

4. PROGRAM HINTS AND EXAMPLES

What is a program?

At its very simplest, a program is just a sequence of commands, each of which performs an action. On the HP38G, a colon (:) is used to separate commands from one another, while a semicolon (;) is used to separate the arguments to a single command.

It is possible to write a program directly on the HOME screen in the Editline.

For example,

```
1 STO> A: 2 STO> B: 3 STO> C
```

followed by **ENTER** will return a result of 3 in Ans (the last command was 3 STO > C), but all three STO > commands were executed.

The PROGRAM CATALOG

To save a program and give it a specific name, press **PROGRAM**, and you will see a title bar for the PROGRAM CATALOG. This contains a directory of all your programs (even including the Editline). The menu keys here are:

E D I T N E W S E N D R E C V R U N

To create a new program, press **NEW**. You are prompted for a name for the program, After you type in the name of your choice (it can be virtually anything you please and of any length), press **ENTER** or **OK**.

The PROGRAM EDITOR

Now you are in the editor for your specific program, and the title bar should indicate the name of the program. For example, if you named your program MINE, then the title bar should read MINE PROGRAM at the top of the screen. The menu keys now are:

S T O > S P A C E A ... Z B K S P

The **A...Z** key is an "alpha-lock" which is can be toggled on or off (when it is on you will see a white square lit up on the menu label and the α symbol lit at the top of the screen). If you press the coloured shift key before toggling on **A...Z**, it changes the key to **a ... z**, a lower-case "alpha-lock".

Typing in your program

The flashing cursor arrow is an insertion marker. While typing a program you can use the directional arrows for moving from line to line or from character to character.

- SPACE** is on the menu for typing convenience
- BKSP** (backspace) is on the menu for typing convenience
- STO>** is the one command provided on the menu (performs the same action as the STO> key in HOME).
- DEL** deletes the character under the insertion arrow (or the last character if you are at the very end)
- ENTER** is a line feed

Numbers, letters, and other characters can be typed from the keyboard as usual. You will find yourself making frequent use of the **CHARS** menu (for retrieving relational symbols like <, >, ², ³ and the ' or " symbols). To use **CHARS**, use the directional arrows to highlight the character you want, and then press **OK**. If there are several characters in a row that you want to retrieve from the **CHARS** menu, press **ECHO** first and then each of the characters you want (press **OK** when you are done). Functions and variable names can either be typed in character-by-character or retrieved from the **VAR** or the **MATH** menus.

All other programming commands (other than **STO>**) are found in the **MATH** menu when **CMDS** is toggled on. An overview and the syntax for all these commands are found in PART 2 of this reference.

When you are finished...

Once you are finished typing in a program, there is no special action necessary to "EXIT". Simply *go somewhere else!* For example, you could press **HOME** to return to the **HOME** screen, or you could press **PROGRAM** to create a new program or **RUN** the program you have just written.

If you highlight the name of a program in the PROGRAM CATALOG, you can **EDIT** it (this returns you to the program editor where you can make changes). **SEND** and **RECV** are used to send and receive programs through infra-red communications with another HP38G, or by serial wire connection to a disk drive.

Cutting and pasting the contents of one program into another

Sometimes you may find that you have several lines of program code that you want to use in another program (perhaps with some slight editing). Rather than type the code in again "from scratch", there is a neat way to cut and paste one program into another:

1. Position the flashing cursor at the location in the current program where you want to paste in the contents of another program
2. Press VAR
3. highlight the Program category and press OK
4. highlight the name of the program whose contents you want to paste
5. toggle on the **VALUE** menu key (instead of **NAME**)
6. press OK

Program structures

Programs that are simply sequences of commands can be very useful, but the real power of programs becomes evident when they allow for *conditional* execution of commands (branching) or *repeated* execution of several commands over and over again (looping). You will see that the HP38G's programming language does not use line numbers or labels and GO TO statements to accomplish branching and looping.

HP38G Branching structures

You can either type these commands in character-by-character or retrieve them from the Branch category of the **CMDS** menu in **MATH**.

IF ... THEN ... ELSE ... END

The syntax for this structure is as follows:

IF test clause Test clause is some statement that is true (1) or false (0)

THEN

do some thing:

do some other thing: These statements are executed if the test clause is true

and so on

ELSE (The ELSE part is optional)

do some thing:

do some other thing: These statements are executed if the test clause is false

and so on

END: Marks the end of the IF THEN ELSE structure

NOT ES:

1. In the statements that follow either THEN or ELSE, it's possible to have loops or additional branch structures, including other IF THEN ELSE END structures.
2. The test clause can actually be any expression, and could have a value other than 1 or 0. The **THEN** part is executed if the test clause has a nonzero value, while the **ELSE** part is executed only if the test clause has the value zero.

Example:

```
IF A < 0
THEN
-A STO> A:      If A is negative, then the sign of A is reversed.
THEN           (Note that this piece of code has no ELSE part.)
```

Example: *This program records the sign of A in S*

```
IF A < 0
THEN
- 1 STO> S:     S = - 1 if A is negative
ELSE
IF A = 0
THEN
0 STO> S:      S = 0 if A = 0
ELSE
1 STO> S:      S = 1 if A is positive
END:
END:
```

CASE

This structure is handy if you have several clauses to test in order, and you want to branch when you encounter the *first* true clause. The syntax for CASE is:

```
CASE
IF first test clause
THEN
do some thing:
do some other thing:
and so on:
END
IF second test clause
THEN
do some thing:
do some other thing:
and so on:
END
.
.
.
END
```

Here are two examples to illustrate the distinction between the use of CASE and simply a sequence of several IF THEN statements.

Example 1:

```
CASE
IF I > 5
THEN 1-5 STO> S:
END
IF I ≥ 5
THEN 0 STO> S:
END
IF I < 5
THEN 5 - I STO> S:
END
END:
```

Example 2:

```
IF I > 5
THEN 1-5 STO> S:
END:
IF I ≥ 5
THEN 0 STO> S:
END:
IF I < 5
THEN 5 - I STO> S:
END:
```

Suppose we store 7 in | (7 STO> | on the HOME screen), and then run the first program involving CASE. The result will be that 2 is stored in S, because the clause in the second IF THEN statement won't even be tested.

In contrast, if the second program is run, the final result will be that 0 is stored in S, for *all* three of the clauses in the IF THEN statements will be tested.

NOTE the punctuation differences between these two examples, particularly the use of colons.

IFERR ... THEN ... ELSE

This structure is one designed for "error trapping." Some program statements may result in an error (thus causing the program to stop) depending on the values of certain variables at the time of execution. IFERR will check any number of statements for such errors and give you the chance to gracefully branch.

The syntax for IFERR is:

IFERR

first command:

second command:

et cetera:

THEN *If an error resulted from any of these commands*

do some thing: *then these statements are executed*

do some other thing:

and so on:

ELSE *Optional---if no error resulted*

do some thing else: *then these statements are executed instead*

do some other thing

else:

and so on:

Example:

IFERR

ABS(COT(X) STO> C: *If this statement results in an error*

THEN *(for example, when X= 0)*

1 STO> A: *then store 1 in A*

MAXREAL STO> C: *and store the largest real number in C*

ELSE *If no error resulted,*

0 STO> A: *then store 0 in A instead*

END:

For example, if X had the value 0 at execution, the attempt to store ABS(COT(X)), as the value of C would result in an error.

Rather than stopping, this program branches to store the largest real number that can be represented on the HP38G as the value of C, and 1 is stored in A (as a marker that an error occurred, perhaps).

If the X has a value that does not result in an error, then ABS(COT(M) is stored in C and 0 is stored in A.

RUN

You can also branch to other programs using the RUN command. The syntax is:

RUN name: where name is the name of the program.

HP38G Looping Structures

You can either type these commands in character-by-character or retrieve them from the Loop category of the **CMDS** menu in **MATH**.

FOR = TO ... STEP ... END

The syntax for this basic loop structure is:

FOR < index variable > = < start value > TO < end value > STEP < increment value >;

do some thing:

do some other thing:

and so on:

END:

Example:

0 STO> S:

1 STO> P:

FOR 1 = 1 TO 10 STEP 1;

S + 1 STO> S:

P * I STO> P:

END:

The result of this program is that the sum of the integers 1 through 10 is stored in **S**, and their product is stored in **P**.

DO ... UNTIL ... END

This loop structure will repeatedly execute a body of statements until some test clause is true. The syntax is:

```
DO
some thing:
some other thing:
and so on:
UNTIL <test clause>
END:
```

Example:

```
0 STO> S:
1 STO> P:
1 STO> I:
DO
S + 1 STO> S:
P * 1 STO> P:
1 + 1 STO> I:
UNTIL 1 > 10
END:
```

The result of this program is the same as the previous one: the sum of the integers 1 through 10 is stored in **S**, and their product is stored in **P**.

WHILE ... REPEAT ... END

This structure repeats the execution of some body of statements while a test clause remains true. The syntax is:

```
WHILE <test clause>
REPEAT
do some thing:
do some other thing:
and so on:
END:
```

Example:

```
0 STO> S:
1 STO> P:
1 STO> I:
WHILE I ≤ 10
REPEAT
S + I STO> S:
P * I STO> P:
I + I STO> I:
END:
```

The result of this program is the same as the previous two: the sum of the integers 1 through 10 is stored in **S**, and their product is stored in **P**.

Example:

This example assumes that a positive integer has been stored in **M**. Then it sets $N = M$ and calculates $3N + 1$ if N is odd or $N/2$ if N is even. This becomes the new value of N and the process is repeated until $N = 1$. The value of **S** at the end of the program's execution is the number of different values of N . ($S=0$ if M is not a positive integer).

NOTE: It is an open question whether **S** is finite for all positive integers M .

```
IF (M ≤ 0) OR (M ≠ INT(M))
THEN 0 STO> S:
ELSE 1 STO> S:
M STO> N:
WHILE N ≠ 1
REPEAT
IF N MOD 2
THEN 3 * N + 1 STO> N:
S + 1 STO> S:
ELSE
N / 2 STO> N:
S + 1 STO> S:
END:
END:
```