170 GOTO 60
180 V=VAL(A$)
190 DISP "OK"
200 WAIT 1000
210 GOTO 60
500 REM TO CALCULATE V.
510 V=20*LGT(PI↑2/9)
520 GOTO 190
2000 REM ANALYSIS WOULD BEGIN HERE.
3000 END
```

Inputting Varying Quantities of Numbers

by Robert Hardesty, DuPont, Rochester, New York

Inputting of several numbers from the HP 9830A keyboard is normally controlled by the number of variables specified in the INPUT statement. Sometimes it may be desirable to allow a varying quantity of numbers to be input; for example, plotting routines for a variable number of curves from different data tape files.

The following schemes allow inputting any quantity (up to 20 in these examples) of numbers. When requested, the numbers are typed in, separated by commas, as usual, and are input as a string. The VAL statement returns the numerical equivalent of the string up to the first nonnumeric digit, a comma. This value is either used immediately or placed in an array for later use. The 9830 then searches for the first comma and resets the string equal to everything beyond the comma. The program then returns to find the VAL of the new string. When a comma is no longer found, the loop is exited and the rest of the array can be filled with dummy zeros. N is then the number of variables input.

To use the numbers immediately:

```
10 DIM A$[80]
20 DISP "ENTER NUMBERS";
30 INPUT A$
40 FOR N=1 TO 20
50 X=VAL(A$)
60 REM AT THIS POINT DO THE DESIRED OPERATION
   ON THE NUMBER X
70 Z=POS(A$,",")
80 IF Z=0 THEN 110
90 A$-A$[Z+1]
100 NEXT N
110 END
```

To accumulate the numbers in an array for later use:

```
10 DIM A$[80],B[20]
20 DISP "ENTER NUMBERS";
30 INPUT A$
40 FOR N=1 TO 20
50 B[N]=VAL(A$)
60 Z=POS(A$,",")
70 IF Z=0 THEN 110
80 A$=A$[Z+1]
90 NEXT N
100 GOTO 140
110 FOR Q=N+1 TO 20
120 B[Q]=0
130 NEXT Q
140 END
```

by William Thompson III, Miner and Miner, Consulting Engineers, Inc., Greeley, Colorado

The HP 9830, despite its small size and cost, is capable of much more than most users realize at purchase time. Through experience it is soon found that with careful programming, major programs developed for larger machines may be adapted to the 9830. The program run time is normally slower on the 9830, but because of the ready access of the 9830, the user will frequently get results hours or days ahead of batch or time-share solutions. If the large programs are run infrequently, converting the program for use on the 9830 may eliminate entirely the necessity for batch or time-shared connections with costly minimum charges. With this in mind, I wish to offer the following tips and outline for such conversions, gained from the conversion of a Load Flow Circuit Analysis program from FORTRAN (run on an IBM 360 and a Xerox Sigma 3) to HP 9830 BASIC.

1. First, the dimension statements were closely examined to determine the number of words of 9830 memory necessary to fully dimension the arrays. In this conversion a 9830 with 5856 words of available memory was to be used, so the dimension statements were set up as in Figure 1, decreasing the size of circuit that could be analyzed, but not critically since past experience indicated that 95% of all cases run in our office used less than 20 busses and spare lines. Note that split precision in the 9830 is the same degree as full precision frequently found in other machines, and in a like manner double precision on other machines may represent the same degree of precision as 9830 full precision.
2. Next, the program flow was examined, and it was found that the program naturally broke into 5 segments. Iterative processes and loops that must cycle repeatedly were retained completely within a segment so that repeated loading of tape files, one of the slowest 9830 functions, was avoided. If disc files are available, such repeated program loads may be tolerable but still are avoided ideally.
3. Conversion on a line-to-line basis was begun. After some thought, the decision was made to replace FORTRAN variables by BASIC variables in sequence as they appear in the program. The alternative, attempting to imitate the FORTRAN variables by the retention of the same first letter or other means, results in a great confusing tangle. The only exception was to retain all index variables such as I, J, K, and M, used in "DO" loops, since they will be used repeatedly. A table starting with A0, A1, A2, etc., was set up and FORTRAN variables assigned as each line was translated.
4. In a like manner, a table listing equivalent line numbers eases the conversion of transfer of control statements (IF...THEN, GO TO, etc.).
5. FORTRAN "IF" statements are replaced by 1 or 2 BASIC "IF" statements. When replacing line numbers in "IF" and "GOTO" statements, use a standard number such as 1 or 9999 if the number refers to a statement that has yet to be translated, rather than attempt to calculate its future line number. After conversion, these are easily picked out and the line numbers changed to effect the proper transfer.
6. "FORMAT" statements that cannot be translated directly because of a required change in format, such as moving from a 132-character line to an 80-character line, are best changed by entering a statement like:

   950 FORMAT "FORTRAN STATEMENT #125", 8F8.0

   Then when you have the program running, it will print out the values and a reference to an appropriate format statement in your FORTRAN programs. With values for the Variables, you can experiment and change the format statement appropriately to achieve a readable output.

HP 9830 BASIC

```
2000 DIM AS[25],BS[25],CS[25,4],DS[25,4],
         ES[25],FS[25],GS[25],HS[25,7],II[25,2]
2010 DIM LS[25],MS[25],NS[25],OS[25],PS[25],
         RS[25],SS[25],TS[25]
2020 DIM US[25,9],VI[25,2],WS[100],XS[25],YS[25],
         ZS[25],JS[25],KS[25]
2030 DIM A$[25],B$[40],C$[240]
```

FORTRAN (IBM 360)

```
C 360 ELECTRIC POWER LOAD FLOW PROGRAM
C CAPACITY 100 LINES AND 100 BUSES
  DIMENSION P(100), Q(100), GS(100), BS(100),
    EMR(100), JSTR(100), EMB(100), ER(100),
    EI(100),PLOD(100), QLOD(100), QMAX(100),
    QMIN(100),PDEL(100), QDEL(100), ZLR(100),
    ZLI(100),ZKLR(100), ZKLI(100), OGEN(100)
  DIMENSION G(100), B(100), NFTO(100), RATO(100),
    RATG(100), B2(100), QQQ(100)
  DIMENSION NAM(5), ITIL(40), LNAM(100,5),
    NNAM(100,5), CLIN(10,7), LLIN(10,7),
    CBUS(10,9), LBUS(10,5), KBUS(200)
```

Figure 1

The Load Flow program was converted to BASIC in about two weeks in spite of a complete lack of knowledge of the actual power system formulas and units that the program used. A sample case was run on the 9830 to compare with results from a FORTRAN solution of the case. The 9830 results were in close agreement.

Do not expect exact agreement. The way the numbers and calculations are handled in different machines varies and may result in a buildup of accumulated "errors" — residuals may be a better word. The results should not be significantly different, however, so if they vary greatly, closer study is certainly necessary.

Converting the Load Flow program for use on the 9830 had an unforeseen outcome. Access to the program and turnaround time was so greatly improved that use of the program on the 9830 increased to a point where former users were finding it difficult to gain access to the 9830. Since our budget didn't allow purchase of another machine, another solution had to be found. The users of the Load Flow program indicated it would be acceptable to run the long cases, which were running up to 2 hours at a time, on an overnight basis. This would normally give results in less than a day or quicker, if priority seemed to demand it. These runs were not necessary every night and hiring another operator did not seem reasonable, so a sort of job-control program and operating system was created.

For this purpose the program listed in Figure 2 was written. Using 3 files (335 words) on a tape, it is able to run the program stored in files 1 through 5 with the data in up to 44 of the following files. The data for the original load flow was input as a card deck, and the converted version retained a semblance of this using "DATA" lines to imitate the cards. This seemed more desirable than keyboard entry during the program because of the great amount of data input necessary and the wish to keep it in a form where minor alterations could be made and the program rerun without duplicating the original key input. The only necessary addition to the data files was the last line (Figure 3) "MERGE 1,2000,2000". Changes to the load flow program itself were minimal. All exits such as "END" statements were changed to "LOAD 6, 10, 10", which returns control to the job-control program. If you have the "SERROR" statement available on your 9830, even data and program errors won't stop the sequence of the jobs to be processed.

File 0

```
10 DIM A$[45]
20 FOR I=1 TO 44
30 A[I]=999
40 NEXT I
50 A[45]=0
60 DISP "NUMBER OF DATA FILES";
70 INPUT N
80 IF N>44 THEN 60
90 FOR I=1 TO N
100 DISP "DATA SET";I;"TAPE FILE NO.";
110 INPUT A[I]
120 NEXT I
130 IF A[45]=0 THEN 150
140 LOAD DATA 7,A
150 A[45]=A[45]+1
160 IF A[A[45]]=999 THEN 190
170 STORE DATA 7,A
180 LOAD A[A[45]],10,10
190 DISP "ALL DATA COMPLETE";
200 END
```

File 6

```
10 DIM A$[45]
20 LOAD DATA 7,A
30 A[45]=A[45]+1
40 IF A[A[45]]=999 THEN 70
50 STORE DATA 7,A
60 LOAD A[A[45]],10,10
70 DISP "ALL DATA COMPLETE";
80 REWIND
90 END
```

Figure 2

File 7 — Data file in which array "A" is stored.

```
10 DATA "NRPPD RUSHVILLE"
20 DATA 1,0,1,1,400,100,115,0.01,1.5,1.5,5,75,"   "
30 DATA 2,1,0,1,2,1.193,2.72,0.1571,460,0,0,0,
    "RUSHVILLE SS-CLAY"
40 DATA 2,2,0,2,3,15,73.5,0,74,1,0,0,"WHITE CLAY
    CREEK SUB"
50 DATA 2,3,0,2,4,2.332,5.32,0.3071,460,0,0,0, "CLAY
    CREEK-LARRABEE"
60 DATA 2,4,0,4,5,15,73.5,0,74,1,0,0, "LARRABEE CREEK
    SUB"
70 DATA 3,1,0,0,0,100,50,0,0,1,0,0, "RUSH SS"
80 DATA 3,2,0,0,0,0,0,0,0,0,0,0, "CLAY 110"
90 DATA 3,3,0,0,0,0,0,0,0,10.831,5.253, "CLAY 25"
100 DATA 3,4,0,0,0,0,0,0,0,0,0,0, "LAR.110"
110 DATA 3,5,0,0,0,0,0,0,0,10.764,5.221, "LAR.25"
120 DATA 6,0,0,0,0,0,0,0,0,0,0,0, "  END  "
130 MERGE 1,2000,2000
```

Figure 3

This program has been used very successfully, expanding the use of the 9830 to around-the-clock on many days. In addition, this job control has been extended to other programs. Indeed, more than one type of program may be run at night by changing only the "MERGE" statement at the end of the data file to link with the appropriate program file. The number of programs or program runs is limited only by your tape or disc capacity or the means of entering external data into the machine. Even without the addition of that lovely HP Mass Memory, we still are planning on greater use of "batch" procesisng aided by a paper tape reader for data and the later addition of an external cassette for more program storage.

Salvaging a Program with ERROR 59 (9830A)

by Andrew Vettel, Jr., Steel Valley School District, Homestead, Pennsylvania, U.S.A.

When loading a program from a tape file and an ERROR 59 (check sum) occurs, more often than not it is impossible to list or display any lines at or beyond the line where the bit error occurred. Further, ERROR 1 occurs if an attempt is made to store those lines that were loaded without error. For example, STORE 2, 10, 240 will produce ERROR 1 even though there are no errors in lines 10 - 240.

The following procedure makes use of the RECALL buffer to transfer the undamaged lines one by one from mainline memory to a special function key, say f9:

STEP	KEY	COMMENTS
(1)	FETCH	Places lowest numbered line in display
(2)	EXECUTE	
(3)	BACK	Removes ⊦
(4)	END OF LINE	Store line in the RECALL buffer
(5)	FETCH	Access SFK where lines are being stored
(6)	f9	
(7)	RECALL	Retrieve line from RECALL buffer
(8)	END OF LINE	Store line on the SFK
(9)	END	Exit the SFK
(10)	$\boxed{\downarrow}$	Place next line in display Repeat procedure beginning with Step 3

Care must be taken not to attempt to place in the display any "damaged" lines. Once all salvagable lines have been placed on the Special Function Key, mainline memory should be SCRATCHed before the lines of the key are then stored in a nondefective tape file.

WRITE and FORMAT Incorporating Variable Length Strings

by G. Fletcher, Plessey Telecommunications Ltd., Beeston, Nottingham NG 9 1LA, England

It is often desirable to produce output with embedded variable lenth strings but maintaining tabulation of output columns. By extending each string to the maximum length as dimensioned, the subsequent variables remain tabulated.

Lines 30 - 50 in the example below will maintain string length and hence tabulation of results.

```
10 DIM A$[32]
20 READ A$,A,B,C
30 FOR I=LEN(A$)+1 TO 32
40 A$[I,I]=" "
50 NEXT I
60 WRITE (15,80)A,A$,B,C
70 STOP
80 FORMAT F3.0,3X,2F12.0
90 DATA "BRISTOL-TAUNTON H & G",1,400,12
100 END
```

WRITE and FORMAT Incorporating Variable Length Strings

By means of Keyboard 1978/2 we discovered that a sure way to receive world-wide comments from our customers is to publish a programming tip using one of several possible techniques to accomplish a particular purpose. In response to the programming tip by G. Fletcher, we received letters from ten readers from Australia, South Africa, Italy, the Netherlands, Spain, the United Kingdom and the U.S.A., suggesting alternative means to improve the method of maintaining string length and tabulation of output columns. We appreciate these letters, and have determined that the following coding will accomplish the task effectively. This was suggested by W. Guttormsen, McWilliam & Partners Pty. Ltd., Brisbane, Australia, and Josep M. Masso, La Vanguardia, Barcelona, Spain.

The coding in line 20 is changed as follows, and lines 30, 40 and 50 can be eliminated.

20 READ A1[1,32],A,B,C

FORMATting the Display

by Andrew Vettel, Jr., Steel Valley School District, Homestead, Pennsylvania, U.S.A.

The formatting capability available with the WRITE statement cannot be used in the lighted display of the 9830A since no select code is provided for the display. However if both the Strings and Extended I/O ROMs are installed, a formatted display may be accomplished as follows:

```
10 DIM A$(40)
20 X=5.4
30 OUTPUT (A$,40)X,X↑2,X,SQRX
40 FORMAT F4.1, "↑2=",F6.2,"&SQR(",F4.1,")=",
   F6.3
50 DISP A$
60 END
```

Transferring Values

by Daniel Treep, Folkert Postlaan 15, Abcoude, The Netherlands

9830A users who want to transfer values from one vector or matrix to another may use the matrix assignment statement if they have the Matrix ROM installed in their calculator. MAT X = Y is the usual procedure. However, if the dimensions of the arrays are not compatible or if there is no Matrix ROM in the calculator, the transfer can be accomplished this way:

```
10 DIM XI[10],YI[5,10]

100 R=4
110 FOR C=1 TO 10
120 Y[R,C]=X[C]
130 NEXT C
```

Those who have the String Variables ROM at their disposal can make transfers more efficiently:

```
10 DIM XI[10],YI[5,10],A$[20]

100 R=4
110 TRANSFER X[1] TO A$
120 TRANSFER A$ TO Y[R,1]
```

Sometimes (not always) this procedure saves calculator memory, but at any rate the transfer of array values takes place much faster. The operation is also possible if the array elements do not represent ASCII characters; even negative values are allowed. The string must be at least twice as long as the number of elements that are to be transferred between the arrays, and the used part of the string must correspond exactly with this number. For instance:

```
10 DIM XI[10],YI[5,10],A$[30]

100 R=4
110 TRANSFER X[1] TO A$
120 TRANSFER A$[1,20] TO Y[R,1]
```

or

```
100 R=4
110 TRANSFER X[1] TO A$[11]
120 TRANSFER A$[11] TO Y[R,1]
```

Parts of arrays can be transferred in a similar way:

```
5 DIM AI[12],QI[6,13],Z$[18]

245 L=3
250 TRANSFER Q[L,5] TO Z$[9,16]
255 TRANSFER Z$[9,16] TO A[3]
```

The elements Q(3,5) through Q(3,8) are now transferred to A(3) through A(6).

The following restrictions must be kept in mind.

1. Only integer arrays can be handled in this way.

2. Matrix rows, not columns, can be transferred via strings. (Sometimes the Matrix "TRN" statement will be helpful.)
3. The maximum length of arrays that can be transferred at one time is 127, because the maximum even string length is 254.

Determining the Centroid of a Figure

by Bob Floren, Red River Valley Potato Research Center, P.O. Box 113, East Grand Forks, Minnesota 56721, U.S.A.

This 9830A program finds the centroid of any figure traced with the HP 9864A Digitizer. The 9864A is commonly used to calculate areas and curve lengths, but the centroid capability may have been overlooked by many users.

The centroid coordinates, $\overline{X}$ and $\overline{Y}$, are closely approximated by the sums:

$$\overline{X} = \sum_{i=1}^{n} Xi\Delta Ai$$

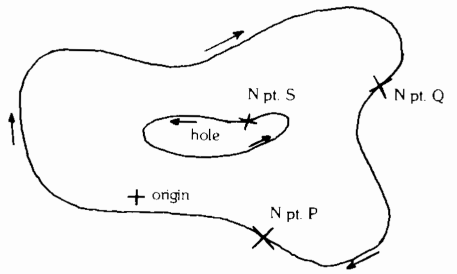
$$\overline{Y} = \sum_{i=1}^{n} Yi\Delta Bi$$

where

N = total number of points entered, at 4.5 pt./sec. rate,
ΔAi = trapezoidal area increments having the Y axis as base,
ΔBi = trapexoidal area increments having the X axis as base.

The centroid increments are calculated and summed in program lines 50 - 90. The SGN(Y) and (X-X1) terms in line 50 insure that the $\overline{X}$ increments are subtracted when X is increasing and $Y<0$, or when X is decreasing and $Y>0$. A similar rule holds for the $\overline{Y}$ increments. A clockwise tracing path around the area has been assumed.

When the area has been once circumscribed, the "O" key on the wand is pressed, which enters the (0,0) point. Lines 100 and 110 interpret this, stop data entry and cause the centroid coordinates to be printed in inches referenced to the origin. The routine of lines 500 - 520 can be used to pinpoint and mark the centroid location.



1. The origin is arbitrarily chosen, except that points S and Q must lie in the same quadrant.
2. Begin clockwise tracing at arbitrary point P on the boundary.
3. At arbitrary point Q, press "C" key on tracing wand to halt data entry. Slide wand to point S and press "C" to resume data entry and trace counter-clockwise around the hole. When point S is reached, press "C". At point Q press "C" again and resume clockwise tracing.
4. When point P is reached, press "O" key on wand to stop data entry and print the centroid coordinates.

```
10 I=1
20 A=P1=P2=0
30 ENTER (9,*)X,Y
40 IF I=1 THEN 150
50 A1=SGN(Y)*(X-X1)*ABS(Y+Y1)/2
60 B1=SGN(X)*(Y1-Y)*ABS(X1+X)/2
70 A=A+A1
80 P1=((Y+Y1)/2)*B1+P1
90 P2=((X+X1)/2)*A1+P2
100 F=X↑2+Y↑2
110 IF F<0.01 THEN 190
120 Y1=Y
130 X1=X
140 GOTO 30
150 I=2
160 X1=X
170 Y1=Y
180 GOTO 30
190 P3=P1/A
200 P4=P2/A
210 PRINT "AREA ="A,"CENTROID: X = "P4"Y ="P3
220 PRINT
230 END
500 ENTER (9,*)X,Y
510 DISP X,Y
520 GOTO 500
```

Implementing a Binary Switch

by Andrew Vettel, Jr., Steel Valley School District, 1705 Maple Street, Homestead, Pennsylvania 15120, U.S.A.

Many programmers use a variable as a flag in order to have two distinct states, usually 0 and 1. If you wish to have the state continuously alternate each time a certain line is reached in a program, the following assignment statement is useful for doing just that in a single line:

100 LET F = (F = 0)

When F's values is 0, then the logical expression F = 0 is true and has a value of one, which is then assigned to F. Conversely, if F's value is one, then the expression is false, or 0.

Further, any two values may be alternately chosen. For example, we may switch F's value between the value of A and the value of B:

100 LET F = A∗(F = B) + B∗(F = A)

Of course, if the value of either A or B were 0, one of the terms may be eliminated.

A Structured Approach To Program Overlays

by Tim D. Barringer, Ames Research Center, Moffett Field, CA 94035 U.S.A.

Many of you with 9830s have likely yielded to the temptation to write large programs requiring many overlays. If you have a mass memory, this temptation is increased and, as in this shop, you are routinely developing systems with five to 15 overlay segments. The straightforward or GOTO approach to structuring overlays results in code that looks like:

```
10 REM...MAIN PROGRAM
   .
   .
100 CHAIN "O'LAY1", 1000, 1000
   .
   .
200 CHAIN "O'LAY2", 1000, 1000
   .
   .
   etc.
```

The overlay module (after being loaded into memory) looks like:

```
1000 REM...OVERLAY1
   .
   .
1200 GOTO 110
```

There are two difficulties encountered with this structure.

1. The last GOTO in the overlay must be especially dealt with during program development. The whole overlay must be RENumbered beginning at the intended load point called out in the chain or GET command to prevent renumbering during loading, and the final GOTO must be updated to the expected return point before storing this overlay module.
2. Each time the load point line number changes, the return point line number changes, or the overlay module is required at more than one place in the main program, the overlay module must be edited to account for these changes.

   For only one or two overlays this structure is fine, but it quickly becomes burdensome after more than two overlays. An alternative approach to the structure of handling overlays is:

```
10 REM...MAIN PROGRAM
20 A=0
30 GOTO A+1 OF 40, 140...
40 REM...FUNCTION 1 STARTS HERE
   .
   .
130 CHAIN "O'LAY1", 1000, 500
140 REM...FUNCTION 2 STARTS HERE
   .
   .
240 CHAIN "O'1LAY2",  1000, 500
   .
   .
500 A=A+1
510 IFA> number of modules to process THEN 540
520 GOSUB 1000
530 GOTO 30
540 END (or possibly GOTO 20 if the program is to
    be automatically restarted)
```

This technique allows the programmer to insert, delete and renumber program lines in both the main and overlay modules without further editing or special handling of the code. The overlay module (after being loaded into memory) looks like:

```
1000 REM...OVERLAY 1
   .
   .
1200 RETURN
```

Using this method, we let the 9830's operating system take care of the return to the main program. However, the 9830 initializes its list of GOSUB returns on each CHAIN or GET command. Thus all hierarchy in program structure must be contained in the main program. This may be viewed as a benefit by structured programming buffs.

A two-level hierarchy might look like:

```
10 REM...MAIN PROGRAM
   .
   .
80 DISP "SUBSYS1, SUBSYS2, SUBSYS3";
90 INPUT A
100 IF A<1 OR A>3 THEN 80
110 GOTO A OF 120, 240, 350
120 REM...SUBSYSTEM 1 STARTS HERE
130 B=0
140 GOTO B+1 OF 150, 160, 170
150 CHAIN "O'LAY1", 1000, 180
160 CHAIN "O'LAY2", 1000, 180
170 CHAIN "O'LAY3", 1000, 180
180 B=B+1
190 if B>3 then 220
200 GOSUB 1000
210 GOTO 140
220 A=A+1
230 GOTO 100
240 REM...SUBSYSTEM 2 STARTS HER
250B=0
260 GOTO B OF 270 ,280
270 CHAIN "O'LAY4", 1000, 290
280 CHAIN "O'LAY5", 1000, 290
290 B=B+1
300 IF B>2 THEN 330
310 GOSUB 1000
320 GOTO 260
330 A=A+1
340 GOTO 100
350 REM...SUBSYSTEM 3 STARTS HERE
   .
   .
etc.
```

Even when using this method, the structure becomes quickly complicated if you attempt too many levels of hierarchy. However, this method maintains the advantage of a consistent pattern in the structure. No special modification of the code is required after simple editing, and the complexity is a function of the level of hierarchy, not the number of modules.

When using GET (or LOAD for cassettes) instead of CHAIN (or LINK), be sure to declare the module indicies A,B, etc., and include other pertinent information in a COMmon statement.

Section 3

9825

Index

Section 3 - 9825

Refill of Word-Oriented Buffers

by Sue Kolb, Hewlett-Packard, Desktop Computer Division

If you want to calculate a complex series of words to be sent repetitively to a black box and perform other calculations at the same time, you can avoid recalculating the series each time it must be sent.

Let's say that "FOON" is as follows:

"FOON"=B$= $\underbrace{\boxed{\text{XYZPQR}}}_{\text{Contents}}\underbrace{\boxed{}}_{\text{Pointers}}$

Code to create buffer:

```
dim B$ (6 + 16)
buf "FOON", B$, 2
wtb "FOON", 88 • 2 ↑ 8 + 89,
90 • 2 ↑ 8 + 80, 81 • 2 ↑ 8 + 82
```

Since "FOON' is a word-oriented buffer, we can think of its contents as pairs of characters:

X	Y	word 1
Z	P	word 2
Q	R	word 3

B$ references the same information as single characters:

X	char. 1
Y	char. 2
Z	char. 3
P	char. 4
Q	char. 5
R	char. 6

The wtb command tried to write characters into a word-oriented buffer, so it had to pad the rest of each word with blanks:

b̸	X
b̸	Y
b̸	Z

Notice that now only half of the old contents will fit; the buffer overflows.

The solution to the problem is as follows:

"FOON" = $\boxed{\text{XYZPQR}}\boxed{}$

T$ = $\boxed{\text{XYZPQR}}$

Retain two copies of the contents. When the transfer is complete and the buffer is empty, fill the buffer with blanks to reset the pointers and then copy T$ into the buffer content area. The buffer is thus refilled with the old contents and can be again transferred. The code to perform this operation, in general, follows:

0: dimB$[2N + 16], T$[2N]
  where N = number of words in buffer.
1: buf"FOON", B$, 2; oni2, "reset"
  set up buffer; when device is through with transfer, branch to service routine "reset".
2: 0 → Q
  Q is a flag indicating end-of-transfer and time to start again.
3: .... fill FOON ....
4: .... using wtb ....
5: B$[1, 2N] → T$; buf "FOON"
  set up T$; wipe out old FOON as though transfer completed.
6: "clean FOON": fmt Nx, z; wrt "FOON"
  fill FOON with blanks (sets pointers to full).
7: "reload FOON": T$ →B$
  replace blanks with old contents.
8: tfr "FOON", 2
  transfer FOON out to device 2.
9: if Q = 0; jmp 0
  wait loop (could be other parts of the main program calculations).
10: 0 →Q; gto "clean FOON"
  when interrupt has been acknowledged, reload FOON and start process again.
11: "reset": 10 → Q; iret
  set Q flag to signal interrupt.

Note: you must substitute a number for N everywhere in the above program.

In summary, you can refill byte-oriented buffers using: wtb "Buffer name", string variable associated with buffer. But word-oriented buffers require lines 5, 6 and 7 above.

How to Move a Program Line

by William Deatrick, Alexandria, Virginia

A friend of mine, Scott Layson, has discovered a useful technique. After writing a program, I sometimes find it necessary to move a line to another position. For long lines, retyping is very tiresome to say the least. The following method allows the user to move a line from one place in a program to another without retyping it. Fetch the line to be moved. Press BACK and then STORE. This places the line in the first of the two RECALL positions. Then fetch the line that is to be after the inserted one. Press RECALL. The last FETCH command will appear in the display. Press RECALL again and the line to be inserted will appear. (That double RECALL is quite useful!) Press INSERT (line) and the line is in place. Then go back and delete the unwanted line.

I would like to say how pleased I am with the 9825A. It is truly a great step forward in desktop programmables.

Filling a String with Spaces

by Howard Rathbun, Hewlett-Packard, Desktop Computer Division

It is often necessary to fill a string or portion of a string with a number of spaces. This method, which uses a for-next loop, is one way to do this:

```
0: dim A$[100]
1: for I=1 to 100
2: " "→A$[I,I]
3: next I
```

The method shown below not only uses less code, but is about 80 times faster. Note, however, that this second method can be used only for filling the entire string with spaces.

```
0: dim A$[100]
1: " "→A$[1,100]
```

Labeling Special Function Keys

by Sam Sands, Hewlett-Packard, Desktop Computer Division

To avoid guessing what your Special Function keys do, label your key files and the individual keys. Then you need only do a list k to see at a glance what each key is for. You can:

1. Put a label in front of an executable statement,
2. Use a label instead of a statement number in a CONT command, or
3. Put in a dsp statement. The key then tells you what it is doing as soon as you press it. You don't have to wait for a program to be loaded from the tape cartridge, for example, before you know whether you hit the right key.

If you should accidentally press the wrong key, press RECALL to see what key you pressed.

```
f0: *"Program                f7: * dsp "Engin
   Modification               eering";trk 1;1
   Keys":                     dp 8

f1: *"Modify                 f10: *dsp "Stati
 Variable":3→X                stics";trk 0;ld
                              p 5

f2: *cont "Entry             f18: *dsp "Graph
 Point"                       ics";trk 0;ldp
                              3

f3: *"Program
     Selection
     Keys":
```

Subroutines and Functions

by Howard Rathbun, Hewlett-Packard, Desktop Computer Division

Callable subroutines and functions should be placed at the beginning of a program for faster execution. Function and subroutine calls always search for the subprogram label starting at line 0. The time saved is not a great amount per line (about 30 µs), but for large, long-running programs the savings in time can add up.

Cataloging Files

by Eldon Brown, Hewlett-Packard Company, Bellevue, Washington, U.S.A.

The following program will produce a catalog of the files on the 9825A tape cartridge. Each file number will be printed along with the file type, current size and absolute file size. The program requires the use of the Strings, General I/O and Extended I/O ROMs.

If the file to be cataloged is a program and line zero of that program is a label (possibly the name or a description of the program), that label wil be printed instead of the file type.

The files will be cataloged starting with track zero, file zero. When the null file (the last file) on track zero is encountered, this program will automatically switch to track one and catalog the files on that track.

You can prematurely switch tracks by setting flag zero. This can be done in live keyboard while the program is running.

If you want to modify this program, make certain that variable N equals the next available program line number (in this program, 34) as it is assigned in line three. Also, ensure that the third parameter of the ldf statement (found in line 21) has the value of the line number that follows the ldf statement (in this program, 22).

```
0: "9825A Cart
   Catalog Program":
1: dim I$[106]
2: buf "lstb",I$,1
3: fxd0;34→N
4: dsp "Insert Tape,
   Press: CONTINUE";
   stp
5: 0→T;cfg0,1
6: trkT
7: spc 2;prt "Track"
   &str(T)
8: prt "--------------
   --------"
9: 0→F
10: if flg0#T;flg0
    →T;spc;gto 6
11: fdf F
12: idf A,B,C,D,E
13: if B=0 and D>0;
    prt str(F)&"--
    Empty File"
14: if B=0 and D=0;
    prt str(F)&
    "--Null File"
15: if B=1;prt str
    (F)&"--Binary"
16: if B=2;prt str
    (F)&"--Numeric
    Data"
17: if B=3;prt str
    (F)&"--String
    Data"
18: if B=4;prt str
    (F)&"--Memory
    File"
19: if B=5;prt str
    (F)&"--Key
    File"
20: if B#6;gto 26
21: ldf F,N,22
22: buf "lstb";list
    #'lst',N,N
23: red "lstb",I$
24: pos(I$,":")→X;
    X+2→X;if I$[X,X]
    #char(34);prt
    str(F)&"--
    Program";gto 26
25: prt str(F)&"-"
    &I$[X+1,X-1+pos
    (I$[X+1],char
    (34))]
26: prt" "&str(C)&
    str(D); spc
27: if D=0 and T=1
    ;spc 3; end
28: if D=0;cmf 0;
    flg0→T;gto 6
29: F+1→F
30: gto 10
31:
32: "lst":ret
    "lstb.1"
33: end
*4170
```

Detecting Missing Data in Formatted Input with the General I/O ROM

by Dr. R.K. Littlewood, Biophysics Laboratory, University of Wisconsin, Madison, Wisconsin, U.S.A.

An undocumented feature of the General I/O ROM for the 9825A allows one to differentiate between zeros and blank fields when doing formatted input. This capability is very helpful when writing statistical programs that must accommodate missing data. When a "red" statement is executed and any field designated by an "fw" specification is completely blank, the value of the corresponding item in the data list is left unchanged, rather than set to the value zero. The following test program illustrates the point. (Select code 2 is a 9883A Paper Tape Reader in this example.)

Code:

```
0: -999→A→B→C
1: fmt 3f5
2: red 2,A,B,C
3: fmt 3f5.0;
 wrt 16,A,B,C
4: gto 0
*5474
```

Input:

```
00000      00001
     0000100000
```

Output:

```
     0 -999    1
  -999    1    0
```

Note also that this General I/O ROM feature requires you to preset values to zero before doing a "red," if blanks are to be interpreted as zeros.

```
0: "START":
1: cll 'DISPLAY
   AVAILABLE
   MEMORY'
2: rcm 3
3: ldf 2,A
4: if A=2; gto
   "MEMORY
   RELOADED"
5: buf "BUFFER",
   15000,3
6: cll 'DISPLAY
   AVAILABLE
   MEMORY'
7: 2→A
8: rcf 2,A
9: ldm 3
10: "MEMORY
    RELOADED":
11: 1→A
12: rcf 2,A
13: cll 'DISPLAY
    AVAILABLE
    MEMORY'
14: stp
15: end
16: "DISPLAY AVAIL
    ABLE MEMORY":
17: dsp "MEMORY
    SIZE & AVAIL
    ABLE MEMORY ARE"
18: wait 3000
19: list -1
20: wait 3000;dsp
21: ret
*20749
```

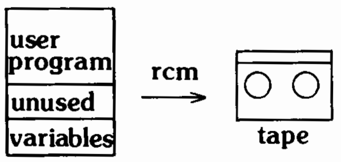
Temporary Buffer

by Don Albrecht, Ford Aerospace, Newport Beach, California, U.S.A.

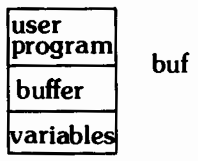
During the course of running a large program in a 9825A, it is sometimes desirable to allocate a temporary buffer, and, when the buffer is no longer needed, to reuse that area of memory. The diagram outlines the basics of the problem and its solution.

A sample program listing is also shown. This solution interrogates a data file containing either a "1", meaning the buffer is not currently allocated, or a "2", meaning that the buffer is allocated.

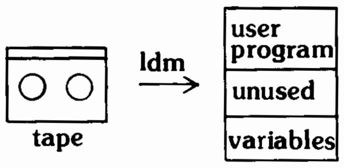
1. Need temporary buffer:
   Memory before buffer is needed.



2. Establish buffer:
   Buffer attached for I/O speed



3. Buffer no longer needed:
   Buffer is gone



Memory Files

by George B. Bosco, Jr., Bosco Engineering, Whittier, California, U.S.A.

You can fool your HP 9825A Opt. 001 or Opt. 002 desktop computer into thinking it has no optional memory.

1. Find an HP 9825 without optional memory. Install the ROMs you plan to use. Enter a one-line dummy program (such as 0: "OPT000":). Create a 7990 byte memory file on tape, or a 7478 byte memory file on a diskette.
2. Put your ROMs in your Opt. 001 or Opt. 002 machine. Turn the 9825A on and load the memory file of step 1. Press RESET.
3. Your 9825 now has no optional memory. Load small programs and/or data. Run programs. Record memory programs and later recall them. In other words; use as a calculator without optional memory. Do not execute "erase a" (erase all).
4. To restore 9825A to normal full memory size: execute "erase a" or turn machine off and then on.

You save file space and time by using this technique if your programs are small enough to run in this size memory. Similarly, use a 9825 with Opt. 001 memory to create an "Opt. 001" memory file for use in an Opt. 002 machine. To create the "Opt. 000" & "Opt. 001" memory fies using your Opt. 002 calculator ask your HP Service Representative about memory switches.

Changing Number Format

by Ricardo Casado, Los Ruices, Caracas, Venesuela

This program changes the numeric format xxxxx.xx to the format xx.xxx,xx as is used in much of the world.

```
0: "Number
   Formatter": dim
   N$[15]
1: ent "Number?",N
2: N→r5;gsb "FORMAT"
3: fmt 1,20x,
   f12.2,5x,c15
4: wrt 6.1,N,N$
5: gto 1
6: end
7: "FORMAT":
8: " "→N$[1,15]
9: int (r5/1000)
   →r2
11: int (r2/1000)
    →r3
12: int (r4/1000)
    →r4
13: fxd 0
14: str ((r5-r1)*
    100)→N$[13,15]
15: ","→N$[13,13]
16: str ((int(r5)/
    1000-int(r2))*
    1000)→N$[9,12]
17: "."→N$[9,9]
18: str ((int(r2)/
    1000-int(r3))*
    1000)→N$[5,8]
19: "."→N$[5,5]
20: str ((int(r3)/
    1000-int(r4))*
    1000)→N$[1,4]
21: fxd 2
22: if N$[15,15]=
    " ";N$[14,14]→
    N$[15,15];
    "0"→N$[14,14]
23: if N$[12,12]=
    " ";N$[11,11]→
    N$[12,12];N$
    [10,10]→N$[11,
    11];"0"→N$
    [10,10]
24: if N$[12,12]=
    " ";N$[11,11]→
    N$[12,12];
    "0"→N$[11,11]
25: if N$[8,8]=" "
    ;N$[7,7]→N$[8,
    8];N$[6,6]→N$
    [7,7];"0"→N$
    [6,6]
26: if N$[8,8]=" "
    ;N$[7,7]→N$[8,
    8];"0"→N$[7,7]
27: if N$[4,4]=" "
    ;N$[3,3]→N$[4,
    4];N$[2,2]→N$
    [3,3];"0"→N$
    [2,2]
28: if N$[4,4]=" "
    ;N$[3,3]→N$[4,
    4];"0"→N$[3,3]
29: if val(N$[2,4])
    =0;" "→N$[5,5]
30: for C=2 to 4
31: if N$[2,2]=" "
    and val(N$[C,C])
    #0;ret
32: if val (N$[C,C])
    =0;" "→N$[C,C]
    ;next C
33: if val (N$[6,8])
    =0;" "→N$[9,9]
34: if N$[2,4]#"   "
    ;ret
35: for C=6 to 8
36: if N$[6,6]=" "
    and val(N$[C,
    C])#0;ret
37: if val(N$[C,C])
    =0;" "→N$[C,C]
    ;next C
38: if val(N$[10,
    12])=0;" "→N$
    [13,13]
39: if N$[6,8]#"   "
    ;ret
40: for C=10 to 12
41: if N$[10,10]=
    " " and val(N$
    [C,C])#0;ret
42: if val(N$[C,C])
    =0;" "→N$[C,C]
    ;next C
43: if N$[10,12]#
    "   ";ret
44: for C=14 to 15
45: if val(N$[C,C])
    =0;" "→N$[C,C]
    ;next C
46: ret
*5681
```

READ/DATA Capability in HPL

by Joseph Pepin, Western Electric Engineering Research Center, Princeton, New Jersey, U.S.A.

A deficiency of HPL as compared to BASIC is that it lacks the capability of storing data within the program section. A program that requires a large amount of constant data is awkward in HPL. You must either enter many lines like 1→A, or write a program to accept the data from the keyboard and put it on tape or floppy disc. BASIC has a "READ" statement that does a read-from-memory of data contained in "DATA" statements.

It is possible to duplicate this capability in HPL using a surprising property of the "list" statement as reported by Howard Rathbun in a DCD paper available from *Keyboard.* This involves using the Advanced Programming ROM's single-quote function to return a string to the General I/O Programming ROM's "list#" statement. This string is actually the name of a buffer set up using the Extended I/O Programming ROM. In this manner, the "list" statement is tricked into using a buffer, something not otherwise allowed by the syntax.

To synthesize a "READ/DATA" capability, another single-quote function is used, this time within a "red" statement. Besides returning the buffer name, this function searches through memory for dummy "data" statements, using the "list#" statement and another single-quote function.

The dummy "data" statement is simply a long label containing the characters "data" and a list of data items. When the function finds one, it blanks out the "data" and statement number and returns the buffer name to the "red" statement. The "red" statement uses this buffer as a fast read/write buffer and reads the data from it.

The following short program demonstrates this technique. The program requires the String, Advanced Programming and Extended I/O ROMs.

```
0: dim D$[100]
1: buf "data",D$,3
2: red 'DATA',A,B,C
3: prt A,B,C
4: red 'DATA',D,E,F
5: prt D,E,F
6: stp
7: "DATA":list#'dat
   ',r0,r0
8: r0+1→r0
9: if (pos(D$,": "
   "data")→r1)=0;
   buf "data";gto
   -2
10: ""→D$[1,r1+7]
11: ret "data."&
    char(p1+48)
12: "dat":ret
    "data.1"
13: "data 1.1,2.2,
    3.3":
14: "data 5,6,7":
15: end
```

Line 0: Dimensions a string that is going to hold a line of the program listing.

Line 1: Establishes a buffer named "data", which is the string already dimensioned in line 0, and is Type 3, byte-oriented fast read/write buffer.

Line 2: Reads three items of data into A, B, and C. Calling the function 'DATA' will return the buffer name "data", after the function has placed the data into the buffer.

Line 3: Prints the data just read.

Lines 4 and 5: Similar to lines 2 and 3, show that the 'DATA' function automatically positions a pointer to the next set of data.

7: The 'DATA' function: Uses r0 as a line pointer, and lists a single line of program into the buffer, whose name is returned by the 'dat' function.

Line 8: Advances the line pointer. Originally at 0, it will point to the next line the next time the 'DATA' function is called. Setting r0 to 0 will mimic the BASIC RESTORE statement.

Line 9: Checks to see if the program line just read in was a dummy data statement. If not, it empties the buffer and tries again.

Line 10: The program line just read in was a dummy data statement. Blank out the statement number and the "data". The red statement reads the remainder of the data statement as if from an external device.

Line 11: Returns the name of the buffer to the red statement. An optional format number may be enclosed within parentheses after a 'DATA' call. Otherwise the standard format is used.

Line 12: The 'dat' function: Returns the name of the buffer to the list# statement on line 7. The .1 ensures that the listed line contains no extra line feeds nor the check sum.

Lines 13 and 14: These are dummy data statements containing the data in a long label.

Erasing 9825A Tape Cartridges

by Jackye Churchill, Hewlett-Packard Company, Desktop Computer Division

Error 43 can occur during an ERT (erase tape) operation to signify either a tape transport failure or an unexpected end-of-tape. Normally, this is due to an incomplete erasure caused by dirt on the tape or a loss of contact between the tape and the tape head during high speed movement. However, action must first be taken to determine if the error 43 was caused by tape transport failure. This can be done by rewinding the tape. If error 43 occurs again, it can be assumed the drive has failed. If error 43 is not caused by transport failure, the following steps can be taken to correct the problem.

The first step is to clean the tape head and capstan. Then rewind the tape and execute ERT. These two procedures can be repeated if necessary.

If the erasure remains incomplete, there still may be dirt on the tape. Several high speed end-to-end operations may be executed in an atttempt to free the tape of dirt. The end-to-end operation is accomplished by executing these two operations:

```
rew
fdf 1000
```

If this procedure does not complete the erasure and eliminate error 43, the tape should be discarded.

A Fix for Backup Copy Command

by Alberto Rodriquez, Condado Santurce, Puerto Rico

If 9825 users have ever tried to do a backup using the copy 0$, D, S, N$, D, S format of the copy command, they may have realized that although the new file is created, the file, in effect, contains unreadable data. One could easily miss this until there is need of using the backup file. Only the format using string (or substring) variables as file names has this problem, so a ready subterfuge is available:

```
drive0;renm0$, "FILE1"
copy "FILE1",D,S,"FILE2",D,S
drive1;renm "FILE2",N$
drive0;rnm "FILE1",0$
```

Although cumbersome, this is the only way to do this backup if the string variable name must be used.

Instrument Approach and Landing Game

by Chris Mills, Cook, Australia

Frequently during program execution it is valuable to be able to modify a variable.

Extended I/O ROM

Enter this program:

```
0: rdi 4→A;dsp A;jmp 0
```

Now press RUN and you will see a free running display. I think interface 4 is the register which holds the result of the machine scan of the keyboard. This can be used to advantage in 'real time' simulations. The following game program simulates an aircraft making an instrument approach and landing. As the program continually loops, rdi 4 is used to scan the keyboard for control inputs which modify variables and hence the power setting, pitch and azimuth angles. Lines 29, 32 and 35 are the control inputs. Note that the rdi 4 statement is used twice to check to see if a key is being held down. Although this simulation was written for fun, the rdi 4 statement could be used in many "serious" applications, e.g. in control applications you could vary the value of variables to increase a temperature limit or change a motor speed.

Conclusion

I have only used the rdi 4 statement in games but can see that is could be a great help in control applications.

```
0: "Landing
   Program":
1: prt "LANDING ";
    spc 2
2: fxd 0;dsp "press
   'CONT' to go";
   gto "ent"
3:
4: "dec":str(p1)→B$
   ;B$[2]→A$[p2,p2+
   len(B$)-2];ret
5:
6: "rnd":
7: rnd(1)→Q;rnd(1)→
   P;if P>.5;-Q→Q;
   ret
8: ret
9:
10: "vec":sin(O)→W;
    √(1-WW)→X;sin
    (S)→Y; √(1-YY)
    →Z
11: H+J(VW+F)→H
12: I-J(VZX+E)→I
13: G+J((VY+r1)+L)
    →G
14: V+J((T-VV)/M-
    50W)→V
15: if H<0;gto "com"
16: if V<150;beep;
    if V<145;dsp
    "you stalled";
    wait 2000;
    gto "end"
17:
18: "tur":
19: if H<500;.5R→R
20: cll 'rnd';RQ+F
    →F;cll 'rnd';
    RQ+E→E; cll '
    rnd';RQ+L→L
21:
22: "ang":
23: asn(H/I)→B
24: asn(G/I)→C
25: asn(F/V)→O
26: asn(r1/V)→S
27:
28: "cont":
29: rdi 4→X;rdi 4→
    Y;if X#Y;gto +3
```

```
30: if X=80;O+1→O;
    T-.1T→T
31: if X=88;O-1→O;
    T+.1T→T
32: rdi 4→X;rdi 4→
    Y;if X#Y;gto +3
33: if X=78;S+1→S
34: if X=89;S-1→S
35: rdi 4→X;rdi 4→
    Y;if X#Y;gto +5
36: if X=81;T+.2T→T
37: if X=79;T-.2T→T
38: min(T,1e6)→T
39: max(400,T)→T
40:
41: "dsp":
42: A+B→B
43: if B>1;char(10)
    →A$[16,16]→A$
     [17,17] →A$[18,
    18];gto +8
44: if B>.5 and B<
    =1;char(10)→A$
     [16,16] →A$[18,
    18];" "→A$[17,
    17];gto +7
45: if B>.2 and B<
    =.5;char(10)→A$
     [17,17] ;" "→A$
     [16,16]→A$[18,
    18];gto +6
46: if B<=.2 and B>
    =-.2;"***"→A$
     [16,18]; gto +5
47: if B<-1.5;char
    (222)→A$[16,16
    ]→ A$[17,17]→A$
     [18,18];gto +4
48: if B>-1.5 and
    B<=-1;"↑↑↑"→A$
     [16,18]; gto +3
49: if B>-1 and B<
    =-.5;"↑ ↑"→A$
     [16,18]; gto +2
50: " ↑ "→A$[16,18]
51: "   "→A$[13,15]
    →A$[19,21]
52: if C>1.5;"→→→"
    →A$[13,15];
    gto +7
53: if C>1 and C<
    =1.5;" →→"→A$
     [13,15]; gto +6
54: if C>.5 and C<
    =1;"  →"→A$[13,
    15]; gto +5
55: if C<=.5 and C>
    =-.5;"+"→A$[15,
    15]→ A$[19,19];
    gto +4
56: if C<-.5 and C>
    -1;char(13)→A$
     [19,19];"  "→A$
     [20,21];gto +3

57: if C<-1 and C>=
    -1.5;char(13)→
    A$[19,19]→A$
     [20,20];" "→A$
     [21,21];gto +2
58: char(13)→A$[19,
    19]→A$[20,20]→
    A$[21,21]
59: -S→r2;if r2<0;
    360+r2→r2
60: if C>.2;cll
    'dec'(r2,19)
61: if C<-.2;cll
    'dec'(r2,13)
62: "K   "→A$[1,4];
    cll 'dec'(V/
    1.69,2)
63: " "→A$[5,11];if
    F<=0;"D"→A$[6,
    6]; "-"→A$[11,
    11];gto +2
64: "C"→A$[6,6];"+"
    →A$[11,11]
65: cll 'dec'(60int
     (abs(F)),7)
66: "A   "→A$[23,
    27];cll  'dec'
     (int(H),24)
67: " R   "→A$[28,
    32];cll  'dec'
     (.00164I,30)
68: "|"→A$[12,12]→
    A$[22,22]
69: dsp A$;gto
    "vec"
70:
71: "ent":
72: 0→X;ent "Instr
    uctions? 1=yes,
    CONTINUE=no",X;
    if X;cll 'ins'
73: dim A$[32],B$
     [20];-3→A→B→O;
    .19→J; 1e4→M;
    2.25e4→T;60000
    →I
74: cll 'rnd';3000
    +10000→H;cll
    'rnd'; 30000→G
    ;cll 'rnd';250
    +1000→V
75: enp "AIR TURBU
    LENCE ? (1 to
    25)", R;R/10→R
76: enp "X WIND KTS
    ?(left neg,
     right pos)",L;
    1.69L→L;L/10→L
77: sfg 14;" "→A$
     [1,32];lkd;gto
     "vec"
78:
79: "com":
80: abs(G)→G;1200-I
    →I

81: prt "You landed"
    ;prt G;prt
    "feet off center
    -";prt "line
    at";prt V/1.69
82: prt "Kts.";spc
    2;prt "Landing
    was"; prt I;prt
    "feet from the"
83: prt "aim point
    with a ";prt
    "vertical speed
    =";prt F,"feet
    /sec"
84: prt "Fighter
    Pilots";prt "
       Do It";prt "
          Better!"
    ;spc 5
85: if abs(G)>300;
    dsp "You landed
    off the runway!"
    ;wait 2000;gto
    "end"
86: if I<0;dsp "You
    landed short!"
    ;wait 2000;gto
    "end"
87: if I>4000;dsp
    "You ran off the
    runway!";wait
    2000;gto "end"
88: if F>-10;dsp
    "Nice landing"
    ;stp
89: if F<=-10 and
    F>-20;dsp "a
    bit heavy!";stp
90: if F<=-20 and
    F>-50;dsp
    "CRUNCH!!!!!!"
    ;stp
91: "end":dsp
    "wreckage is
    now burning"
    ;stp
92:
93: "ins":
94: prt "instruc
    tions";prt "you
    are landing";
    prt "on a run
    way on a"
95: prt "heading of
    north";prt
    "(000- 360).";
    prt "to control
    the"
96: prt "aircraft
    use:";prt "'0'
   =LEFT"; prt "'
    ,'=RIGHT"

97: prt "'.'=DOWN"
    ;prt "'2'=UP";
    prt "'1'=POWER
    OFF"
98: prt "'3'=POWER
    ON";prt "air
    craft stalls";
    prt "at 85
    Kts.";spc 2
99: prt "K=Speed
    (knots) ";prt
    "DorC=Vertical
    vel";prt "A=
    altitude(ft)"
100: prt "R=range
     in naut-";prt
     "ical miles
     *10"
101: prt "→→ means
     fly";prt
     "right";prt
     "↑↑ means fly
     up"
102: prt "and so
     on.";prt "head
     ing is dis-";
     prt "played
     when off"
103: prt "center
     line.  ";prt
     "happy land
     ings";spc 5;
     ret
*14535
```

Section 4

9820 and 9821

One-Line Averaging

by Philip A. Dawty, Lansing, Michigan

The following one-line program for the 9820 averages N numbers. END RUN PROGRAM should be pressed before each series of numbers is entered. This causes printing 0.0000, which can be ignored.

```
0:
FXD 4;PRT A;B+A→
B;C+1→C;ENT "N",
A;GTO 0;IF FLG 1
3;PRT "TOT:",B,"
AV:",B/(C-1);
SPC 8;TBL 5⊢
R394
```

Use of Card Reader and Printer To List Cards

by Bob Huston of Surface Effect Ship Test Facility, U.S. Naval Air Station, Patuxent River, Maryland

Following is some information we have discovered in our use of the HP 9820A Calculator with 9866A Printer, 9862A Plotter, 9869A Card Reader, and Peripheral Control ROMs I and II.

1. Load Cards into reader.
2. Transfer 1.8 (PC II).
3. EXECUTE.
4. CONTINUOUS PICK (on 9869).

This will list cards on the printer. We have been using this feature to list 80-column cards containing Fortran programs. We have the punched card option on the reader.

If FMT "AD"; WRT 1 is executed, and then RDB1→√R( ), a decimal code will be returned to the register that is the decimal equivalent of the ASCII. For instance, a space returns a 32, C is 40, 48 through 57 are digits 0 through 9, 65 through 90 are A through Z, and so forth. A 10 is found at the end of a card. By looping back to the RDB command and not the FMT, an entire card can be read in and decoded. This feature can be used to sort cards with the select hopper option on the card reader and will work on alpha or numeric data.

In our application we use the card reader and plotter to produce report-quality plots. In order to make the lettering of plots automatic, we use the routine mentioned above. All plot heading data and plot points are put on cards by a computer. The plots are done completely by the 9820, including lettering. Heading data is read into the calculator one column at a time. It is decoded using the short program given below and plotted using the plot commands of the PC I ROM.

We also use this method for special lettering of plots. It allows us to keypunch lettering and have the plotter produce high quality, finished work.

```
0:
"ONE";CFG 1;RDB
1→R9;IF R9≠10;
LTR R30,R33,211;
GSB "PRT"⊢
1:
IF FLG 1;GTO "ON
E"⊢
2:
GTO "NXT"⊢
3:
"PRT";SFG 1;IF R
9=42;32→R9⊢
4:
IF R9<31;32→R9⊢
5:
IF R9>32;IF R9<3
9;32→R9⊢
6:
IF R9>57;GTO "L"
⊢
7:
"R";IF R9=32;
PLT "";GTO "N"⊢
8:
IF R9=40;PLT "("
;GTO "N"⊢
9:
IF R9=41;PLT ")"
;GTO "N"⊢
10:
R9-42→R9;JMP R9⊢
11:
PLT "+";JMP 15⊢
12:
PLT ",";JMP 14⊢
13:
PLT "-";JMP 13⊢
14:
PLT ".";JMP 12⊢
15:
PLT "/";JMP 11⊢
16:
PLT "0";JMP 10⊢
17:
PLT "1";JMP 9⊢
18:
PLT "2";JMP 8⊢
19:
PLT "3";JMP 7⊢
20:
PLT "4";JMP 6⊢
21:
PLT "5";JMP 5⊢
22:
PLT "6";JMP 4⊢
23:
PLT "7";JMP 3⊢
24:
PLT "8";JMP 2⊢
25:
PLT "9"⊢
26:
GTO "N"⊢
27:
"L";IF R9<64;32→
R9;GTO "R"⊢
28:
IF R9>90;32→R9;
GTO "R"⊢
29:
R9-64→R9;JMP R9⊢
30:
PLT "A";JMP 26⊢
31:
PLT "B";JMP 25⊢
32:
PLT "C";JMP 24⊢
33:
PLT "D";JMP 23⊢
34:
PLT "E";JMP 22⊢
35:
PLT "F";JMP 21⊢
36:
PLT "G";JMP 20⊢
37:
PLT "H";JMP 19⊢
38:
PLT "I";JMP 18⊢
39:
PLT "J";JMP 17⊢
40:
PLT "K";JMP 16⊢
41:
PLT "L";JMP 15⊢
42:
PLT "M";JMP 14⊢
43:
PLT "N";JMP 13⊢
44:
PLT "O";JMP 12⊢
45:
PLT "P";JMP 11⊢
46:
PLT "Q";JMP 10⊢
47:
PLT "R";JMP 9⊢
48:
PLT "S";JMP 8⊢
49:
PLT "T";JMP 7⊢
50:
PLT "U";JMP 6⊢
51:
PLT "V";JMP 5⊢
52:
PLT "W";JMP 4⊢
53:
PLT "X";JMP 3⊢
54:
PLT "Y";JMP 2⊢
55:
PLT "Z"⊢
56:
"N";R30+1.25→R30
;RET ⊢
R324
```

Entry Space Saving

by D. J. Harley, John Wilson and Partners, Brisbane, Australia

Frequently it is desired to print input data as it is entered. This often results in the duplication of alphanumeric strings in DISPLAY and PRINT statements. Trials to find ways of eliminating this duplication led to the following:

When a STOP instruction is executed following a PRINT statement the printed information also appears in the display thus serving as the alpha part of an enter statement. Data may then be entered in the normal way and the implied Z store operates so that the input value enters the Z-register, i.e.,

Conventional **Modified Method**

```
BREADTH (INS) =          BREADTH (INS) =
            1.10                     1.10
LENGTH (INS) =           LENGTH (INS) =
            2.20                     2.20
DEPTH (INS) =            DEPTH (INS) =
            3.30                     3.30

0:                       0:
FXD 2;ENT "BREAD         FXD 2;PRT "BREAD
TH (INS) =",A,"L         TH (INS) =";STP
ENGTH (INS) =",B         ⊢
,"DEPTH (INS) ="         1:
,Z⊢                      PRT Z→A,"LENGTH
1:                       (INS) =";STP ⊢
PRT "BREADTH (IN         2:
S) =",A,""LENGTH         PRT Z→B,"DEPTH (
(INS) =",B,"DEPT         INS) =";STP ⊢
H (INS) =",Z⊢            3:
R408                     PRT ⊢
                         R414
```

Note:
- a. A saving of 6 registers.
- b. The ‘PRT’ in line 3 of the modified method. It is not necessary to say PRT Z. This applies with a normal input statement too, i.e., ENT “X-”, X; PRT;... will cause the entered value to be printed

A similar technique may be applied using DISPLAY statements where it is not desired to print everything as in array input or with questions. The following example illustrates this for a series of questions (options) where the code is: RUN PROGRAM to say ‘no’: *Any number* RUN PROGRAM to say ‘yes’, i.e., to select the option. Repeated pressing of RUN PROGRAM will cause a cycle through the options available.

Conventional **Modified Method**

```
0:                       0:
CFG 13;ENT "NEW          0→Z;DSP "NEW CIR
CIRCLE?",Z;IF            CLE?";STP ⊢
FLG 13=0;JMP 50⊢         1:
1:                       IF Z≠0;JMP 53⊢
CFG 13;ENT "NEW          2:
SLICE WIDTH?",Z;         DSP "NEW SLICE W
IF FLG 13=0;JMP          IDTH?";STP ⊢
10⊢                      3:
2:                       IF Z≠0;JMP 12⊢
CFG 13;ENT "NEW          4:
SOIL PROPS?",Z;          DSP "NEW SOIL PR
IF FLG 13=0;JMP          OPS?";STP ⊢
20⊢                      5:
3:                       IF Z≠0;JMP 21⊢
CFG 13;ENT "NEW          6:
RUN?",Z;IF FLG 1         DSP "NEW RUN?";
3;JMP -3⊢                STP ⊢
R389                     7:
                         IF Z=0;JMP -7⊢
                         R392
```

Note the saving of 3 registers.

Recovering A “Lost” Program From A Tape File

by Arthur F. Graf, San Antonio, Texas

I once “lost” a very long, complicated program. The wrong file identifier appeared at the heading of program file 20, and I did not want to spend several hours re-entering and editing. Here is a method to recover such a “lost” program.

Clear calculator.
Load in about 10 lines of GTO +1
Then stack other programs in the memory until this new “program” is slightly longer than the “lost” program.
GTO 0
RCF 20
The instant the new heading has been recorded on the tape, open the cassette door. Remove the tape and shut off the calculator.
Restart and initialize.
LDF 20
When the machine detects an error in loading and starts to rewind, press STOP and hold until operations cease.
CLEAR
GTO 0
LIST

The first few lines will be GTO +1 and other irrelevant data. The end of the program also may contain irrelevant data. Edit out this data and replace missing lines. The bulk of the program should have been loaded intact.

Arctan Between ± 180 Degrees

by Dr. Anthony F. Gangi, Professor of Geophysics, and L. David Jones, graduate student, both of Texas A&M University, College Station, Texas

Their tip involves calculating the phase angle of a complex number (i.e., taking the inverse tangent of a ratio) so that it lies between ± 180 degrees. The inverse tangent routine on the 9820 Math ROM gives an answer between ± 90 degrees. This is because the inverse tangent is multivalued. However, when the signs of the numerator and the denominator are individually known, as in the case of complex numbers, the proper quadrant can be determined for the inverse tangent. The algorithm is based on the following:

given (1) a complex number

$$z = x + iy$$

and (2) the phase of the complex number

$$\theta = \mathrm{TAN}^{-1}(y/x),$$

then the phase angle in degrees, radians, or grads can be found by using the following one line of code (Table 1, Table 2, or Table 3, respectively, must be set; assume X in x and Y in y, then θ will be in A):

```
0: SFG 14, ATAN (Y/X) −2 ATN 1E99
   (0 > X) [(0 > Y) − (0 ≤ Y)] → A⊢
```

The need for SFG 14 is to avoid NOTE 10 for 90°, $x = 0$, $y > 0$ and −90°, $x = 0, y < 0$.

Stored Data Printout

It is often useful to examine the quantities stored in the 9820's data registers without manually searching through the memory. This short program scans the available memory and prints out only the contents of all data registers that contain non-zero values. The alpha register contents are printed first, followed by the R( ) registers in ascending order through R402.

The program listed here is for the expanded internal memory, either without plug-in ROMs or with the Mathematics ROM. For other ROM and/or memory configurations, lines 8, 9, and 10 may be edited easily to scan all of the available data registers.

This program can be entered to replace the one using the stored data in order to list the data, after which the original program can be reentered with the data still intact. The data can also be recorded on a magnetic card for easy reentry, as shown on pp. 5 - 40 of the 9820 Operating and Programming Manual.

Instructions

1. END EXECUTE LOAD EXECUTE
2. END RUN
3. Identification of all data storage registers containing non-zero values and their contents are printed.

Program Listing

```
0:
PRT " DATA REGIS
TER"," CONTENT
S"⊢
1:
FLT 9;SPC 2⊢
2:
IF A≠0;PRT "A=",
A⊢
3:
IF B≠0;PRT "B=",
B⊢
4:
IF C≠0;PRT "C=",
C⊢
5:
IF X≠0;PRT "X=",
X⊢
6:
IF Y≠0;PRT "Y=",
Y⊢
7:
IF Z≠0;PRT "Z=",
Z⊢
8:
0→R403⊢
9:
IF RR403≠0;FXD 0
;PRT R403;FLT 9;
PRT RR403⊢
10:
IF (R403+1→R403)
<402;GTO -1⊢
11:
SPC 8⊢
12:
END ⊢
R398
```

Example

```
  DATA REGISTER
    CONTENTS

A=
-1.414213562E 00
C=
 1.732050808E 00
X=
-8.366600265E-35
Z=
-1.824828759E 00
              21
 2.300000000E-04
             211
 9.800000000E-66
             331
-2.808914381E 01
             402
-3.316624790E 00
```

Foolproof Data Entry Line

by Richard Trommer, Kew Gardens, New York

When the 9820 comes to an 'ENTER' statement it stops and waits for data. If the user does not enter any data the program continues using the number that was stored in the variable previously. But if you ignore an 'ENTER' statement like this, flag 13 automatically is set. You can use this to your advantage. At the end of an 'ENTER' statement simply press the following: IF FLG 13; CFG 13; JMP 0. If this is done the calculator will not be satisfied until data is entered. This avoids the possibility of accidentally running a program with incorrect data.

Divisibility Test

by Richard Trommer, Kew Gardens, New York

Many times while programming the programmer discovers that he would like to test a number for divisibility by another number. This can be accomplished very easily on the 9820. Suppose you want to see if A is divisible by B. The following expression is equivalent to saying 'if A is divisible by B':

IF (INT(A/B)*B) = A;

The Math ROM or equivalent is needed for this test.

Transferring Program Lines

by Professor Anthony F. Gangi, Professor of Geophysics, Texas A&M University, College Station, Texas, U.S.A.

An important operation in editing programs in the Hewlett-Packard 9820 Programmable Calculator is not specified in any of the operating manuals. This is the operation of moving lines from one part of the program to another without rewriting them. If lines are long or a large number of lines are to be shuffled, it is time consuming (and error producing) to rewrite the lines to insert them in the proper place.

It has been found that it is possible to reshuffle lines on the 9820 without rekeying the lines. This is performed in the following way: Consider the simple program shown on the listing; assume we wish to take line 4 and insert it in front of line 1 of the program; that is, we wish to make line 4 into line 1, line 1 into line 2, etc. without rekeying line 4 into the calculator. The operation GTO 4 is executed from the keyboard and the line is *recalled* to display. The *back* key is pressed once to elimate the *end of line* (⊢) symbol and the instructions ;GTO 1 are keyed into the machine. At this point the display is:

4:4 → R4;GTO 1

This line is then executed by pressing the *execute* key and then the *back* key; the display then becomes

4 → R4; GTO 1

but now the program line counter is at line 1. Now the *back* key is pressed three times to eliminate the symbols (;GTO 1) and then *insert* and *store* keys are pressed in succession.

If the program is listed now (after *end* and *execute* are pressed) the program will be modified as shown. It can be seen that line 4 has been inserted in front of the original line 1 which has now become line 2.

Some precautions must be observed in transferring complex lines. For example, when a line contains a JUMP statement, replace it with DISPLAY until the line is transferred. Lines containing IF statements may be transferred using this technique by either satisfying the IF conditions before the line is transferred or by inserting the line readdressing command preceding the first IF statement, then deleting it after execution.

Normal Mode

```
ORIGINAL      MODIFIED
 PROGRAM       PROGRAM
  0:            0:
  0→R0⊢         0→R0⊢
  1:            1:
  1→R1⊢         4→R4⊢
  2:            2:
  2→R2⊢         1→R1⊢
  3:            3:
  3→R3⊢         2→R2⊢
  4:            4:
  4→R4⊢         3→R3⊢
  5:            5:
  5→R5⊢         4→R4⊢
  6:            6:
  END ⊢         5→R5⊢
  R417          7:
                END ⊢
                R416
```

Trace Mode
Recall Line 4

```
GTO 4⊢
4:
4→R4⊢
```

Press Back (3 Times)
Press Insert Store

```
1:
4→R4⊢
```

Modify Line 4
And Execute

```
4→R4:GTO 1⊢
         4.00
```

Incrementing Logarithmic Scales

by James Lovell of AIL Division of Cutler-Hammer, Long Island, New York

I have had a number of occasions in which I needed a logarithmic scale. Without thinking too deeply, I simply incremented the independent variable in the usual way, say A + 1 → A. Unfortunately, with a logarithmic scale the increments get closer and closer, and one never knows whether to stay and wait or go for a cup of coffee while it plots. Recently I realized that if I simply increment with A + A →A, the independent variable doubles in value with each step, the steps along the independent variable axis are uniform on the plotter, and the plot is soon over. The plots have sufficient detail for my purposes, but variations on this could give more or less detail, such as A + A/2 → A and A + 2A → A. Now someone else can have the machine while I have my coffee.

Multiple Execution of Single Line

by James N. Shapiro and Dr. Anthony F. Gangi of Texas A&M University, College Station, Texas

The technique relies on the 9820's buffer being able to contain a large number of statements at one time, and consists of executing a single line as many times as desired manually. An important feature of this procedure is that it may be performed without modifying program or register memory. Single line execution can often be used to advantage during program execution at an ENTER statement stop.

The single line execution technique is particularly valuable when program memory is full, as it requires no additional storage. It may also be used to advantage during execution of an ENTER statement. In this case program operation need not be interrupted.

Two examples are given below:

1. The following single line may be executed repeatedly to print out the addresses and contents of sequential non-zero registers. (A suitable register, in this case Z, is first initialized to one less than the address of the desired starting register.)
   Z+1;IF RZ ≠ 0; FIXED 0; PRT Z; FLOAT 9; PRT RZ; SPC.
   Each time EXECUTE is pressed the next register will be printed out if it is non-zero. No printout occurs in the case that the register is zero.
2. The following program will load 1 → R11, 4 → R12, 9 → R13, etc., with Z initialized to 10: Z+1; (Z-10)(Z-10) → RZ. Keep pressing EXECUTE until the last register is filled.

The above technique is a real time saver and in some cases, i.e., when the memory is full, invaluable.

Integer X Without Math ROM

by Professor Anthony F. Gangi, Texas A&M University

This is an improved technique for obtaining INTEGER X using the 9820 without a Math ROM. It has the advantage of working with negative numbers and numbers in the range 0 to ±1.5, which were not usable with the originally published technique.

Line 3 in the program shown here contains the main INTEGER X routine. Lines 1 and 2 take into account the special case where X = ±1.4999999999. In this case line 3 would otherwise fail, since the Model 20 would round this to ±1.5 for display, printing, and comparison purposes, but retain the exact input number for calculations. X would then be reduced to ±0.999999999, resulting in INT X = 0. Lines 1 and 2 circumvent this and make the function continuous for all positive and negative numbers where $|X| \leqslant 10^{10}-1$.

Integer X Program

```
0:
ENT "NONINT X =
?",X;PRT "NONINT
 X =";PRT X;SPC
⊢
1:
IF X=1.5;1→X⊢
2:
IF X=-1.5;-1→X⊢
```

```
3:
PRT "INT X =",X-
.5((1.5≤X)-(X≤1
.5))+1E11-1E11;
SPC 3⊢
4:
GTO 0⊢
5:
END ⊢
```

"Table" Identification

by D. L. Schacher of Tel-Instrument Electronics Corp., Carlstadt, New Jersey

The one-line program below determines for which trigonometric units (degrees, radians, or grads) the Math ROM trigonometric functions are set in the 9820 or 9821 calculator. This is used in the beginning of a subroutine, before setting the table needed for the subroutine. The trigonometric functions can then be reset to the original TABLE 1, 2, or 3 before leaving the subroutine.

```
0:
(SIN 180=0)+2(0>
SIN 180)+3(SIN 1
80>0)→X⊢
```

Logical Comparison

by Dr. R. K. Littlewood, University of Wisconsin, Madison, Wisconsin

The 9820 performs logical comparisons by rounding both operands to 10 digits of significance before executing any logical operation (=, ≠, >, ≤). Thus, for example, tests for equality may not be executed properly for pairs of numbers differing only in the eleventh or twelfth digit. However, testing for the equality of their difference to the value zero will work correctly.

9820 Data Storage

In many applications the use of magnetic cards in loading and recording data in the 9820 is an infrequent occurrence, so a few reminders may help. Aside from the syntax for loading data, LOD "DA" EXECUTE and recording data, REC "DA"R( ) EXECUTE it is useful to know that the highest register specified when recording data should be just the highest one needed. This minimizes the number of magnetic card sides required.

If a blank magnetic card is used accidentally with the load data instructions, this will not change the value of data already existing in the storage registers. This operation will produce a NOTE 14 when the card finishes going through the card reader.

Pressing ERASE or switching the power off momentarily will clear the entire user memory including programs as well as data storage. However, if the Math block is inserted, pressing TBL 4 will clear only the available R( ) registers, leaving intact any programs residing in the memory. TBL 5 clears the A, B, C, X, Y, and Z registers.

A specified number of R( ) registers starting with R0 can be cleared of data by loading a magnetic card which has the desired number of R( ) registers recorded with zeroes. This will not affect the contents of previously loaded R( ) registers above the highest-numbered one zeroed by the magnetic card.

Fast Circle Plot

by Sy Ramey, of the Hewlett-Packard Santa Rosa Division

This routine for fast plotting of a circle (up to 10 points or more per second) uses the 9820 or 9821, 9862, plus Math and PC I ROM. The user sets equal X and Y limits manually on the plotter. The routine requires pressing SET FLAG to terminate plotting and move the pen away from the plotting area after the circle is completed.

```
0:
SCL -1,1,-1,1;
DSP "ADJ TO SQUA
RE";STP ⊢
1:
ENT "SCALE=?",C;
"RADIUS=?",X/C→X
;0→Y0
2:
COS (5/√X→C)→A;
SIN C→B⊢
3:
GTO +0;PLT X,Y;A
(X→Z)-BY→X;AY+BZ
→Y;IF FLG 0;GTO
+1⊢
4:
LTR 1,1;END ⊢
R405
```

Note that the speed is attained by first plotting a point on the specified radius, then successively rotating the axes using a routine involving only simple multiplication, addition, and subtraction. Mr. Ramey advises that this technique is applicable to linear sine wave plots and function plots such as sin x/x.

Identifying the Last Marked File

by Dr. R. K. Littlewood of the University of Wisconsin, Madison, Wisconsin

I sometimes find it useful to know exactly how many files have been marked on a cassette tape. The following 9820 coding sequence automatically does an Identify File operation on the last marked file on a tape, provided that the tape is not currently positioned beyond that mark.

FDF 999; BKS; IDF A,B,C,X, BKS

Under normal circumstances, B and C will both have the value zero, as A will be a "dummy" file; i.e., the extra file marked in the last Mark Tape operation.

Tape Duplication

by William H. Clayton, College of Marine Sciences, Texas A&M University

Here is a tape duplication program for the 9820 that does not copy empty files, is not wiped out by a binary file, and copies the files exactly. Also, it incorporates adequate explanatory material concerning the use of the program and the copying of binary files.

I have used this program several times and found it efficient and trouble free. It may be that other people have run into the problem of copying tapes and could use this program.

```
0:
GTO 13⊢
1:
FXD 0;ENT "FIRST
 FILE NO.=?",A→C
⊢
2:
DSP "LOAD MASTER
";STP ⊢
3:
FDF A;IDF A,B,X,
Y;BKS ;IF X=0;A+
1→A;GTO +0⊢
4:
GTO 6;IF X=28;
PRT "FILE IS TYP
E 28"GTO 5⊢
5:
PRT "A=",A,"C=",
C, "Y=",Y;STP ⊢
6:
PRT "COPY FILE",
A;IF X=28;GTO 13
;LDF A,8⊢
7:
LDF A⊢
8:
DSP "LOAD COPY";
STP ⊢
9:
IF C=0;REW ;MRK--
1,Y⊢
10:
FDF C;MRK 1,Y;
IF X≠2;GTO 13;
RCF C;GTO 12⊢
11:
RCF C,R(B-1)⊢
12:
C+1→C;A+1→A;GTO
2⊢
13:
PRT "TAPE DUPLIC
ATING","PROGRAM
FOR 9820"⊢
14:
PRT "---DOES NOT
 COPY","EMPTY FI
LES.  IF"⊢
15:
PRT "INITIAL FIL
E NO.","ON COPY
TAPE IS⊢
16:
PRT "NOT ZERO (N
OT ON","CLEAR LE
ADER),","ENTER T
HIS INIT-"⊢
17:
PRT "IAL FILE NO
. IN","REGISTER
C (WHEN"⊢
18:
PRT "THE COPY TA
PE IS","LOADED).
";SPC 2⊢
19:
PRT "THE PROGRAM
 WILL","PRINT OU
T A,C,&Y"⊢
20:
PRT "VALUES AND
THEN","HALT WHEN
 A TYPE"⊢
21:
PRT "28 FILE IS
MET.";SPC 2;PRT
"BEFORE LOADING
A"⊢
22:
PRT "TYPE 28, RE
CORD","THIS DUPL
ICATING"⊢
23:
PRT "PROGRAM BEC
AUSE","ALL OF TH
E USER","RWM WIL
L BE DE-"⊢
24:
PRT "STROYED BY
SUCH","LOADING."
;SPC 2⊢
25:
PRT "TO COPY THE
 TYPE","28, PRES
S LDF A ("⊢
26:
PRT "EXC)--THEN
LOAD","COPY TAPE
--PRESS","FDF C
(EXC)-MRK"⊢
27:
PRT "1,Y(EXC)-IS
P 1,C","(EXC)-IS
P 2(EXC)","--FIL
E IS COPIED";
SPC 2⊢
28:
PRT "RELOAD DUPL
ICAT-","ING PROG
RAM AND","GTO 1
TO RESUME-"⊢
29:
PRT "BE SURE TO
ENTER","CORRECT
NEW VAL-","UES F
OR A AND C.";
SPC 8⊢
30:
GTO 1⊢
31:
END ⊢
R287
```

Speeding Counters

by Howard Rathbun, Hewlett-Packard, Desktop Computer Division

The following program is a straightforward method that takes 20 seconds to execute 1,000 iterations.

```
0:
0→R0⊢
1:
IF (R0+1→R0)≤100
0;JMP 0⊢
2:
END ⊢
Σ11999
R1417
```

The next program does the same thing in 16 seconds. The speed increases because the calculator must calculate where R0 is, whereas, it knows where A is.

```
0:
0→A⊢
1:
IF (A+1→A)≤1000;
JMP 0⊢
2:
END ⊢
Σ12036
R1418
```

The next program is even faster, but the reason is not so obvious. This program takes 10 seconds to do 1,000 iterations. The reason is that the calculator must use the "number-building routine" three times (for 1, 1,000 and 0) for each iteration in the preceding program, but in this program the number-building routine is used only in line 0 before entering the loop. Finding the numbers in registers is faster than creating them.

```
0:
0→A→B;1→C;1000→X
⊢
1:
IF (A+C→A)≤X;
JMP B⊢
2:
END ⊢
Σ15203
R1417
```

Finally, the last program is a bit faster — 9 seconds. This is because the two statements are replaced by one in line 1. This program also illustrates the use of a relational operator to determine whether to JMP 0 or JMP 1.

```
0:
0→A;1→C;1000→X⊢
1:
JMP (A+C→A)>X⊢
2:
END ⊢
Σ13856
R1417
```

Double Unary Minus

by R. M. Holford of Deep River, Ontario, Canada

A double unary minus (--) can sometimes be used to force a change in the normal heirarchy of various operations in a program line, as illustrated in the following sequence to print the Fibonacci number series:

```
        Program                    Output

0:                                    0
FXD 0;PRT 0→X,1→                      1
Y⊢                                    1
1:                                    2
PRT --X+(Y→X)→Y;                      3
JMP 0⊢                                5
2:                                    8
END ⊢                                13
R419                                 21
                                     34
                                     55
                                     89
                                      .
                                      .
                                      .
```

In the above program, the double unary minus in Line 1 takes priority in the operational sequence, and the value in X is stored in a temporary location before the action indicated by the parentheses is taken. Removing the double unary minus results in the output below, since the highest operational priority is then removal of the parentheses; the value in Y is stored in X first, so the original X value is lost.

```
  0
  1
  2
  4
  8
 16
 32
 64
128
256
  .
  .
  .
```

The only drawback is that the last set of X and Y data is printed twice. The following program corrects this but requires the use of RØ and R1. Can someone find a simple way to correct this problem and still only use the alpha registers?

```
0:
ENT X,Y; FLG0(.5
(X-R0)(Y+B)+R1)
R1;X→R0;PRT R→
FLG 13;SFG 0;
GTO 0;IF FLG 13=
0;PRT Y→B; SPC⊢
```

Change Settings During Program Execution

by Steven W. Weeks, Division of Environmental Health, Kansas State Department of Health, Topeka, Kansas

Changes in the fixed/float and flag settings other than restoring the previous condition or setting flag 0 are often desirable while running a program. Such changes may be made when the calculator pauses for an ENT statement. If the response to

ENT "NEXT X?", X

is... FXD 2;SFG 8;25 RUN PROGRAM

the first two actions will be taken before 25 is stored in X. The value to be stored must always come last. For example, if the response to the above ENT statement was

25; FXD 3 RUN PROGRAM

3 would have been stored in X. Also, arithmetic expressions may be executed at such a pause. In the above example, if the response was

10 →B; SIN (π/8) RUN PROGRAM

10 would have been stored in B and the result of the expression in X.

Flag 1 Switch During Program Execution

by Mr. C. T. McCullough, Collins Radio Company, Cedar Rapids, Iowa

This feature is used to toggle FLAG 1 to indicate whether or not to print intermediate data during program execution. Key in the following program and see how it works.

```
0:                      2:
FLG 1≠FLG 0→A;          DSP "FLAG 1 IS A
CFG 1;SFG 1=A;           ZERO";GTO 0⊢
CFG 0⊢                  3:
1:                      END ⊢
IF FLG 1; DSP "FL       R392
AG 1 IS A ONE";
GTO 0⊢
```

Little Things That Count

The 9820 can save you time in solving day-to-day problems. Besides calculating equations in algebraic form and having a very powerful but simple programming language, the 9820 has many time-saving features that can be used in several different ways. Which of the following functions do you use each day?

* EDIT *
* TRACE *
* SET FLAG *

All three of these have obvious capabilities corresponding to their name. But they will do much more.

For example, the edit functions DELETE, INSERT, RECALL, BACK, and FORWARD can be used to alter calculations performed from the keyboard as well as to edit program lines. Once an expression has been executed, it can be recalled by pressing DELETE, BACK or FORWARD. With the equation back in the display editing can proceed as usual.

The TRACE function is a very important tool in debugging programs but it can also be used to print data. Press TRACE prior to entering the data and a record of both input and output will automatically be printed.

SET FLAG is often used in a program to alter the logic flow, but is can also do the same thing from the keyboard. Flag zero is set to 1 by pressing SET FLAG while a program is running. This allows the user to interact with the program, for instance to initiate a print or plot routine whenever he wishes.

One-Line X/Y Integration

by Erik Siwertz of the Institut National de la Recherche Agronomique in Thonon-Les-Bains, France

What do you think about this one-line X/Y integration for the 9820A (either with or without a Math ROM)? The tip works with increasing or decreasing values of X (positive or negative), two flags (0 and 13), and only the alpha register.

```
0:
ENT X,Y;((Y+B)/2
)Γ((X-A)↑2)(FLG
0-FLG 13)+C→C;X→
A;Y→B;SFG 0;GTO
0;IF FLG 13;PRT
C⊢
1:
END ⊢
```

Answering the Challenge of One-Line X/Y Integration

by Carlton E. Thurston, Martin Marietta Cement, Thomaston, Maine

The programming tip entitled "One-Line X/Y Integration" in Vol. 8, No. 2 was quite interesting, and I am unable to resist Mr. Siwertz's challenge. Here is my entry:

```
0:
ENT X,Y;FLG 0(.5
 (X-A)(Y+B)+C)→C;
PRT X→A,Y→B;SPC
;SFG 0;GTO 0;IF
FLG 13;PRT C⊢
```

The basic structure of the program has not been altered, but two operating improvements have been made:

1. Register C is automatically initialized to zero, and
2. Input data is printed out for reference.

Also note that the FLG 13 term is not required in the mathematical expression, since the (X-A) term is always zero when the program is run without a data entry. By omitting the absolute value operation on (X-A), it is possible to make corrections to data after it is entered. This is accomplished by simply reversing the order of data entry until a good entry is re-entered. Then proceed normally.

Lettering Syntax

by Mr. Jan Kuncar of Prague, Czechoslovakia

The syntax of the "letter" statement described in the 9820 PC I Operating Manual,

LTR X, Y, hwd

can be generalized to the following form:

LTR X, Y, E

where E is an arbitrary expression. The sign and decimal point of the value E do not affect the results. The three most significant digits are interpreted as h, w, d, respectively. When h or w is zero, height or width of the character, respectively, is also zero. The value of d is taken modulo 4, i.e., 0, 4, and 8 have the same effect. A missing character is interpreted as zero (e.g., LTR 0, 0, 72 is interpreted as LTR 0, 0, 72.0)

Examples

The following program will plot a series of "A" characters of increasing height:

```
0:
SCL 0,10,0,10;1→
A→Z;FXD 0⊢
1:
LTR A+.8→A,7,10Z
+3.1;DSP Z;DSP ;
DSP ;PLT "A";IF
(Z+1→Z)≤9;JMP 0⊢
2:
DSP "END";END ⊢
```

Each of the following lines has the same effect ($\sqrt{30}$ = 5.4772...):

```
LTR 5,5,543;PLT
"E"⊢
LTR 5,5,547;PLT
"E"⊢
LTR 5,5,5.47;
PLT "E"⊢
LTR 5,5,√→⊢ PLT
"E"⊢
LTR 5,5,-√30;
PLT "E"⊢
```

The following program plots the "D" characters from the same point in all directions:

```
0:
SCL 0,10,0→Z,10⊢
1:
LTR 3,3,870+Z;
DSP Z;DSP ;PLT "
D";IF (Z+1→Z)≤9;
JMP 0⊢
2:
DSP "END";END ⊢
```

Extending Definable Functions

by Mr. D.F. Ashcroft, Senior Mining Engineer, Cobar Mines PTY. LTD., Cobar, N.S.W., Australia

When using a 9820 with three plug-in ROMs (e.g., UDF, Math, PC I) only five keys remain available for user-defined functions or subprograms. This limitation can be overcome by combining several functions on a single key by using the UDF parameter "P1" to define a jump to the particular routine. For example the code would be organized as follows:

```
0: "SUB"; JMP P1⊢
1: Routine one
2: Routine two
        .
        .
        .
```

A particular routine can be called by:

CLL SUB 5

where 5 is the line number of the routine being called.

BASIC Integer To Algebraic

by D. L. Schacher of Tel-Instrument Electronics Corp., Carlstadt, New Jersey

Recently a problem was encountered in translating a BASIC program into algebraic for the 9820A. BASIC says that the function INT "gives the largest integer ≤ the expression", while the MATH ROM INT "eliminates fractional part of value; does not affect sign or integer value." For positive numbers, there is no difference, but for negative numbers, INT (BASIC)(−1.5) = −2, while INT (Algebraic)(−1.5) = −1. Thus, when rewriting a BASIC program for the 9820A or 9821A where negative values may occur, instead of INT(X), write INT[X−(0>X)], to maintain the same meaning.

Faster Integer Powers

by D. L. Schacher, Tel-Instrument Electronics Corporation, Carlstadt, New Jersey

A surprising amount of program execution time can be saved by efficient coding. Functions such as X↑Y, which is calculated by the 9820 Math ROM as $Z = e^{(Y \ln X)}$ where both e^A and lnA are calculated by an iterative method, are time consuming. Consequently, when a function is squared or taken to an integer power of a reasonable size, it improves execution time to use straight multiplication rather than the power function. For example the function:

(A + B − 2c/x) ↑ 2 → Y

should be coded as

(A + B − 2c/x) → Z;ZZ → Y.

High Speed File Identification

by Koichi Tanaka of Yokogawa-Hewlett-Packard, Ltd., Tokyo, Japan

IDENTIFY FILE (IDF) is often used to determine the construction of the cassette files. Although Program A (below) will accomplish this, it works very slowly when the files are large.

My program, B, works rapidly because of using high speed search capability. It takes about half as much time to identify files using B as using A.

Program A

```
0:
FXD 0;⁜FDF 0⊢
1:
⁜IDF A,B,C,X⊢
2:
PRT A,C,B,X;SPC
⊢
3:
GTO 1⊢
4:
END ⊢
Σ23621
R1442
```

Program B

```
0:
FXD 0;⁜FDF 0⊢
1:
⁜IDF A,B,C,X⊢
2:
IF X>5;⁜BKS ;⁜
FDF A+2;PRT A,C,
B,X;SPC ;⁜BKS ;
GTO 1⊢
3:
PRT A,C,B,X;SPC
;GTO 1⊢
4:
END ⊢
Σ20131
R1437
```

Example

```
 17 ◄──────── File no.
  2 ◄──────── File type
 85 ◄──────── Current size
100 ◄──────── Absolute size

 18
  0
  0
100

 19
 20
 99
109
```

"DO" Loops

by D. L. Schacher of Tel-Instrument Electronics Corp., Carlstadt, New Jersey

Figure 1 below shows the normal manner of programming two nested loops on a 9820 or 9821 which, while efficient, does not always indicate the broad picture of what is being accomplished. In FORTRAN, for example, the "DO" loop indicates what is to be done, without getting involved in the question of how to do it.

It is possible to write "DO" loops on a 9820 or 9821, as shown in Figures 2 and 3, using a "DO" key on the UDF ROM. This "DO" key is somewhat better than the FORTRAN "DO", as it can be nested up to 11 deep, and can have positive or negative initial, final, and incremental values.

The "DO" key (Figure 2) uses five passing parameters: P1 is the DO loop number (from 1 to 11), to allow keeping track of which loop ends where; P2 is the variable, P3 is the initial value, P4 is the final value, and P5 is the increment or decrement. The line before each CLL DO must have CFG N: SFG 12, where N is the loop number; while the end of the loop is signified by IF FLG N = 0; GTO (CLL DO line). Figure 3 gives a sample program using the "DO" key, while Figure 4 is the resulting printout.

```
0:
4→A⊢
1:
FXD 2;PRT 3↑A→Z⊢
2:
1→B⊢
3:
FXD 0;PRT 2↑B→Z⊢
4:
B+1→B;IF B≤3;
GTO -1⊢
5:
A-1→A;IF 1≤A;
GTO -4⊢
6:
"END";SPC 8;END
⊢
```

Fig. 1 Conventional loop nesting

```
0:
"DO ";P2+P5→P2;
IF FLG 12;P3→P2;
CFG 12⊢
1:
IF (P5/ABS P5)(P
2+P5-P4)≤0;GTO +
2⊢
2:
SFG P1⊢
3:
END 0
```

Fig. 2 Defined "DO" key

```
0:
CFG 1;SFG 12⊢
1:
CLL DO 1,A,4,1,-
1⊢
2:
FXD 2;PRT 3↑A→Z⊢
3:
CFG 2;SFG 12⊢
4:
CLL DO 2,B,1,3,1
⊢
5:
FXD 0;PRT 2↑B→Z⊢
6:
IF FLG 2=0;GTO -
2⊢
7:
IF FLG 1=0;GTO -
6⊢
8:
SPC 8;END 0
```

Fig. 3 Mainline program for "DO" loops

```
81.00
    2
    4
    8
27.00
    2
    4
    8
 9.00
    2
    4
    8
 3.00
    2
    4
    8
```

Fig. 4 Output of "DO" routine

'No-Operation' Editing Aid

In modifying a 9820 program containing line-dependent branching addresses, such as GTO + 3 or GTO 17, you may wish to include some no-operation lines. These lines can replace active lines which are to be deleted, thus avoiding changes in branching addresses.

To replace an active line by a no-operation one, just address the desired line, such as GTO 2, then press CLR STORE. The resulting line,

2: ⊢

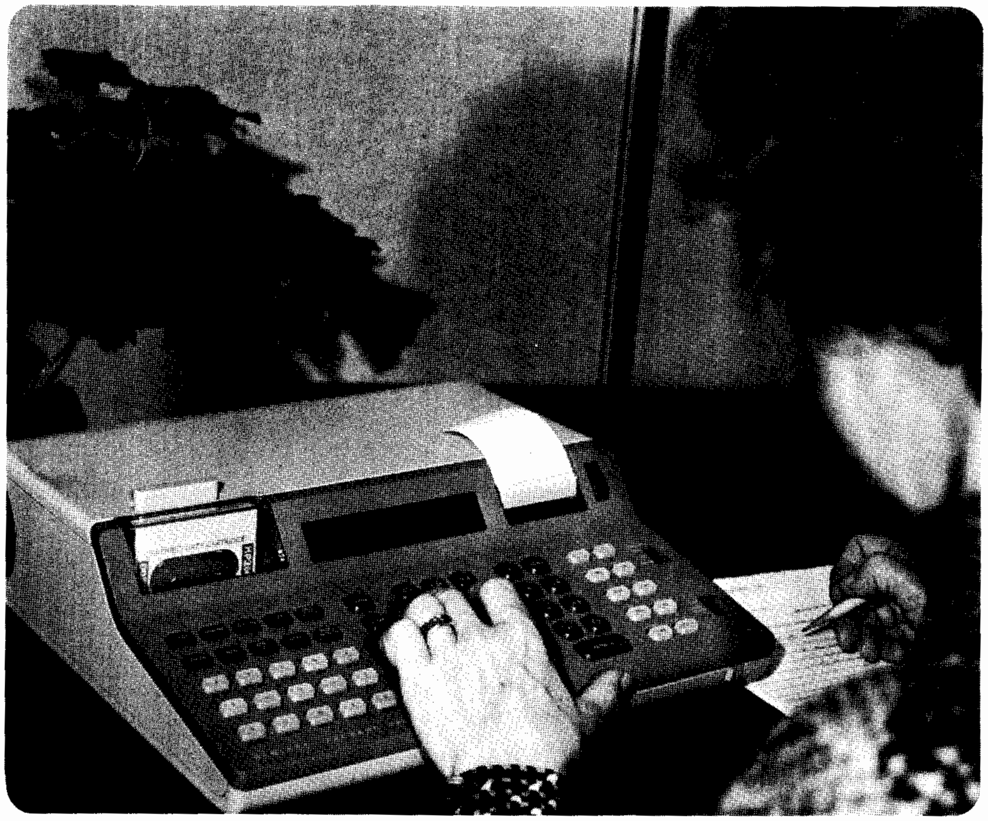
is a no-operation line which takes minimum memory, replacing the previous line 2.

At the STORE command for this type of line, the line counter does not advance; you must manually address it to the next line. In this case, press GTO 3, followed by CLR STORE if line 3 is to be a no-operation line, or GTO 3 EXECUTE if line 3 is to contain active instructions. During program operation, the line counter automatically steps past the no-operation lines until the next active line is reached.

No-operation lines are most useful in editing existing programs to minimize the number of edits. Using label addresses in writing a new program gives the maximum editing flexibility, since this makes the program independent of line insertion or deletion.

Section 5

9815



Index

Section 5 - 9815

Duplicating Tape Cartridges

Originally submitted by F. William Schueler, Rollway Bearing Company, Syracuse, New York 13201, U.S.A., modified by Desktop Computer Division

Here is a program to duplicate the contents of a tape cartridge either for use at another location or as a safety measure to insure against accidental loss of programs due to tape damage.

The program runs on the 9815A Opt. 001 (2008 steps) and has the following limitation:

1. Only cartridges containing file types 0, 2, 5 or 6 can be duplicated and
2. Files containing more that 1812 program steps or 227 data registers will not be duplicated. A 2000-step empty file will be marked and this will be noted on the printout. An empty file will be marked as such and also noted on the printout.

The operation of the program is as follows: After entry "END" and "RUN" are keyed. The printout calls for "MIN.FILE#". This is the algebraically lowest file number, i.e.: −3<−0<0<1. This number is entered and "RUN" is keyed. The printout calls for "MASTER". The cartridge to be duplicated is placed in the tape drive and "RUN" keyed. The printout calls for "COPY". The cartridge on which the duplicate recording is to be made is placed in the tape drive and "RUN" is keyed. Continue alternating the two cartridges until duplication is complete and "END" is printed out.

Steps 120 through 125 govern the size of file in which each program is recorded. In order to allow for changes that may be made in the program, the file will be marked at least 150 steps longer than the program.

The 1812-step limitation on the length of the program to be duplicated is obviously caused by the necessity of retaining the duplicating program in memory. In order to handle as many steps as possible, I have tried to reduce the ALPHA to a minimum.

When copying an entire tape, the min and max file numbers should not include the extra files, since they are implicitly copied.

If a different 'cushion' is desired between the program length and the file length, merely change the constant in lines 120-122 and, if the change is an order of magnitude, adjust the constant in line 124 accordingly; i.e., if lines 120-122 were changed to 10, then line 124 would be changed to 1. If no 'cushion' is desired, delete lines 120 through 125.

```
0000 0
0001 #REGS
0002 2
0003 EEX
0004 3
0005 STO     J
0006 1
0007 9
0008 6
0009 STO     I
0010 PRNTα
0012 M
0013 I
0014 N
0015
0016 F
0017 I
0018 L
0019 E
0020
0021 #
0022 ENDα
0023 STOP
0024 PRINT
0025 STO     B
0026 IF -
0027 SFG     1
0028 PRNTα
0030 M
0031 A
0032 X
0033
0034 F
0035 I
0036 L
0037 E
0038
0039 #
0040 ENDα
0041 STOP
0042 PRINT
0043 STO     C
0044 IF -
0045 SFG     2
0046 IF CFG 1
0047 GOTO    0160
0049 0
0050 IF SFG 2
0051 RCL     C
0052 IF SFG 2
0053 +⇌-
0054 STO     A
0055 RCL     B
0056 +⇌-
0057 STO     F
0058 FOR     A→F
0059 PRNTα
0061 M
0062 A
0063 S
0064 T
0065 E
0066 R
0067 ENDα
0068 STOP
0069 GOSUB   0185
0071 IDENT
0072 ROLL↓
0073 STO     D
0074 ROLL↓
0075 STO     E
0076 1
0077 8
0078 1
0079 2
0080 IF X<Y
0081 SFG     3
0082 ROLL↓
0083 ROLL↓
0084 2
0085 IF X=Y
0086 SFG     4
0087 CLX
0088 5
0089 IF X=Y
0090 SFG     5
0091 IF CFG 4
0092 GOTO    0099
0094 RCL     E
0095 8
0096 ÷
0097 #REGS
0098 STO     G
0099 RCL     I
0100 IF SFG 4
0101 0
0102 GOSUB   0185
0104 IF SFG 5
0105 GOTO    0109
0107 IF CFG 3
0108 LOAD
0109 PRNTα
0111 C
0112 O
0113 P
0114 Y
0115
0116
0117 ENDα
0118 STOP
0119 RCL     E
0120 2
0121 0
0122 0
0123 +
0124 2
0125 ROUND
0126 IF SFG 3
0127 RCL     J
0128 IF SFG 5
0129 RCL     D
0130 1
0131 GOSUB   0185
0133 MARK
0134 IF SFG 3
0135 SFG     5
0136 IF SFG 5
0137 GOTO    0169
0139 RCL     G
0140 IF SFG 4
0141 0
0142 IF CFG 4
0143 RCL     I
0144 GOSUB   0185
0146 IF SFG 4
0147 RCDATA
0148 IF CFG 4
0149 RCPRGM
0150 CFG     3
0151 CFG     4
0152 CFG     5
0153 NEXT    A
0154 IF SFG 2
0155 GOTO    0189
0157 IF CFG 1
0158 GOTO    0189
0160 RCL     B
0161 IF SFG 1
0162 0
0163 STO     A
0164 RCL     C
0165 STO     F
0166 CFG     1
0167 GOTO    0058
0169 GOSUB   0185
0171 PRNTα
0173 #
0174
0175 PRINT
0176
0177 E
0178 M
0179 P
0180 T
0181 Y
0182 ENDα
0183 GOTO    0150
0185 RCL     A
0186 IF SFG 1
0187 +⇌-
0188 RETURN
0189 PRNTα
0191 E
0192 N
0913 D
0194 ENDα
0195 END
```

9815A Data Entry

by Chris Jennings, Graylingwell Hospital, Chichester, Sussex, United Kingdom

Entering and storing strings of single digit numbers on the 9815A can be a tedious process because of the need to press Run/stop after each digit. The routine below enables the user to key in up to 10 digits at a time, pressing Run/stop once only at the end of the string, thus saving time. The program splits the 10 digit number into 10 single digit numbers and stores each one in a separate register.

```
0000 0
0001 #REGS
0002 1
0003 1
0004 #REGS
0005 CLRA→J
0006 1
0007 STO     E
0008 1
0009 STO     I
0010 1
0011 0
0012 STO     B
0013 STOP
0014 STO     D
0015 RCL     B
0016 RCL     I
0017 -
0018 STO     J
0019 0
0020 STO     H
0021 1
0022 STO     A
0023 RCL     B
0024 RCL     I
0025 ÷
0026 STO     F
0027 FOR     A→F
0028 RCL     D
0029 1
0030 0
0031 RCL     J
0032 Y↑X
0033 ÷
0034 INT
0035 STO     C
0036 RCL     H
0037 1
0038 0
0039 RCK     I
0040 Y↑X
0041 *
0042 -
0043 STO I   E
0045 PRINT
0046 RCL     C
0047 STO     H
0048 RCL     I
0049 STO-    J
0050 1
0051 STO+    E
0052 NEXT    A
0053 END
```

The number of digits entered together can be set to any number from 2 to 10 by changing the value of B. The value of E indicates the register into which the single digit number will be placed. The program can also be used to store multi-digit numbers. By changing the value of I, a string of digits can be split, instead, into 2-, 3-, 4- or 5-digit numbers.

This routine can be incorporated into larger programs and by use of a further loop, longer strings of digits can be entered in groups of up to 10 (e.g. 50 digits in 5 groups of 10).

Biocurve and bionumbers on the 9815

by Edgar Albert, Kronenstrasse 15, 7809 Denzlingen, West Germany

For those readers who believe biorhythm information can be helpful, here are two programs you may want to try. Both programs produce information related to biorhythms.

The first program takes your date of birth, as well as a second date of your choice, and calculates your values for three factors on that second date. These factors are:

man rhythm = M
woman rhythm = W
intelligence rhythm =J

The second program prints a + or − in each of the above categories for each day of a month you specify.

```
0001 SPACE
0002 PRNTα
0004 B
0005 I
0006 O
0007 N
0008 U
0009 M
0010 B
0011 E
0012 R
0013 S
0014 ENDα
0015 FIX     0
0017 1
0018 3
0019 #REGS
0020 2
0021 8
0022 STO     R002
0024 3
0025 0
0026 STO     R004
0028 STO     R006
0030 STO     R009
0032 STO     R011
0034 3
0035 1
0036 STO     R001
0038 STO     R003
0040 STO     R005
0042 STO     R007
0044 STO     R008
0046 STO     R010
0048 STO     R012
0050 CLEAR
0051 SPACE
0052 GOSUB   0260
0054 STO     B
0055 GOSUB   0273
0057 STO     C
0058 GOSUB   0286
0060 STO     D
0061 IF SFG 1
0062 GOTO    0073
0064 SPACE
0065 GOSUB   0260
0067 STO     I
0068 GOSUB    0273
0070 STO     G
0071 GOSUB   0286
0073 STO     H
0074 SPACE
0075 CLEAR
0076 STO     J
0077 RCL     B
0078 STO-    J
0079 RCL     I
0080 STO+    J
0081 RCL     C
0082 STO     A
0083 RCL I   A
0085 STO+    J
0086 1
0087 2
0088 RCL     A
0089 IF X<Y
0090 GOTO    0097
0092 CLEAR
0093 STO     A
0094 3
0095 6
0096 5
0097 STO-    J
0098 1
0099 STO+    A
0100 RCL     A
0101 RCL     G
0102 IF X=Y
0103 GOTO    0106
0105 GOTO    0082
0107 RCL     D
0108 STO     A
0109 RCL     A
0110 4
0111 ÷
0112 ENTER↑
0113 INT
0114 IF X<Y
0115 GOTO    0118
0117 1
0118 STO+    J
0119 RCL     A
0120 RCL     H
0121 IF X=Y
0122 GOTO    0131
0124 3
0125 6
0126 5
0127 STO+    J
0128 1
0129 STO+    A
0130 GOTO    0108
0132 RCL     B
0133 STO     R000
0135 RCL     C
0136 GOSUB   0302
0138 RCL     D
0139 GOSUB   0299
0141 RCL     R000
0143 FIX     4
0145 PRNTα
0147 D
0148 A
0149 T
0150 E
0151
0152 1
0153 :
0154 PRINT
0155 ENDα
0156 RCL     I
0157 STO     R000
0159 RCL     G
0160 GOSUB   0302
0162 RCL     H
0163 GOSUB   0299
0165 RCL     R000
0167 PRNTα
0169 D
0170 A
0171 T
0172 E
0173
0174 2
0175 :
0176 PRINT
0177 ENDα
0178 FIX     0
0180 RCL     J
0181 PRNTα
0183 D
0184 A
0185 Y
0186 S
0187
0188 =
0189 PRINT
0190 ENDα
0191 2
0192 3
0193 GOSUB   0308
0195 PRNTα
0197 M
0198
0199 =
0200 PRINT
0201 ENDα
0202 GOSUB   0321
0204 2
0205 8
0206 GOSUB   0308
```

```
0208 PRNTα
0210 W
0211
0212 =
0213 PRINT
0214 ENDα
0215 GOSUB  0321
0217 3
0218 3
0219 GOSUB  0308
0221 PRNTα
0223 J
0224
0225 =
0226 PRINT
0027 ENDα
0028 GOSUB  0321
0230 SPACE
0231 PRNTα
0233 C
0234 H
0235 A
0236 N
0237 G
0238 E
0239
0240 D
0241 A
0242 .
0243 1
0244
0245 O
0246 R
0247
0248 2
0249 ENDα
0250 SPACE
0251 2
0252 STOP
0253 IF X=Y
0254 GOTO   0063
0256 1
0257 IF X=Y
0258 SFG     1
0259 GOTO   0049
0261 PRNTα
0263 D
0264 A
0265 Y
0266
0267
0268
0269 ?
0270 ENDα
0271 STOP
0272 PRINT
0273 RETURN
0274 PRNTα
0276 M
0277 O
0278 N
0279 T
0280 H
0281
0282 ?
0283 ENDα
0284 STOP
0285 PRINT
0286 RETURN
0287 PRNTα
0289 Y
0290 E
0291 A
0292 R
0293
0294
0295 ?
0296 ENDα
0297 STOP
0298 PRINT
0299 RETURN
```

```
0300 EEX
0301 2
0302 ÷
0303 EEX
0304 2
0305 ÷
0306 STO+   R000
0308 RETURN
0309 SPACE
0310 STO     E
0311 RCL     J
0312 RCL     E
0313 ÷
0314 ENTER↑
0315 INT
0316 -
0317 RCL     E
0318 *
0319 IF 0
0320 RCL     E
0321 RETURN
0322 2
0323 *
0324 RCL     E
0325 LSTX
0326 -
0327 EEX
0328 2
0329 *
0330 RCL     E
0331 ÷
0332 PRNTα
0334 %
0335
0336 =
0337 PRINT
0338 ENDα
0339 ROLL↓
0340 RETURN
0341 END
```

```
BIONUMBERS

GOSUB OVERFLOW

BIONUMBERS

DAY   ?
                24
MONTH ?
                 6
YEAR  ?
                45

DAY   ?
                17
MONTH ?
                12
YEAR  ?
                79

DATE 1:  24.0645
DATE 2:  17.1279
DAYS =     12594

M =           13
% =          -13

W =           22
% =          -57

J =           21
% =          -27

CHANGE DA.1 OR 2
```

```
0000 PRNTα
0002 B
0003 I
0004 O
0005 C
0006 U
0007 R
0008 V
0009 E
0010 ENDα
0011 GOTO   0109
0013 LBL
----- 00
0015 RCL     A
0016 PRNTα
0018 PRINT
0019
0020 -
0021 -
0022 -
0023 ENDα
0024 RETURN
0025 LBL
----- 01
0027 RCL     A
0028 PRNTα
0030 PRINT
0031
0032 -
0033 -
0034 +
0035 ENDα
0036 RETURN
0037 LBL
----- 02
0039 RCL     A
0040 PRNTα
0042 PRINT
0043
0044 -
0045 +
0046 -
0047 ENDα
0048 RETURN
0049 LBL
----- 03
0051 RCL     A
0052 PRNTα
0054 PRINT
0055
0056 -
0057 +
0058 +
0059 ENDα
0060 RETURN
0061 LBL
----- 04
0063 RCL     A
0064 PRNTα
0066 PRINT
0067
0068 +
0069 -
0070 -
0071 ENDα
0072 RETURN
0073 LBL
----- 05
0075 RCL     A
0076 PRNTα
0078 PRINT
0079
0080 +
0081 -
0082 +
0083 ENDα
0084 RETURN
0085 LBL
----- 06
0087 RCL     A
0088 PRNTα
```

```
0090 PRINT
0091
0092 +
0093 +
0094 -
0095 ENDα
0096 RETURN
0097 LBL
----- 07
0099 RCL     A
0100 PRNTα
0102 PRINT
0103
0104 +
0105 +
0106 +
0107 ENDα
0108 RETURN
0109 1
0110 3
0111 #REGS
0112 2
0113 8
0114 STO     R002
0116 3
0117 0
0118 STO     R004
0120 STO     R006
0122 STO     R009
0124 STO     R011
0126 3
0127 1
0128 STO     R001
0130 STO     R003
0132 STO     R005
0134 STO     R007
0136 STO     R008
0138 STO     R010
0140 STO     R012
0142 SPACE
0143 FIX     0
0145 PRNTα
0147 D
0148 A
0149 Y
0150 ENDα
0151 STOP
0152 STO     B
0153 PRINT
0154 EEX
0155 2
0156 *
0157 STO     I
0158 GOSUB   0326
0160 STO     C
0161 GOSUB   0335
0163 STO     D
0164 RCL     I
0165 STO     E
0166 IF SFG  1
0167 GOTO    0180
0169 CLEAR
0170 STO     1
0171 GOSUB   0326
0173 STO     G
0174 GOSUB   0335
0176 STO     H
0177 RCL     I
0178 STO     R000
0180 SPACE
0181 FIX     2
0183 RCL     E
0184 PRNTα
0186 D
0187 /
0188 B
0189 PRINT
0190
0191
0192
```

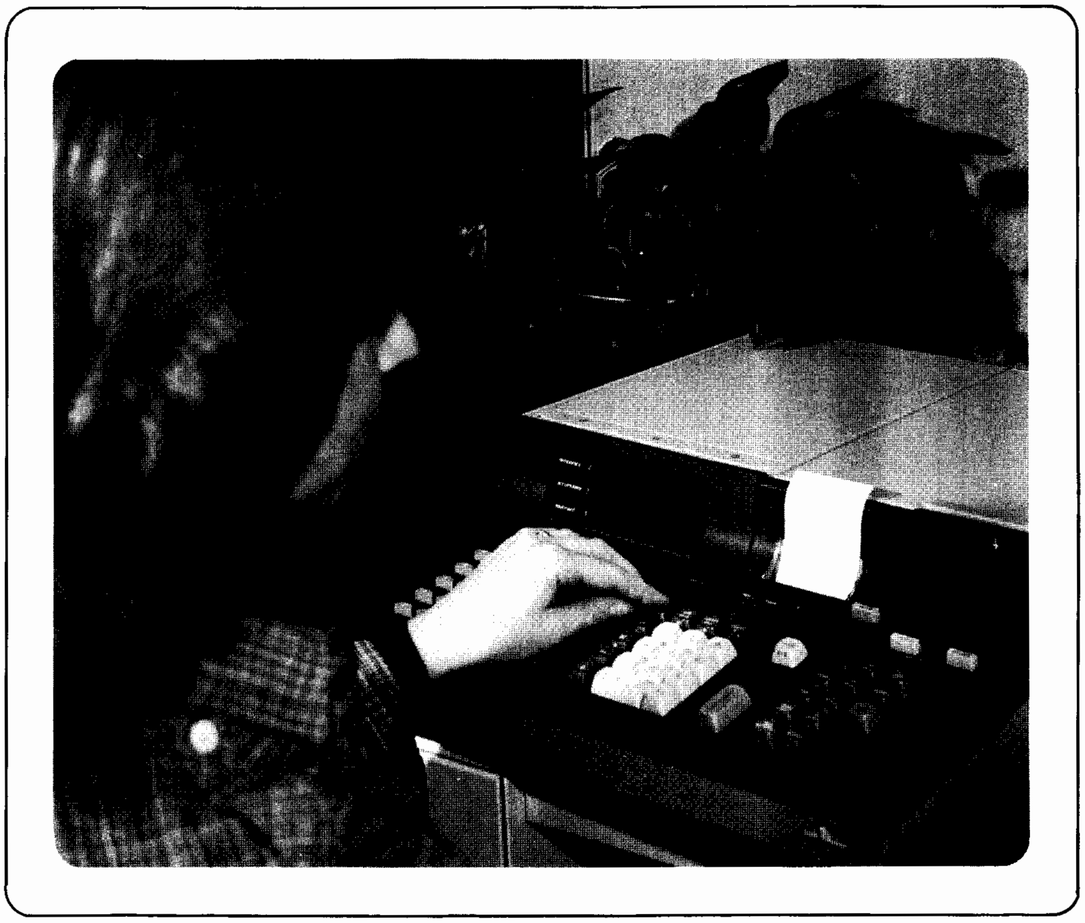
```
0193
0194 ENDα
0195 RCL    R000
0197 PRNTα
0199 M
0200 T
0201 H
0202 PRINT
0203
0204 M
0205 W
0206 I
0207 ENDα
0208 FIX    0
0210 1
0211 STO    J
0212 RCL    B
0213 STO-   J
0214 RCL    C
0215 STO    A
0216 RCL I  A
0218 STO+   J
0219 1
0220 2
0221 RCL    A
0222 IF X<Y
0223 GOTO   0231
0225 CLEAR
0226 STO    A
0227 3
0228 6
0229 5
0230 STO-   J
0231 1
0232 STO+   A
0233 RCL    A
0234 RCL    G
0235 IF X=Y
0236 GOTO   0240
0238 GOTO   0216
0240 RCL    D
0241 STO    A
0242 RCL    A
0243 4
0244 ÷
0245 ENTER↑
0246 INT
0247 IF X<Y
0248 GOTO   0252
0250 1
0251 STO+   J
0252 RCL    A
0253 RCL    H
0254 IF X=Y
0255 GOTO   0265
0257 3
0258 6
0259 5
0260 STO+   J
0261 1
0262 STO+   A
0263 GOTO   0242
0265 1
0266 STO    A
0267 RCL I  G
0269 STO    F
0270 FOR    A→F
0271 0
0272 STO    I
0273 4
0274 ENTER↑
0275 2
0276 3
0277 GOSUB  0351
0279 2
0280 ENTER↑
0281 2
0282 8
0283 GOSUB  0351
0285 1
```

```
0286 ENTER↑
0287 3
0288 3
0289 GOSUB  0351
0291 RCL    I
0292 GOSUB  LX
0293 1
0294 STO+   J
0295 NEXT   A
0296 SPACE
0297 PRNTα
0299 C
0300 H
0301 A
0302 N
0303 G
0304 E
0305
0306 D
0307 A
0308 .
0309 1
0310
0311 O
0312 R
0313
0314 2
0315 ENDα
0316 2
0317 STOP
0318 IF X=Y
0319 GOTO   0169
0321 1
0322 IF X=Y
0323 SFG    1
0324 GOTO   0145
0326 PRNTα
0328 M
0329 T
0330 H
0331 ENDα
0332 1
0333 GOTO   0345
0335 PRNTα
0337 Y
0338 E
0339 A
0340 R
0341 ENDα
0342 EEX
0343 2
0344 +⇌-
0345 STOP
0346 *
0347 STO+   I
0348 LSTX
0349 PRINT
0350 RETURN
0351 RCL    J
0352 X⇌Y
0353 ÷
0354 ENTER↑
0355 INT
0356 -
0357 .
0358 4
0359 7
0360 IF X<Y
0361 CLEAR
0362 1
0363 ROLL↑
0364 *
0365 STO+   I
0366 RETURN
0367 END
```

```
BIOCURVE

DAY
              24
MTH
               6
YEAR
              45
MTH
              12
YEAR
              79

D/B  2406.45
MTH    12.79 MWI
           1 -++
           2 -++
           3 -++
           4 +++
           5 +++
           6 +++
           7 +++
           8 +++
           9 +-+
          10 +-+
          11 +-+
          12 +--
          13 +--
          14 +--
          15 ---
          16 ---
          17 ---
          18 ---
          19 ---
          20 ---
          21 ---
          22 ---
          23 -+-
          24 -+-
          25 -+-
          26 -+-
          27 ++-
          28 ++-
          29 +++
          30 +++
          31 +++

CHANGE DA.1 OR 2
```

Section 6

9810



Index

Section 6 - 9810

Printer-Alpha Test

The Model 9810 Calculator may be purchased with the column printer with or without the Model 11211A Printer Alpha ROM which gives alpha printing capability. The ROM can be purchased separately and plugged in later. Programs can be written to include alpha statements but they are capable of operating either with or without this ROM. This requires a test for the presence of the ROM. The following program sequence will always operate correctly.

	With Alpha			Without Alpha		
	x	y	z	x	y	z
0001--CLR	0	0	0	0	0	0
0002-- 1	1	0	0	1	0	0
0003--FMT	1	0	0	1	0	0
0004--FMT	1	0	0	1	0	0
0005--CLR	1	0	0	0	0	0
0006--FMT	1	0	0	0	0	0
0007--X=Y	1	0	0	0	0	0
0008--GTO						
0009-- 0	} Any Address					
0010-- 2						
0011-- 6						
0012						

If the Alpha ROM is in the system, the equality test at step 0007 is not met, so the program skips the next four instructions and continues. Without the Alpha ROM, the test is met and the program branches to the designated address.

Note that the GTO statement *must* follow the X = Y statement in this case only; in the general case branching is automatic, as it is in the 9100A/B. Also note that in the 9810, the branching address normally takes four steps, and a not-met condition causes the next four steps to be skipped.

From the above point, the program usually follows one of three routes.

1. If there are no other alpha sequences in the program it might continue as follows:

```
0012--CLR          0023-- T
0013--FMT          0024-- A
0014--FMT          0025--FMT
0015-- E           0026--STP
0016-- N           0027--PNT
0017-- T           0028--PNT
0018-- E           0029--XTO
0019-- R           0030-- 5
0020--CNT          0031-- .
0021-- D           0032-- .
0022-- A           0033-- .
```

2. If additional alpha sequences are used in the program, further branching can be directed by activating SET FLAG at the end of each alpha section except the final one. For example:

```
0012--CLR          0098-- 5
0013--FMT          0099--PNT
0014--FMT          0100--PNT
0015-- E              .
0016-- N              .
0017-- T              .
0018-- E           0120--LBL
0019-- R           0121-- A
0020--CNT          0122--FMT
0021-- P           0123--FMT
0022-- T           0124-- 0
0023-- S           0125-- U
0024--FMT          0126-- T
0025--SFL          0127-- P
0026--STP          0128-- U
0027--PNT          0129-- T
0028--PNT          0130--CNT
    .              0131-- P
    .              0132-- T
    .              0133-- S
    .              0134--FMT
0092--IFG          0135--SFL
0093--GTO          0136--S/R
0094--S/R             .
0095--LBL             .
0096-- A              .
0097--XFR          0250--END
```

3. If additional alpha sequences are used in the program but the SET FLAG is not available, the 1 or 0 left in the x register at step 0005 above can be stored, then recalled for a test prior to each subsequent alpha sequence. For example:

```
0012--XTO          0060--X<Y
0013-- 0           0061-- 0
0014--CLR          0062-- 0
0015--FMT          0063-- 7
0016--FMT          0064-- 8
0017-- D           0065--STP
0018-- A           0066--PNT
0019-- T           0067--XTO
0020-- A           0068-- 7
0021--FMT             .
    .                 .
    .              0075--GTO
    .              0076-- 1
    .              0077-- 0
0026--STP          0078-- 6
0027--PNT          0079--FMT
0028--PNT          0080--FMT
0029--XTO          0081-- D
0030-- 6           0082-- A
    .              0083-- T
    .              0084-- A
    .              0085--FMT
0056--XFR             .
0057-- 0              .
0058-- UP             .
0059--CLX          0106--END
```

Clearing Data Registers

by John A. Beaujean, Continental Can Company, Augusta, Georgia

The step sequences shown below will clear the 9810 data registers for the basic machine or with Option 001. Actually this would be the first section of a larger program which requires cleared registers before starting entries, summations, etc. The remainder of the program would start at Step 0015 (0016 for the Option 001 illustration), but a digit could not be used there. The undesirable termination of program execution by a status condition is avoided.

```
  THIS PROGRAM
   CLEARS ALL
   REGISTERS
   INCLUDING
A, B, X, Y, + Z.


0000---CLR----20
0001--- 4 ----04
0002--- 8 ----10
0003---XTO----23
0004--- + ----33
0005--- a ----13
0006---YTO----40
0007---IND----31
0008--- a ----13
0009--- a ----13
0010---x>Y----53
0011---CHS----32
0012--- 1 ----01
0013---GTO----44
0014--- 3 ----03
0015---STP----41
```

```
  THIS PROGRAM
   CLEARS ALL
   REGISTERS
   INCLUDING
A, B, X, Y, + Z.

  (OPTION 001)

0000---CLR----20
0001--- 1 ----01
0002--- 0 ----00
0003--- 8 ----10
0004---XTO----23
0005--- + ----33
0006--- a ----13
0007---YTO----40
0008---IND----31
0009--- a ----13
0010--- a ----13
0011---x>Y----53
0012---CHS----32
0013--- 1 ----01
0014---GTO----44
0015--- 4 ----04
0016---STP----41
```

Economical "If Y = 0" Test

by Professor L. Glasser, Chemistry Department, Rhodes University, Grahamstown, South Africa

The test "if y = 0" may be economically applied on the 9810 Calculator by adding the contents of x- and y-registers into the y-register, and testing the resulting x- and y-register contents for equality.

Thus, with y containing the quality to be tested, and any quantity to be operated on in x, include in the program:

```
--- + ----33
---X=Y----50
--- X ----36
--- X ----36
--- X ----36
--- X ----36
```

If y = 0, then the equality will be satisfied, otherwise not.

The same technique will suffice to provide "if y > 0" and "if y < 0", tests with the "x = y" key being substituted by the "x < y" and "x > y" keys, respectively. These operate correctly whatever the x-register contents, except where the inequality between x and y is large enough so that one of the numbers is lost by rounding. The test will generally work for magnitude differences up to 10^9.

Sequential 'If' Conditions

by C.D. Goode, University of Manchester, Manchester, England

Two sequential 'IF' conditions can be used in a 9810 program to give an instruction to jump only if both conditions are true. As an example, the sequence

```
    IF X = Y
    IF FLAG
     GO TO
     LABEL
       4
   CONTINUE
```

causes a jump to LABEL 4 only if both conditions are true. The CONTINUE acts as a no-operation step if the first condition is not true.

This type of instruction sequence is also useful in testing whether the value (X) lies between (Y) and (Z):

```
    IF X > Y
     ROLL ↑
    IF X > Y
     GO TO
     LABEL
     ROLL ↑
   CONTINUE.
```

If (Z) > (X) > (Y), the program branches to 'LABEL ROLL ↑.' It is convenient to use 'ROLL ↑' as the label because whichever condition is not met the registers contain the same result Z = y, Y = x, X = z.

Terminating Data Entry

by Mr. Oliver H. McKagen, III, of Joseph C. Draper & Associates, Blacksburg, Virginia

Many programs call for entering a series of data values into a summation or repetitive routine as a first or intermediate step in computing a final answer. Very often the series is terminated by a SET FLAG by the operator. This often results in entering an incorrect or zero value if he forgets to set the flag before pressing CONTINUE. A possible solution to this problem is to test the entered value for zero and when this condition is met have the program branch to the appropriate routine. Thus the task of the operator is simplified to entering a zero and pressing CONTINUE once all the data has been entered.

Extending Definable Function Key To Any Number of Functions

by Professor A. S. Gladwin, McMaster University, Hamilton, Ontario, Canada

This tip points out the fact that the 9810 Math Block Definable Function key can be programmed to call any number of user-defined functions. The general concept is to store various functions as subroutines and then by a stored code call them from the f( ) program. In other words, given the function $f_n(y)$, y is a number stored in the y-register and n is a number identifying the function stored in say the x-register. The flow of the program would be to check the code in the x-register for the designated function and then make the computation on the value in the y-register. A sample program might be coded as follows:

```
   LBL
    F
   RUP
    1       }
   X=Y      }
   CNT      }  Check for f1(y)
   GTO      }
   LBL      }
    A       }
    2       }
   X=Y      }
   CNT      }  Check for f2(y)
   GTO      }
   LBL      }
    B       }
    3
   X=Y
    .
    .
   LBL       }
    A        }
    DN       }  Subroutine to
[Calculate]  }  calculate f1(y)
[  f1(y)  ]  }
   S/R
   LBL       }
    B        }
    DN       }  Subroutine to
[Calculate]  }  calculate f2(y)
[  f2(y)  ]  }
   S/R
    .
    .
    .
```

Data Printout

by W. J. Butterworth of the Admiralty, Underwater Weapons Establishment, Portland, Dorset, England

This subroutine prints out the contents of the 9810's data storage registers. A STOP instruction is included so that the user may enter the number of registers required. The total number of registers is limited to 108 for the 9810 with Option 001 (111 registers) or 48 for the basic machine. The labels may, of course, be changed to suit the user.

Program Listing

```
0126-- 0 ---00        0159-- O ---71
0127-- 0 ---00        0160--CNT---47
0128-- 0 ---00        0161-- 1 ---01
0129--LBL---51        0162-- 0 ---00
0130-- + ---33        0163-- 8 ---10
0131--CLR---20        0164--CLR---20
0132--FMT---42        0165--CLR---20
0133--FMT---42        0166--STP---41
0134-- C ---61        0167--RUP---22
0135-- O ---71        0168--LBL---51
0136-- N ---73        0169-- - ---34
0137--XTO---23        0170--XTO---23
0138-- E ---60        0171-- + ---33
0139-- N ---73        0172-- b ---14
0140--XTO---23        0173-- b ---14
0141--YTO---40        0174--X>Y---53
0142--CNT---47        0175--STP---41
0143-- O ---71        0176--CNT---47
0144-- F ---16        0177--CNT---47
0145--CNT---47        0178--CNT---47
0146-- a ---13        0179--PNT---45
0147-- E ---60        0180--XFR---67
0148-- G ---15        0181--IND---31
0149-- I ---65        0182-- b ---14
0150--YTO---40        0183--PNT---45
0151--XTO---23        0184--PNT---45
0152-- E ---60        0185-- 1 ---01
0153-- a ---13        0186--GTO---44
0154--YTO---40        0187--LBL---51
0155--CLR---20        0188-- - ---34
0156-- 0 ---00        0189--CNT---47
0157--CNT---47        0190--END---46
0158--XTO---23
```

Partial Data Printout

```
CONTENTS OF REGI                5.00
STERS                         252.00
0 TO 108
                                6.00
                0.00            0.00
                0.00
                                7.00
                1.00            0.00
                9.00
                                8.00
                2.00           47.22
               28.00
                                9.00
                3.00            5.31
                0.00
                               10.00
                4.00            0.00
                8.89
```

Extending Lagrangian Interpolation

by M. Jean-Pierre Borgogno of Marseilles, France

This suggestion modifies the Lagrangian Interpolation program, III-8 in the 9810 Math Pack, to compute up to n = 19. It consists of changing four program steps:

Existing Step			Change To	
Step	Key	Code	Key	Code
0015	2	02	3	03
0016	3	03	0	00
0092	2	02	3	03
0093	3	03	0	00

EXAMPLE: $y = \frac{x^2}{4}$, n = 19

Find y at x = 2.50
x = 7.50
x = 18.50
x = 19.50

```
  1.00*          13.00*
  0.25           42.25

  2.00*          14.00*
  1.00           49.00

  3.00*          15.00*
  2.25           56.25

  4.00*          16.00*
  4.00           64.00

  5.00*          17.00*
  6.25           72.25

  6.00*          18.00*
  9.00           81.00

  7.00*          19.00*
 12.25           90.25

  8.00*           2.50*
 16.00            1.56

  9.00*           7.50*
 20.25           14.06

 10.00*          18.50*
 25.00           85.56

 11.00*          19.50*
 30.25           95.06

 12.00*
 36.00
```

Special Label Sequence

by Cristian Langfelder, Hewlett-Packard, Böblingen, West Germany

This tip demonstrates the usefulness of the 'LBL' key in branching routines.

Up to three normal program steps may be inserted after a conditional branching instruction without canceling the branch condition. In the following sequence:

```
0154--- a ----13
0155--- UP----27
0156--- 0 ----00
0157---X>Y----53
0158--- YE----24
0159--- a ----13
0160---SFL----54
0161---GTO----44
0162---LBL----51
0163--- π ----56
0164--- DN----25
0165---XSQ----12
0166--- 0 ----00
0167--- 0 ----00
```

The 'LBL π' instructions will be ignored if the condition for branching is not met.

Program Listing

```
0000---CLR----20      0026---XFR----67
0001---STP----41      0027---IND----31
0002---XTO----23      0028--- b ----14
0003--- + ----33      0029---PNT----45
0004--- a ----13      0030--- UP----27
0005---XTO----23      0031--- a ----13
0006---IND----31      0032---DIV----35
0007--- b ----14      0033--- DN----25
0008---CLX----37      0034---PNT----45
0009---STP----41      0035---PNT----45
0010---IFG----43      0036---CLX----37
0011--- 0 ----00      0037--- UP----27
0012--- 0 ----00      0038--- b ----14
0013--- 2 ----02      0039---X=Y----50
0014--- 3 ----03      0040--- 0 ----00
0015--- UP----27      0041--- 0 ----00
0016--- 1 ----01      0042--- 5 ----05
0017---XTO----23      0043--- 1 ----01
0018--- + ----33      0044--- 1 ----01
0019--- b ----14      0045---XTO----23
0020--- DN----25      0046--- - ----34
0021---GTO----44      0047--- b ----14
0022--- 2 ----02      0048---GTO----44
0023--- a ----13      0049--- 2 ----02
0024---PNT----45      0050--- 6 ----06
0025---PNT----45      0051---END----46
```

Non-Zero Data Printout

by A. S. Hausrath, manager of mechanical design for the Minuteman, Systems Group of TRW, Inc., San Bernardino, California

For debugging purposes, it is frequently desirable to be able to "dump" the contents of only the data registers containing other than zeros. It is convenient to keep a program handy that will be compatible with almost any program in the 9810 and be able to provide this dump. This requirement more or less precludes the use of symbolic addresses.

By entering the following program to end at the highest-numbered step in the memory, and using fixed addresses, the chances of interfering with the main program are minimized. Note that no end statement is needed. The program lists the contents of the a and b registers, and lists all other non-zero data entries and their locations.

Sample Data Print-Out

```
A:B
 1.200000000  01         5.800000000  01
 2.700000000  01         1.000453700  00

 0.000000000  00         7.200000000  01
 7.563648732  04         8.549370000  01

 5.000000000  00         1.000000000  02
 2.245678000  01         9.200000000 -03

 4.300000000  01
 4.336385700  02
```

Program Listing

```
2000---FMT-----42      2018--- 2 -----02
2001---FMT-----42      2019--- 0 -----00
2002--- A -----62      2020--- 2 -----02
2003---CLX-----37      2021--- 7 -----07
2004--- B -----66      2022--- a -----13
2005---FMT-----42      2023---PNT-----45
2006--- a -----13      2024--- DN-----25
2007---PNT-----45      2025---PNT-----45
2008--- b -----14      2026---PNT-----45
2009---PNT-----45      2027--- 1 -----01
2010---PNT-----45      2028---XTO-----23
2011---CLR-----20      2029--- + -----33
2012---XFR-----67      2030--- a -----13
2013---IND-----31      2031---GTO-----44
2014--- a -----13      2032--- 2 -----02
2015--- UP-----27      2033--- 0 -----00
2016--- 0 -----00      2034--- 1 -----01
2017---X=Y-----50      2035--- 2 -----02
```

Pen Drop Control

by Dr. James Lindauer, MD of San Francisco General Hospital

The routine overrides the pen drop when proceeding to the first point plotted, then allows plotting of a solid or dashed line. This applies to the 9810 with the plotter ROM.

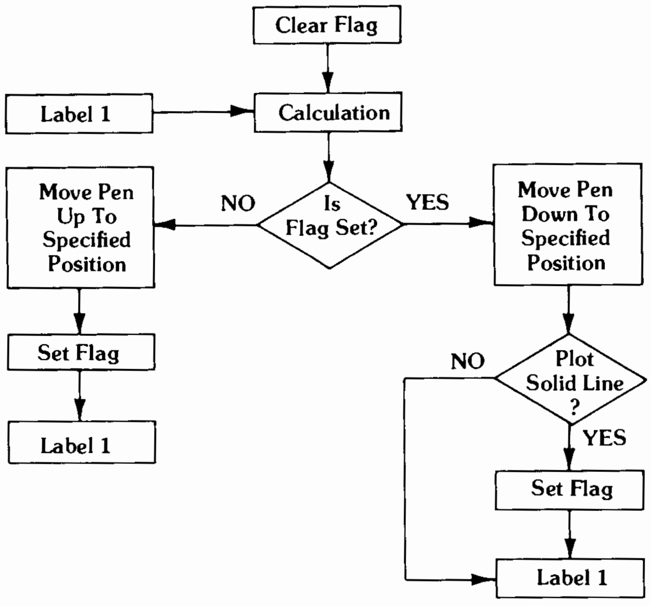


Figure 1. Flow Chart

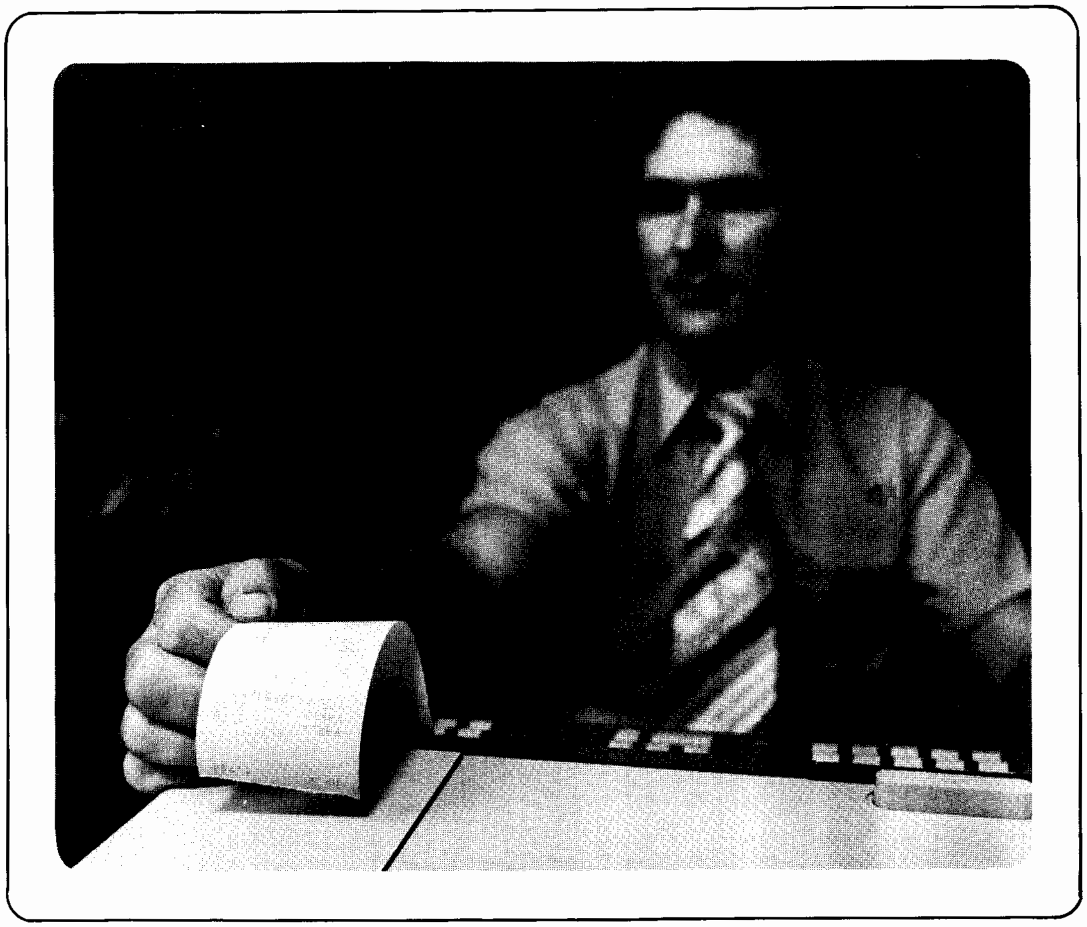
Sample Program

The routine has been incorporated into a program to plot the line X = Y, in increments of 100 until Y = 9000.

```
0000--- 9 -----11      0037--- b -----14
0001--- 9 -----11      0038--- a -----13
0002--- 9 -----11      0039--- UP-----27
0003--- 9 -----11      0040--- b -----14
0004--- UP-----27      0041---IFG-----43
0005---CLX-----37      0042---GTO-----44
0006---FMT-----42      0043---LBL-----51
0007--- 1 -----01      0044--- a -----13
0008--- 2 -----02      0045---CNT-----47
0009---FMT-----42      0046---SFL-----54
0010--- 1 -----01      0047---FMT-----42
0011--- 3 -----03      0048--- 1 -----01
0012---CLR-----20      0049--- UP-----27
0013---STP-----41      0050---CNT-----47
0014---XTO-----23      0051---GTO-----44
0015--- 0 -----00      0052---LBL-----51
0016---LBL-----51      0053--- 1 -----01
0017--- 1 -----01      0054---LBL-----51
0018--- 9 -----11      0055--- a -----13
0019--- 0 -----00      0056---FMT-----42
0020--- 0 -----00      0057--- 1 -----01
0021--- 0 -----00      0058--- DN-----25
0022--- UP-----27      0059---CNT-----47
0023--- a -----13      0060---XFR-----67
0024---X>Y-----53      0061--- 0 -----00
0025---STP-----41      0062--- UP-----27
0026---CNT-----47      0063--- 1 -----01
0027---CNT-----47      0064---X=Y-----50
0028---CNT-----47      0065---GTO-----44
0029--- 1 -----01      0066---LBL-----51
0030--- 0 -----00      0067--- 1 -----01
0031--- 0 -----00      0068---CNT-----47
0032---XTO-----23      0069---SFL-----54
0033--- + -----33      0070---GTO-----44
0034--- a -----13      0071---LBL-----51
0035---XTO-----23      0072--- 1 -----01
0036--- + -----33      0073---END-----46
```

Section 7

General



Index

Section 7 - General

9800 Program Verification

by Don Sullivan, Raytheon, Burlington, Massachusetts

To check a program that was printed out correctly at a previous date but operates incorrectly when reentered in the calculator, list the program again. Place the two printouts over each other and hold them up to the light. Any differences show up readily, allowing corrections to be made.

Remember that occasional cleaning of the magnetic heads in your cassette reader or card reader is needed to maintain high reading accuracy.

Reducing Forward Search Time In Cassette Applications

The method presented here is useful in any application requiring the use of large files (sizes greater than 50 registers) on the Cassette Memory. A forward search is initiated by the LOAD FILE, FIND FILE, or the RECORD INTO FILE command when the current file number is less than the file number being searched for. The forward search procedure is to fast search up to the file before the one in question and then do a slow search until the file in question is found. A significant delay occurs when the slow search is done through a long file. This delay does not occur in a backward search because backward searches are performed entirely in the fast mode.

When the tape is initially marked, insert a short file (minimum 1 register) between any two files in which the first file is longer than approximately 50 registers. The effect of this added file is that the slow search through a short file is barely detectable. The cost of this reduced search time is 54 words for header information for the file and 6 words to store 1 register of data, or 60 words per 1-register file. Since the approximate total number of words on a tape is 44,000, one 1-register file will occupy only .1363% of the total tape capacity. Seventy-three such buffer files will occupy 9.95% of the tape capacity.

Cassette Commands

	9810A	9820A
Find File	FMT, 5, 5, CLX	FDF
Load Program	FMT, 5, 5, CNT (S/R)	LDF
Load Data	FMT, 5, 5, XFR	LDF
Record Program	FMT, 5, 5, K	RCF
Record Data	FMT, 5, 5, XTO	RCF

	9830A
Find File	FIND
Load Program	LOAD
Load Data	LOAD DATA
Record Program	STORE
Record Data	STORE DATA

Improved Tape Identification (9865A)

by A. Scott Parrish, Bureau of Research, Maryland Department of Transportation, Brooklandville, Maryland

This tip concerns the tape identification program in the 9865A cassette memory pac for the 9815. The program has been altered so that each file marked on the tape is identified whether there are any errors or not. The changes begin at step 209.

Mr. Parrish's changes make the program identify each file as it finds it rather than using register 'a' as a counting sequence.

```
0209---FMT----42        0236--- 1 ----01
0210--- 5 ----05        0237--- 3 ----03
0211--- 5 ----05        0238---LBL----51
0212---CLX----37        0239--- 4 ----04
0213---FMT----42        0240---FMT----42
0214--- 5 ----05        0241---FMT----42
0215--- 5 ----05        0242---CLR----20
0216---EEX----26        0243--- X ----36
0217---PNT----45        0244--- X ----36
0218---FMT----42        0245--- X ----36
0219--- 5 ----05        0246--- X ----36
0220--- 5 ----05        0247--- X ----36
0221---CLX----37        0248--- X ----36
0222---RUP----22        0249--- X ----36
0223---PNT----45        0250--- X ----36
0224---RUP----22        0251--- X ----36
0225---PNT----45        0252--- X ----36
0226---RUP----22        0253--- X ----36
0227---PNT----45        0254--- X ----36
0228---PNT----45        0255--- X ----36
0229---FMT----42        0256--- X ----36
0230--- 5 ----05        0257--- X ----36
0231--- 5 ----05        0258--- X ----36
0232---CHS----32        0259---CLR----20
0233---GTO----44        0260---FMT----42
0234--- 0 ----00        0261---S/R----77
0235--- 2 ----02        0262---END----46
```

Magnetic Card Versatility (9810, 9820)

The magnetic cards designed for use with the 9100A/B Calculator can be used to record programs for the 9810 and 9820. One side of a 3 5/8 inch (9.2 cm) card, Part No. 9320-1144, will record an average of 225 program steps on each side. Card sides can be recorded sequentially until the 9810 INSERT CARD light extinguishes, or until NOTE 14 no longer appears in the 9820 display. Similarly, the 6 inch (15.2 cm) cards, Part No. 9162-0012, for the Model 10 can be used for the Model 20 for short programs.

Use of the 10 ½ inch (26.7 cm) magnetic cards, Part No. 9162-0045, for recording longer programs on the 9810 may provide both economy and increased loading and storage convenience.

Changing Programs From the HP 65 to 9815A

by Neville Joseph, Bucks, England

It is fairly clear that the programming languages of the 9815A and the HP 65/67 are similar, and no doubt a number of readers have converted programs from the smaller machines, normally a fairly trivial procedure with obvious differences such as conditional skips.

Not so obvious (and not appearing in the 9815A manual) is the different treatment of Last X after a RECALL instruction. The HP65 leaves Last X unchanged, while the 9815A loads the old X (and new Y) into it.

I hope that publicity on this point will save some of my colleagues a little debugging time.

Making Dashed Plots with the 9862A

by J.N. Shapiro and R.J. Woodward, Texas A&M University, College of Geosciences, College Station, Texas

Hewlett-Packard's routine for making dashed plots with the 9862 (page 3-6 of the 11220A Peripheral Control I Operating Manual) alternates solid lines a preselected number of x units long with spaces a preselected number of x units long. If the function being plotted has a small slope, the resulting plot can be made to consist of dashes (and spaces) of about the same length, as desired. However, if the function has steep parts, the length of the dashes (and spaces) can get very long, theoretically approaching infinity for a function of infinit slope; for example, ln x as $x \rightarrow 0$. See Figure 3.9 on page 3-6 of the above reference for an example using cos 3x.

The reason for this is very simple. Equal increments in the independent variable, x, are not generally equal increments in arc length, s. For plotting purposes, it is equal increments in s which are desirable.

Mathematically speaking, the effects of a changing slope may be taken into account quite easily. One simply uses the definition of ds, a differential element of arc length.

$$ds = \sqrt{dx^2 + dy^2}$$

$$ds = dx\sqrt{1 + (dy/dx)^2}$$

Here dy/dx may be calculated for each point, or the finite difference $\Delta y/\Delta x$ between adjacent points may be calculated.

In practice two other effects must be considered. First, both y and x must be scaled by the total number of units of each covered by the plot. That is, a line corresponding to .1 y units will cover .1/10 = .01 of the graph's height if y goes from 0 to 10 (or −5 to 5, etc.), whereas the same .1 units will plot twice as long if y goes instead from 0 to 5.

A second effect is the physical size of the graph. The appropriate coordinates should be multiplied by the length of the graph in their direction. In terms of units along the x axis, and including both of the above effects, ds is given by

$$ds = dx\sqrt{1 + (A\ dy\&dx)^2}$$

$$\text{where } A = \frac{(x\max - x\min)}{(y\max - y\min)} \bullet \frac{(\text{height of graph})}{(\text{length of graph})}$$

The program (Figure 1) generates equal length dashes as shown in the two plots of ln x, one with equal size dashes and one without (Figure 2). Note that the x increment should be small and the number of increments per dash should be large for best dash equality.

This suggestion is interesting because of its utility, but perhaps even more so because it illustrates very simply an elementary concept of calculus.

```
0:
ENT "X MIN",R0," X MAX",R1⊢
1:
ENT "Y MIN",R2," Y MAX",R3⊢
2:
ENT "LENGTH/HEIGHT",R4⊢
3:
(R1-R0)/(R3-R2)R4→A⊢
4:
0→Z;SCL R0,R1,R2,R3⊢
5:
ENT "X INCREMENT",B⊢
6:
ENT "INCR. PER DASH",C⊢
7:
R0-B→X⊢
8:
GSB "*"⊢
9:
PLT X,LN X⊢
10:
IF Z≤C;GTO -1⊢
11:
PEN ⊢
12:
GSB "*"⊢
13:
IF Z≤2C;GTO -1⊢
14:
0→Z;GTO -6⊢
15:
GTO +3⊢
16:
"*":X+B→X;IF X>R1;GTO +3⊢
17:
Z+r(1+AA/XX)→Z⊢
18:
RET ⊢
19:
END ⊢
R366
```

Figure 1. Routine for equal dash length.

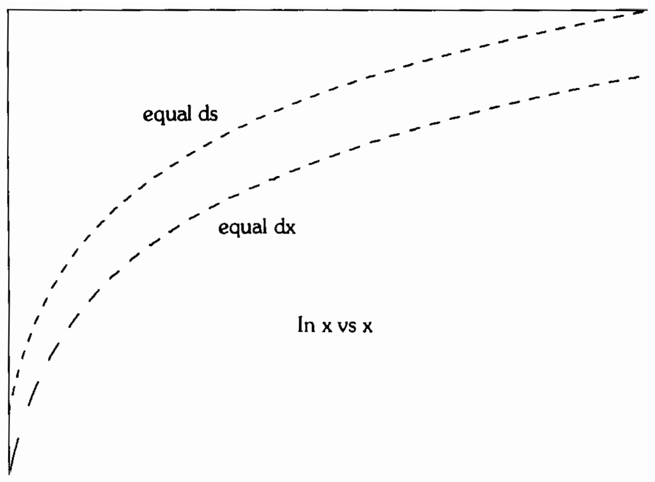


Figure 2. Comparison between using equal x increments and equal arc increments for dash length.

HEWLETT PACKARD

Printed in U.S.A.
(2/80)13.6K