Programming the hp 49g+ in User RPL language using your PC

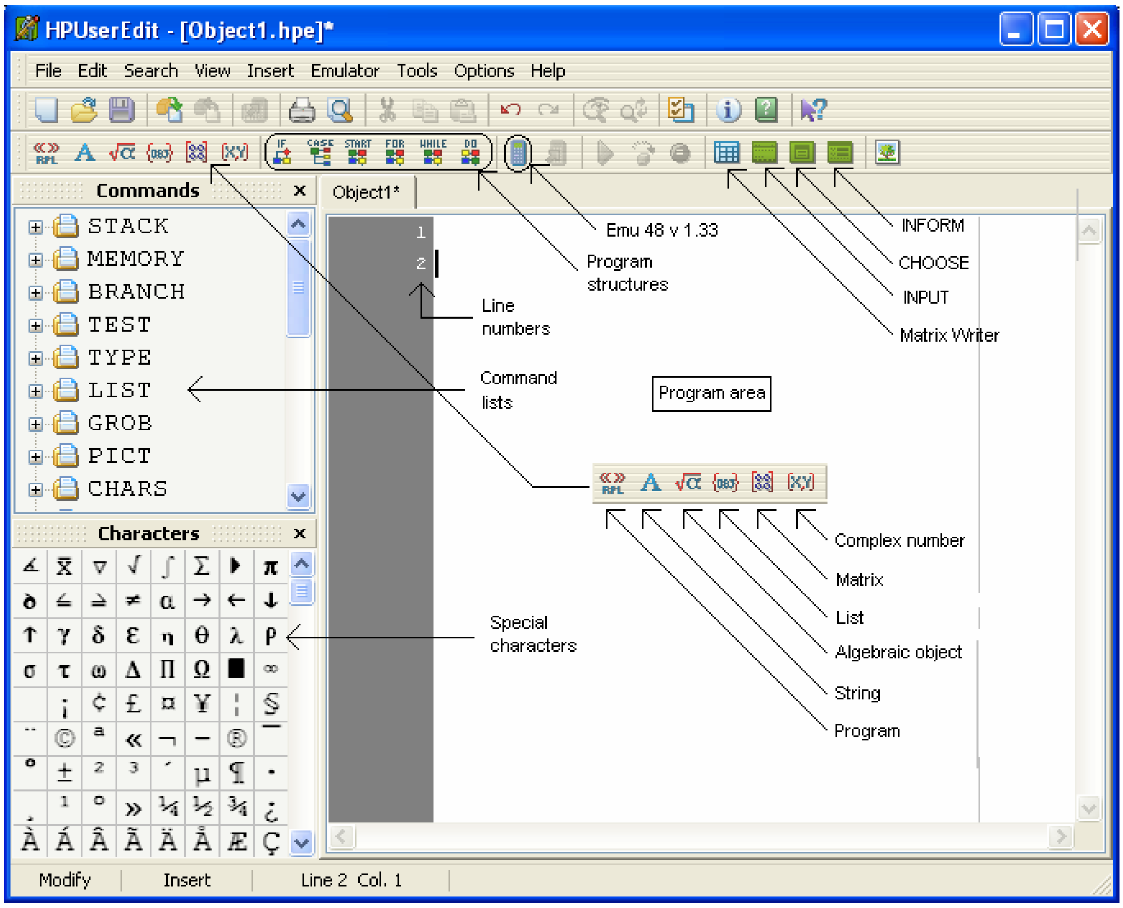
by Gilberto E. Urroz, March 2006.

These instructions are aimed at the production of User RPL language programs for the *hp 48gii*, *hp 49g*, and *hp 49g+* graphic calculators using a personal computer. The programs can be developed in the PC using the *HPUserEdit 4.0* editor program. After developing your program in *HPUserEdit 4.0*, you can load it into the *Emu48* emulator program set up with the *hp 49g* ROM (unsupported ROM 1.19-6, dated from mid 2003), or with the *hp 49g+* ROM (version 2.0, dating from 2005). You can test and debug the program in the emulator and then save it to a file, that can then be transferred to the calculator via the hp communications software.

Instructions for the installation of *Emu48* and *HPUserEdit 4.0* are available in a different document found in the same web site where you downloaded this document. The same web site provides references on *UserRPL* programming.

Example 1 - Developing a program in *HPUserEdit 4.0*

Figure 1, below, shows the interface for *HPUserEdit 4.0*. (Press *Cntl L* to see the line numbers, if needed.)

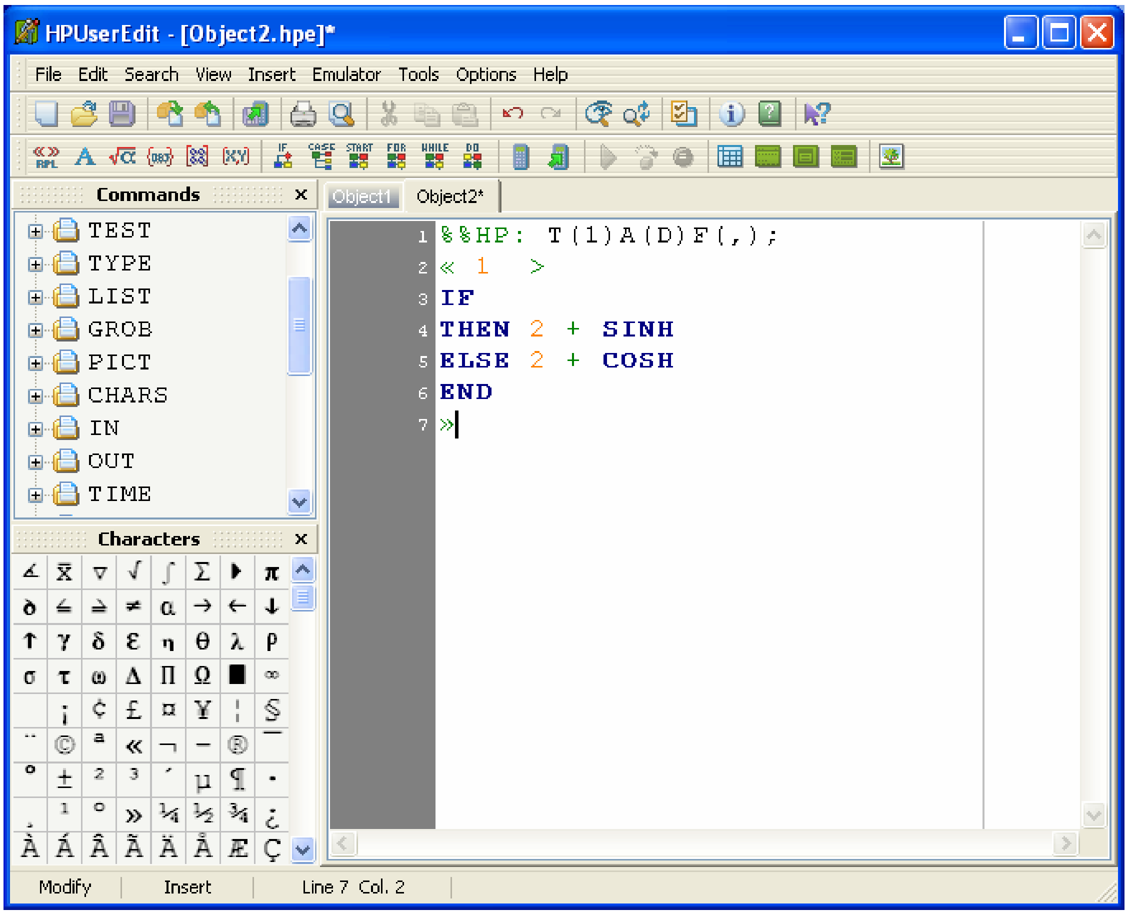


The figure shows the main program area with line numbers on the left margin, the folders containing command lists, a menu for object delimiters (starts with *Program*), a menu for program structures, a menu for input commands, and a list of special characters. The tab atop the programming area shows the default file name for the program, namely, *Object 1*. Click the IMPORT button:

and import the example whose name is *ejemplo1.hp*. The first line in the program contains the characters:

```
%%HP: T(0)A(D)F(.);
```

This line is needed when transferring an ASCII program file through the Kermit file transfer protocol (Kermit is a protocol developed in the early 1990s). Both Kermit and the more recent transfer protocol XMODEM are available in the calculators and the hp communications software. At this point, we'll keep the first line as shown above while we type the following program:



```
%%HP: T(1)A(D)F(,);
« 1   >
IF
THEN 2 + SINH
ELSE 2 + COSH
END
»
```

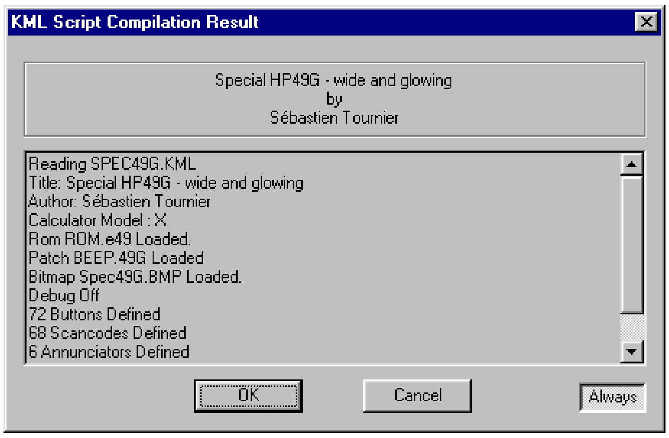
Notice that in this case we use the IF … THEN … ELSE … END command structure and a couple of functions from the HYPERBOLIC folder in the list of commands. To enter the program containers, << >>, double click on the *PROGRAM* button. This will place the opening and closing program containers in the program area leaving the cursor in between. At this point, type the characters "1 >". To load the IF … THEN … ELSE … END command simply click on the IF button. Move the cursor in front of the word THEN and type the characters "2 +". To load functions from the HYPERBOLIC folder, first click on the HYPERBOLIC folder on the left-hand side of the interface. This action opens the folder to show all the HYPERBOLIC functions. Then, to load SINH or COSH, simply double click on the corresponding name. Complete the program as shown above.

The operation of this program is equivalent to the function *f(x) = sinh(x+2)* for *x>1*, or *f(x) = cosh(x+2)* for *x<1*. The value of *x* is the number entered in stack level 1, in RPN mode, before activating the program.

To see the program in operation, first save the program to a file called *Prog1.hpe* by using the option *File>Save As...* in *HPUserEdit 4.0*.

Running the program in *Emu48*

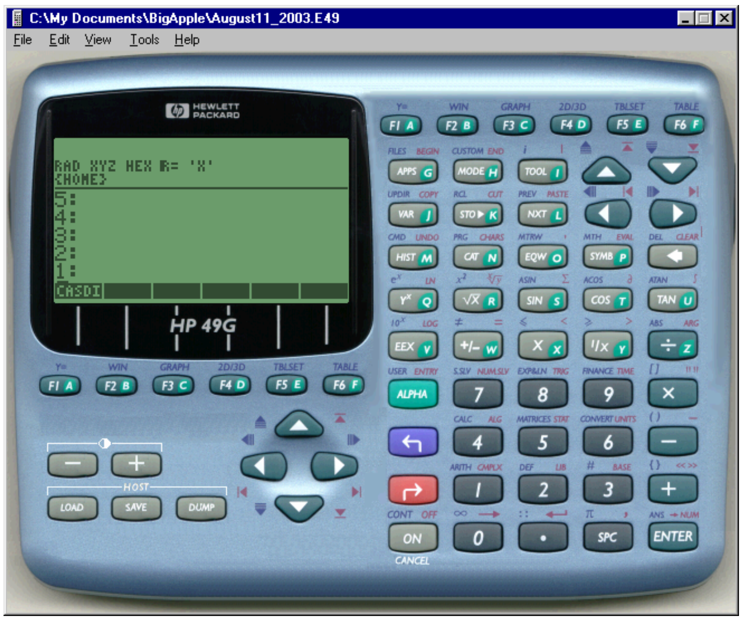
Activate the emulator program *Emu48* making sure that an HP 49G "skin" is selected. For example, when running *Emu48*, you may get the following response:



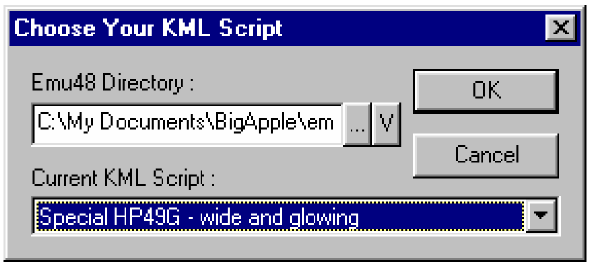
Special HP49G - wide and glowing
by
Sébastien Tournier

```
Reading SPEC49G.KML
Title: Special HP49G - wide and glowing
Author: Sébastien Tournier
Calculator Model : X
Rom ROM.e49 Loaded.
Patch BEEP.49G Loaded
Bitmap Spec49G.BMP Loaded.
Debug Off
72 Buttons Defined
68 Scancodes Defined
6 Annunciators Defined
```

This window indicates that a file named *SPEC49G.KML* will be loaded which uses the HP 49G ROM (the fifth line reads: *Rom ROM.E49 Loaded*). This "skin" will produce a *Special HP49G - wide and glowing*, which, as you will see, is not the standard HP 49G format. Press the *OK* button to activate the emulator. This is what we get for this "skin":



If you don't get this "skin" the first time you run the emulator, you can always change it by selecting the option *View>Change KML Script…* Click the down arrow in the selection list entitled *Current KML Script:* and select *Special HP49G - wide and glowing*. Finally, press the *OK* button to activate the emulator.

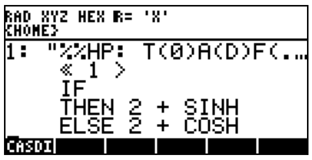


```
Choose Your KML Script
Emu48 Directory :
C:\My Documents\BigApple\em
Current KML Script :
Special HP49G - wide and glowing
```

I like this HP 49G "skin" because it fits in all type of computer screen sizes, and it shows the calculator keyboard intact on the right-hand side of the interface. It also shows the F1 through F6 keys under the screen for manipulating menu keys.

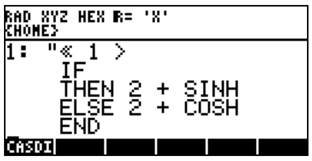
Notice that for programming development I prefer to use the calculator in RPN mode, and set system flag 117 to *soft MENU*.

To load the program we prepared in *HPUserEdit 4.0* use the option *Edit>Load Object..* Browse through your files until you locate the file *Prog1.hpe*, and press the *Open* button. The file will be listed in the calculator screen as a character string:



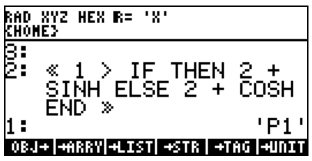
```
RAD XYZ HEX R= 'X'
{HOME}
1: "%%HP: T(0)A(D)F(.…
    « 1 >
    IF
    THEN 2 + SINH
    ELSE 2 + COSH
CASDI
```

Before storing the program into a variable, we need to remove the characters from the first line and convert the remaining string into a program. Press the down arrow key activate the text editor, remove the characters `%%HP: T(0)A(D)F(.);` from the string, and press ` .
The resulting string is now:



```
RAD XYZ HEX R= 'X'
{HOME}
1: "« 1 >
    IF
    THEN 2 + SINH
    ELSE 2 + COSH
    END
CASDI
```

To convert to a string use the OBJ→ command by typing „ ° TYPE OBJ². Then, store the resulting program into a variable called 'P1', i.e.,

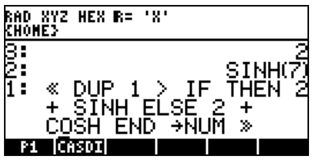


```
RAD XYZ HEX R= 'X'
{HOME}
3:
2:  « 1 > IF THEN 2 +
    SINH ELSE 2 + COSH
    END »
1:                  'P1'
OBJ→ →ARRY →LIST →STR →TAG →UNIT
```

Press K to store the variable, and press J to recover the variable menu. You should have a variable called P1 in your menu keys. To see the program in action, type a number, say 5, and press P1. The result is a 2 in the stack and an error message. The error message results from the fact that the value 2 is used for the IF comparison and, therefore, is no longer available for operating (adding 2 and calculating SINH). The program needs to be modified by adding a DUP command before the 1.

To effect this correction, press , @@P1@@, then press the down arrow key to see the program. Insert the function DUP before the 1 by using „ ° @STACK! @@DUP@. Then, press ` „ @@P1@@ to store the new version of the program in variable P1. Now, type a 5 and press @@P1@@. The correct result is now SINH(7).

If we wanted to produce a floating-point, rather than a symbolic, result, we may add function →NUM at the very end of the program. Edit the program to include this function following a procedure similar to that used earlier to add function DUP. The program, before storing it into P1, should look like this:



```
RAD XYZ HEX R= 'X'
{HOME}
3:                      2
2:                SINH(7)
1: « DUP 1 > IF THEN 2
   + SINH ELSE 2 +
   COSH END →NUM »
```

After storing the program into P1, and running with argument 5, the result is 548.3161….

In this exercise we illustrated the loading of a program into the emulator as well as their editing and modification. We could have also performed those modifications in the editor and re-loaded the program into the emulator. In the present case, the modifications were made in the emulator itself, therefore, they are not present in the original program file *Prog1.hpe*.

Transferring the program to the calculator

Before transferring the program to the calculator, we need to save the program to a binary file by listing the program in the emulator screen (*Edit>Save Object…*) and saving it into the name *Prog1* (no suffix). The program is now ready to be transferred to the calculator through the hp communications software.

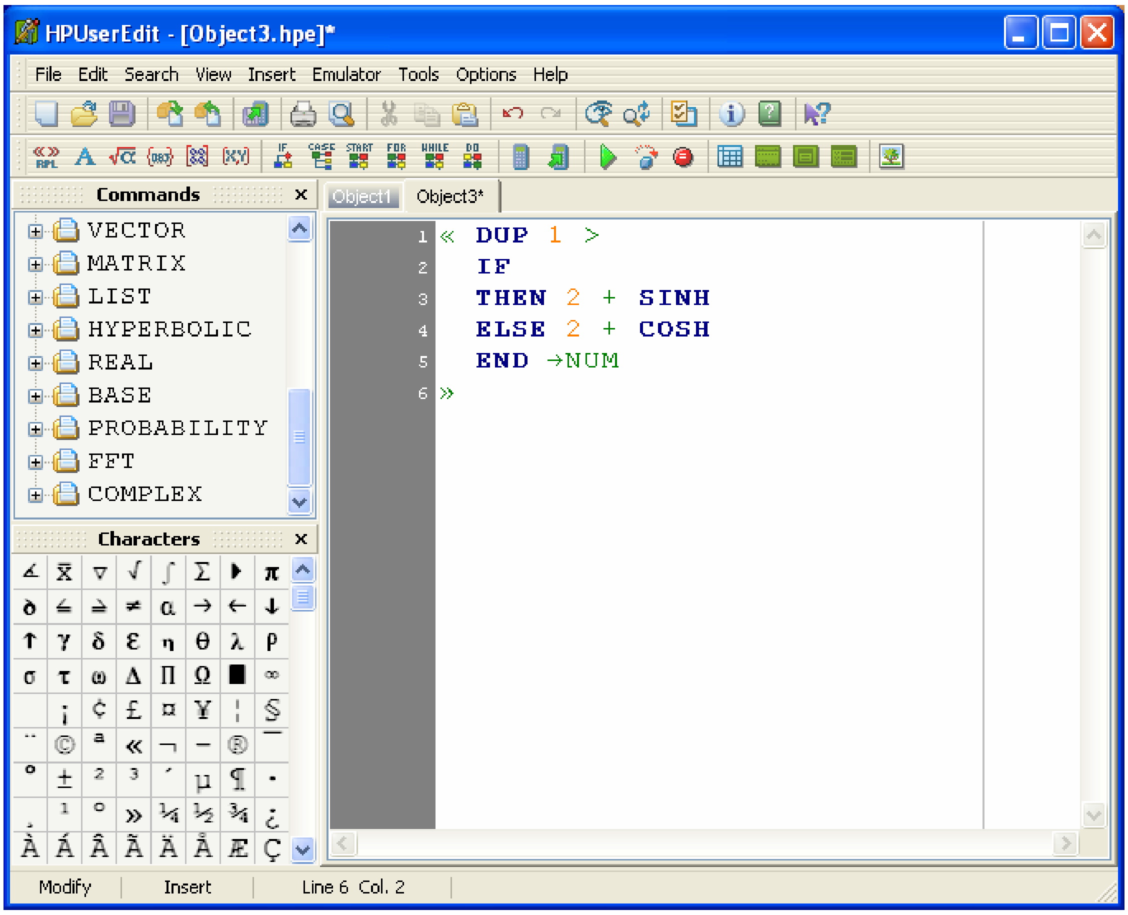
Please notice that this exercise shows a relatively short program that can simply be written in the calculator itself. The use of the editor (*HPUserEdit 4.0*) is justified only when writing a long and elaborate program that will be difficult to type and follow in the calculator's limited-size screen.

Example 2 - Typing a non-Kermit file in *HPUserEdit 4.0*

In this exercise we will type the complete text for the program developed earlier while eliminating the heading line for Kermit transmission (`%%HP: T(0)A(D)F(.);`).

Activate *HPUserEdit 4.0*, open a new file (use option *File>New*), and remove the top line in the file before typing the remaining lines in the program. Save the program to a file called *Prog2.hpe*. The *HPUserEdit 4.0* interface should look as shown in the following page.

To type the function name "→NUM". This function is not available in the function folders listed in the left-hand side of the interface. Therefore, its full name needs to be typed in the program area by the user.



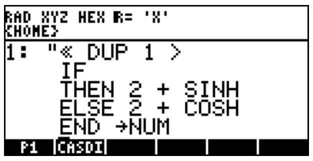
```
« DUP 1 >
  IF
  THEN 2 + SINH
  ELSE 2 + COSH
  END →NUM
»
```

NOTE: The user can type a calculator's function name into the program at any time. Thus, double-clicking on the function's name to enter it in the program is not the only way to enter function names.

To type the character "→", double click on the corresponding cell in the *Characters* in the *HPUserEdit 4.0* interface.

NOTE: Double clicking in any special character cell will enter the corresponding character at the current cursor location.

With the file saved in *Prog2.hpe* we can now load the file into the emulator by using the option *Edit>Load Object…* The resulting calculator screen is now:



```
RAD XYZ HEX R= 'X'
{HOME}
1: "« DUP 1 >
    IF
    THEN 2 + SINH
    ELSE 2 + COSH
    END →NUM
```

This object is delimited by double quotes (""), therefore, it represents a string object in the calculator. To convert it to a program, use function OBJ→, i.e., „ ° @TYPE@ @OBJ²@, and then save it into variable P2. Enter the argument 5, and press J @P2@ to run the program. The result is the same as before, namely, 548.3161….

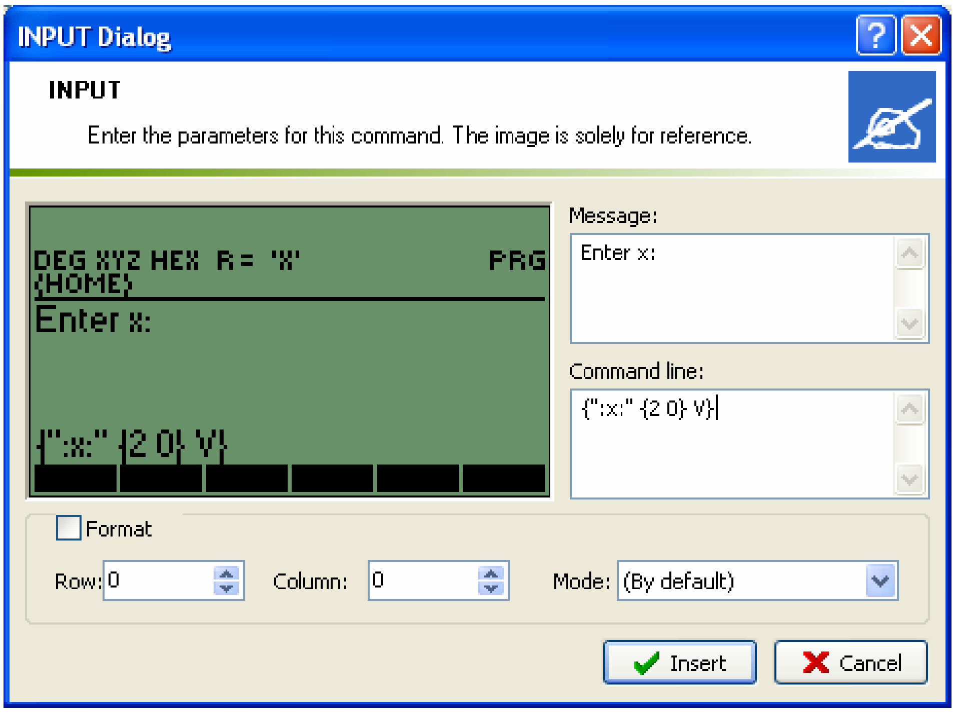
In summary, it is not necessary to retain the heading line (`%%HP: T(0)A(D)F(.);`) when creating a new file in the *HPUserEdit 4.0*, unless you will be using the Kermit file transfer protocol and transferring the file in ASCII format. The procedure for program debugging and transfer suggested in this document obviates the use of ASCII format, and, since XMODEM is the preferred mode of file transmission for the *hp 49g+* and the *hp 48gii*, we need not to worry about retaining the heading line.

NOTE: In the procedure suggested in this document, we first debug and test the program in the emulator and export it as a binary file, which is eventually transferred to the calculator. As an alternative, you could type your program in *HPUserEdit 4.0*, save it in your computer, and transfer it to your calculator without going through the emulator. The object received by the calculator will be a calculator string object, which will need to be converted into a program by using function OBJ→. In this case, you will do the debugging and testing of the program in the calculator itself.

Example 3 - Using the INPUT command for a single input

The INPUT command can be used as a simple way to prompt the user to enter data to a program. As an exercise, we'll type a simple application of the INPUT command in *HPUserEdit 4.0*.

Activate the program, request a new file (*File>New*), and double click on INPUT button in the . The cursors should now be between the two program containers. Double click on the INPUT item in the right-hand side of the interface. This will produce an input box with fields labeled "Message" and "Command Line". Type the entries as shown below:


INPUT Dialog

INPUT

Enter the parameters for this command. The image is solely for reference.

```
DEG XYZ HEX R= 'X'          PRG
{HOME}
Enter x:

{":x:" {2 0} V}
```

Message: Enter x:

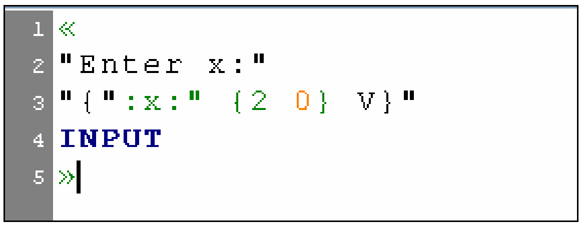
Command line: {":x:" {2 0} V}

Format

Row: 0   Column: 0   Mode: (By default)

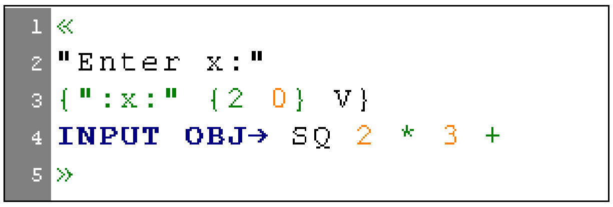
Insert   Cancel

Make sure to enter the two set of colons around the ‘x’ and to keep a space before and after the list delimiters “{“ and “}”. Press the *Insert* button to enter the command. The program area in *HPUserEdit 4.0* should now contain the lines:



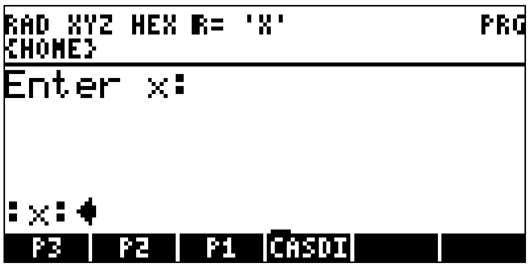
```
1 «
2 "Enter x:"
3 "{":x:" {2 0} V}"
4 INPUT
5 »
```

The outer set of double quotes in line number 3 is not required, thus, we edit those double quotes out, while also adding the additional commands show below to complete the program:



```
1 «
2 "Enter x:"
3 {":x:" {2 0} V}
4 INPUT OBJ→ SQ 2 * 3 +
5 »
```

Save the program to file *Prog4.hpe*, and load it into the emulator as a character string. Use function OBJ→ („ ° TYPE OBJ²) to convert the string to a program, and store it into variable P3. Then, press J P3 to activate the program. The emulator screen will look as follows:

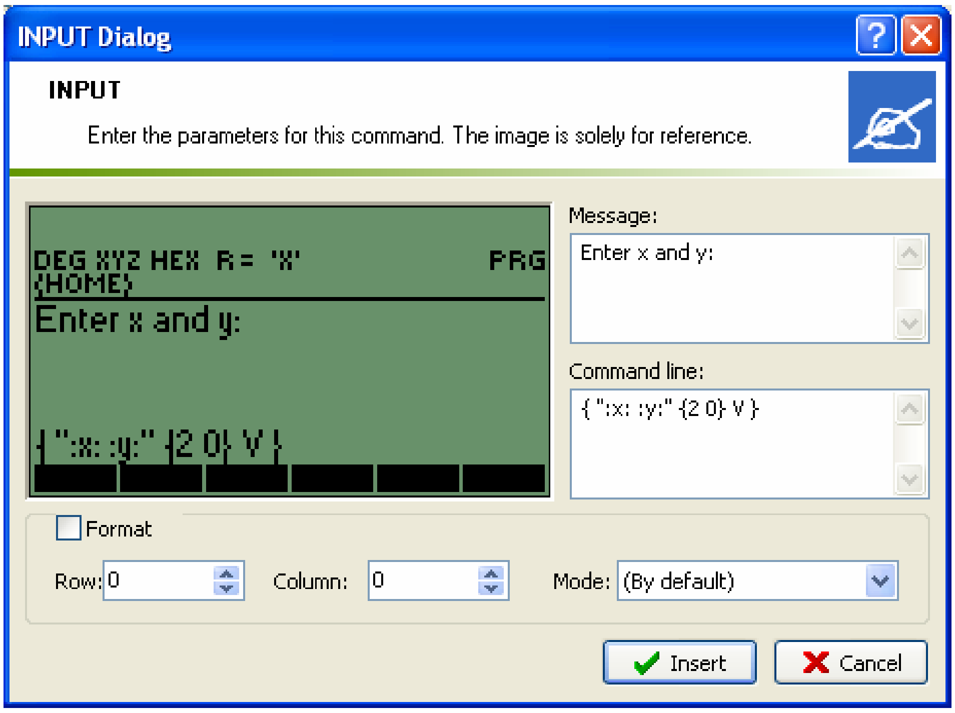


```
RAD XYZ HEX R= 'X'                PRG
{HOME}
Enter x:



:x:◆
 P3 | P2 | P1 |CASDI|     |
```

The left-facing arrow in front of the prompt `:x:` indicates the position of the cursor ready to receive a numerical input from the user. Type the number 2 and press [ENTER] to execute the program. The result is the number 11 (i.e., $2*2^2 + 3 = 2*4+3$).

Example 4 - Using the INPUT command for two inputs

In this example we use the INPUT command to allow the user to enter two input numbers. As in the previous exercise, activate *HPUserEdit 4.0*, request a new file (*File>New*), eliminate the top line, double click on the program containers item in the right-hand side of the interface, and double click on the INPUT item in the right-hand side of the interface. Type the following entries in the resulting input form as shown next:


INPUT Dialog

INPUT

Enter the parameters for this command. The image is solely for reference.

```
DEG XYZ HEX R= 'X'          PRG
{HOME}
Enter x and y:

{":x: :y:" {2 0} V }
```

Message: Enter x and y:

Command line: { ":x: :y:" {2 0} V }

☐ Format

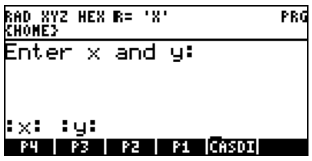
Row: 0 Column: 0 Mode: (By default)

Press the *Insert* button, edit out the first and last double quotes in line 3, and complete the program to look as follows:

```
1 «
2 "Enter x and y:"
3 "{ ":x: :y:" { 2 0 } V}"
4 INPUT OBJ→ → a b
5 «'√(a^2+b^2)' →NUM
6 »
7 »
```

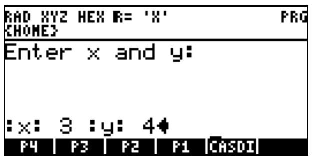
Notice that after the command OBJ→ is executed, the values of *x* and *y* are stored into *a* and *b*, respectively (→ *a b*), and a subprogram is included to calculate the value $\sqrt{a^2+b^2}$, i.e., we actually calculate the value $\sqrt{x^2+y^2}$. The subprogram consists of the commands in the inner set of program containers that end with function →NUM. The local variables a and b are used only within the subprogram and are not available once the program is executed. The result, namely, $\sqrt{x^2+y^2}$, is shown in the stack after program execution.

Save this program into file *Prog4.hpe*, emulator as a character string. Use function OBJ→ („ ° TYPE OBJ²) to convert the string to a program, and store it into variable P4. Then, press J P4 to activate the program. The emulator screen will look as follows:



```
RAD XYZ HEX R= 'X'          PRG
{HOME}
Enter x and y:


:x: :y:
 P4 | P3 | P2 | P1 |CASDI|
```

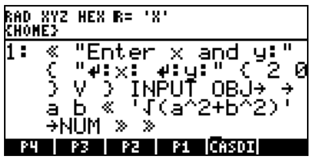
To enter the numerical values of x and y, say x = 3 and y = 4, you need to insert them after the corresponding prompts, e.g.,



```
RAD XYZ HEX R= 'X'          PRG
{HOME}
Enter x and y:


:x: 3 :y: 4◆
 P4 | P3 | P2 | P1 |CASDI|
```

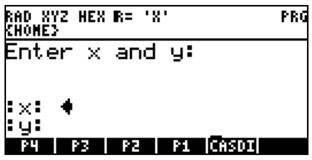
Press [ENTER] to activate the program. The result is the number 5, i.e., $\sqrt{3^2+4^2}=5$.

To make the prompt screen easier to read, we could add a couple of RETURN characters (‚ ë) before the prompts :x: and :y: in the emulator. Thus, edit the program P4 by using ‚ ë ˜ , insert the two RETURN characters and press ` . The program, in the calculator screen should now look like this:



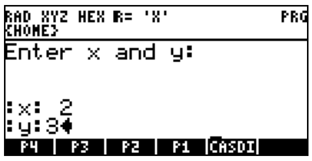
```
RAD XYZ HEX R= 'X'
{HOME}
1: « "Enter x and y:"
   { "↵:x: ↵:y:" { 2 0
   } V } INPUT OBJ→ →
   a b « '√(a^2+b^2)'
   →NUM » »
 P4 | P3 | P2 | P1 |CASDI|
```

Notice that, when you pressed ‚ ë , new lines were inserted in the screen. After you pressed ` , however, the screen shows the RETURN characters explicitly (↵). Store this new version of the program into variable P4. Execute the program once more by pressing **P4** to obtain the following screen:



```
RAD XYZ HEX R= 'X'          PRG
{HOME}
Enter x and y:

:x: ◆
:y:
 P4 | P3 | P2 | P1 |CASDI|
```

The input cursor (⬅) is now in front of the :x: prompt. Type 3 and press the down arrow key (˜) to place the input prompt cursors in front of the :y: prompt. Type 4, and press ` to execute the program. Before pressing ` the screen will look as follows:



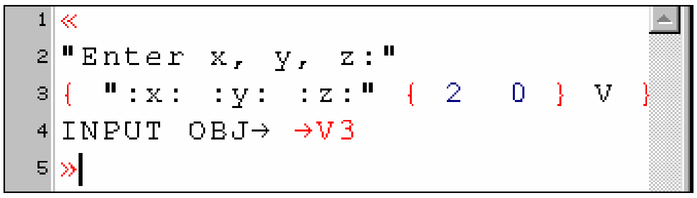
```
RAD XYZ HEX R= 'X'          PRG
{HOME}
Enter x and y:

:x: 2
:y:3◆
 P4 | P3 | P2 | P1 |CASDI|
```

The result is once more the number 5.

NOTE: Because *HPUserEdit 4.0* does not include the RETURN character (↵) as one of the special characters available, you must enter it in the emulator (or in the calculator, if skipping the emulator step) after loading the program. To enter the RETURN character, use ‚ ë .

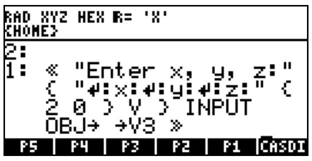
Example 5 - Using the INPUT command for three inputs

This example uses an INPUT string to enter three numerical values. The values are then used to build a vector with function →V3. Type the following program in the editor, and save it to file *Prog5.hpe*.



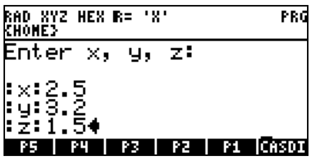
```
«
"Enter x, y, z:"
{ ":x: :y: :z:" { 2 0 } V }
INPUT OBJ→ →V3
»
```

Load *Prog5.hpe* into the emulator as a string, and convert it from a string into a program („ ° TYPE OBJ²). Add the RETURN characters (, ë) before each prompt in the input string so that the program will look like this before storing it into variable P5:



```
RAD XYZ HEX R= 'X'
{HOME}
2:
1: « "Enter x, y, z:"
   { "↵:x:↵:y:↵:z:" {
   2 0 } V } INPUT
   OBJ→ →V3 »
 P5 | P4 | P3 | P2 | P1 |CASDI
```

Store and run the program (J P5), entering the following input values:

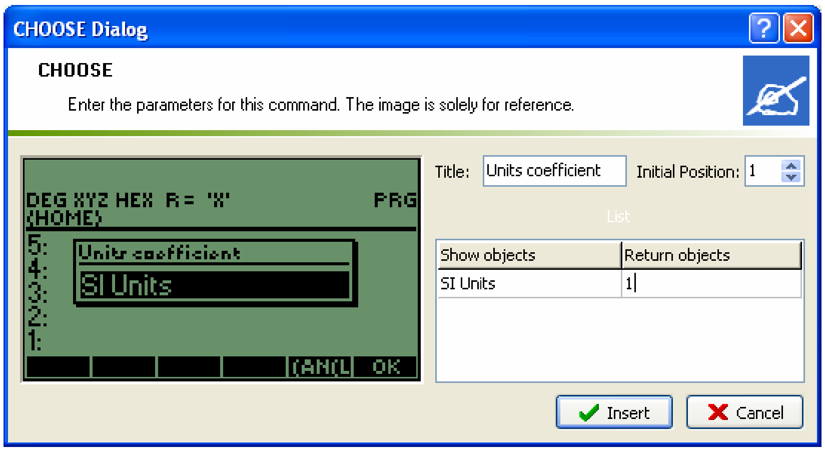


```
RAD XYZ HEX R= 'X'             PRG
{HOME}
Enter x, y, z:

:x:2.5
:y:3.2
:z:1.5◆
 P5 | P4 | P3 | P2 | P1 |CASDI
```

Press ` to execute the program. The result is the vector [2.5, 3.2, 1.5].

Example 6 - Using the CHOOSE input function

The CHOOSE input function is used to select among a list of possible values for input. In this example, we'll create a CHOOSE box with two options. First, open a new file (*File>New*), remove the top line, double click on the program container icon in the right-hand side of the interface, and then double click on the CHOOSE item in the same area. A CHOOSE input form will be generated that looks like this (with empty fields, of course):



CHOOSE Dialog

CHOOSE

Enter the parameters for this command. The image is solely for reference.

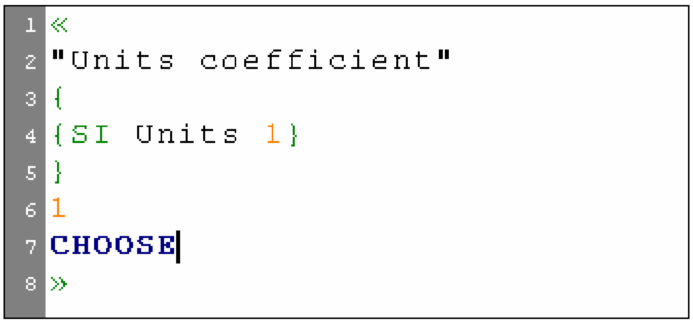
```
DEG XYZ HEX R= 'X'            PRG
{HOME}
5: Units coefficient
4: SI Units
3:
2:
1:
                   |CANCL| OK
```

Title: Units coefficient   Initial Position: 1

List

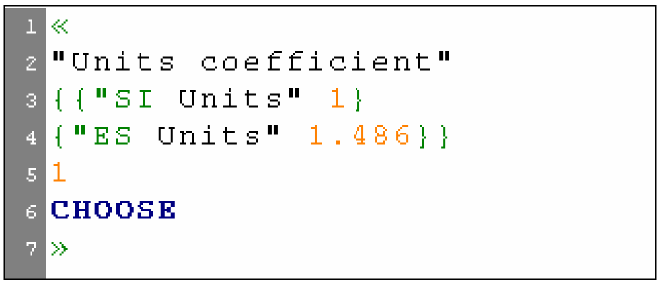
Show objects	Return objects
SI Units	1

To fill in the fields type Units coefficient in the “Title” field. Then, enter *SI Units* under the *Show objects* heading. Enter *1* under the *Return objects* heading. Ideally, this entry form should allow the entry of all options for the CHOOSE box. However, it doesn’t seem to be working properly in this version. Thus, when you press the *Insert* button, you only get the following entry:



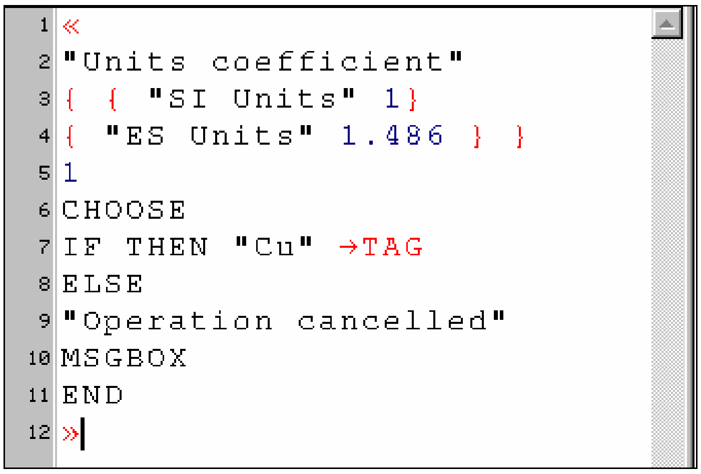
```
«
"Units coefficient"
{
{SI Units 1}
}
1
CHOOSE
»
```

Considering this limitation of the CHOOSE entry form, we need to complete and edit the command in the *HPUserEdit 4.0* window to read as follows:



```
«
"Units coefficient"
{{"SI Units" 1}
{"ES Units" 1.486}}
1
CHOOSE
»
```

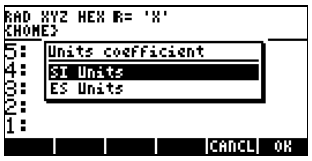
Add the additional commands shown below to the program and save it in file *Prog6.hpe*. Load the file into the emulator as a string, and convert it from a string into a program („ ° @TYPE@ @OBJ²@). Save the resulting program into variable P6.



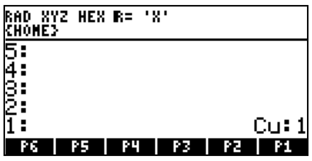
```
«
"Units coefficient"
{ { "SI Units" 1}
{ "ES Units" 1.486 } }
1
CHOOSE
IF THEN "Cu" →TAG
ELSE
"Operation cancelled"
MSGBOX
END
»
```

Run the program (J @@P6@@) to see the CHOOSE box in action. The following figures show the result of three possible outcomes:

1 - Selecting “SI Units” - Press @@OK@@

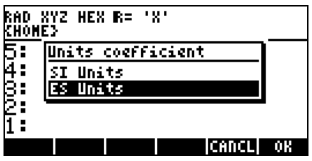


```
RAD XYZ HEX R= 'X'
{HOME}
5: Units coefficient
4: SI Units
3: ES Units
2:
1:
               CANCL  OK
```

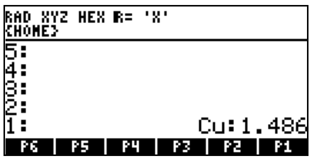


```
RAD XYZ HEX R= 'X'
{HOME}
5:
4:
3:
2:
1:                 Cu:1
 P6  P5  P4  P3  P2  P1
```

2 - Selecting “ES Units” - Press ˜ @@OK@@

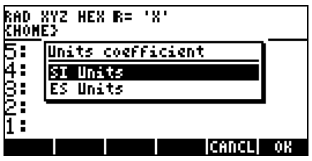


```
RAD XYZ HEX R= 'X'
{HOME}
5: Units coefficient
4: SI Units
3: ES Units
2:
1:
               CANCL  OK
```

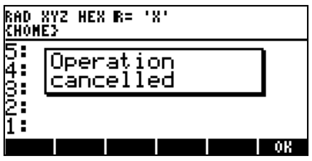


```
RAD XYZ HEX R= 'X'
{HOME}
5:
4:
3:
2:
1:             Cu:1.486
 P6  P5  P4  P3  P2  P1
```

3 - Canceling the program - Press @CANCL@



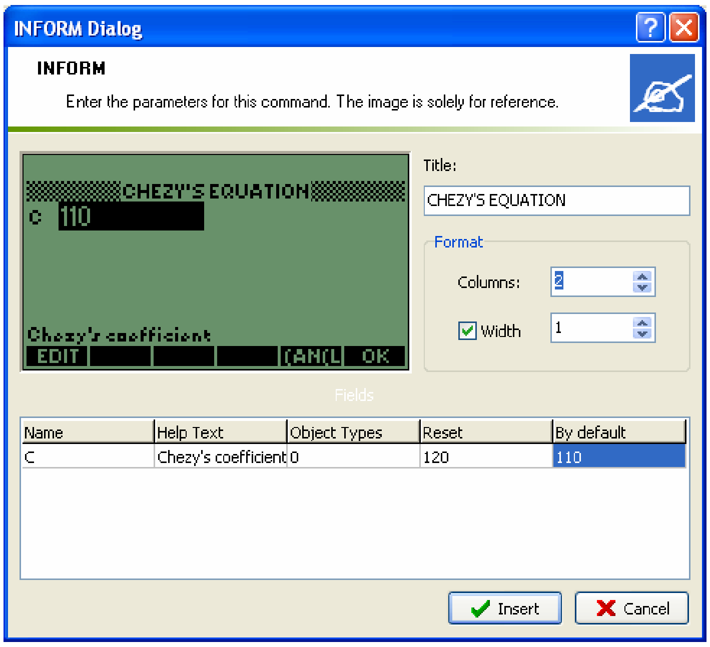
```
RAD XYZ HEX R= 'X'
{HOME}
5: Units coefficient
4: SI Units
3: ES Units
2:
1:
               CANCL  OK
```



```
RAD XYZ HEX R= 'X'
{HOME}
5: Operation
4: cancelled
3:
2:
1:
                     OK
```

Example 7 - Using the INFORM input function

The INFORM input function is used to produce a well-documented entry form. In this example we produce an input form to allow the user to enter three different values, C, R, and S, defined as “Chezy’s coefficient”, “Hydraulic Radius,” and “Channel bed slope,” respectively. The corresponding reconfiguration values will be { 120 1 0.00001}, while the default will be {110 1.5 0.0001}. The format for the INFORM command will be given by {2 1 }. The required information, for the first line of the form, is listed in the following input form in *HPUserEdit 4.0*:



INFORM Dialog

INFORM

Enter the parameters for this command. The image is solely for reference.

```
CHEZY'S EQUATION
C 110

Chezy's coefficient
EDIT              CANCL  OK
```

Title: CHEZY'S EQUATION

Format

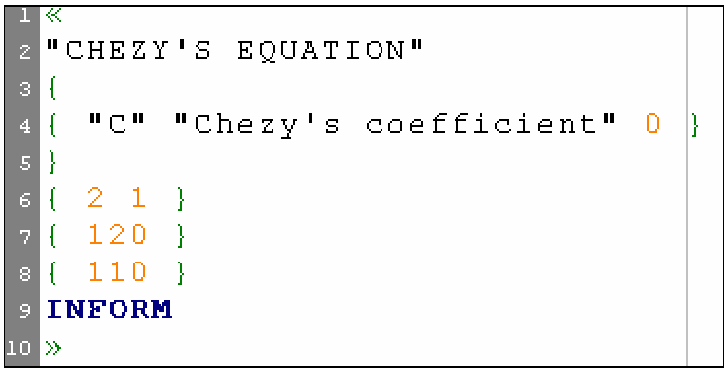
Columns: 2

☑ Width 1

Fields

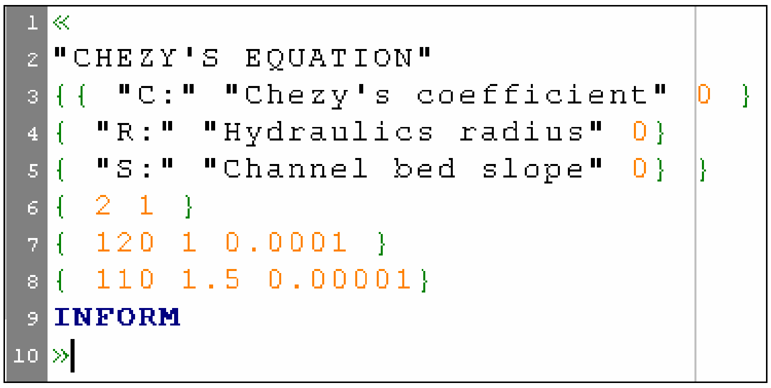
Name	Help Text	Object Types	Reset	By default
C	Chezy's coefficient	0	120	110

After pressing the *Insert* button, you get the following entry:



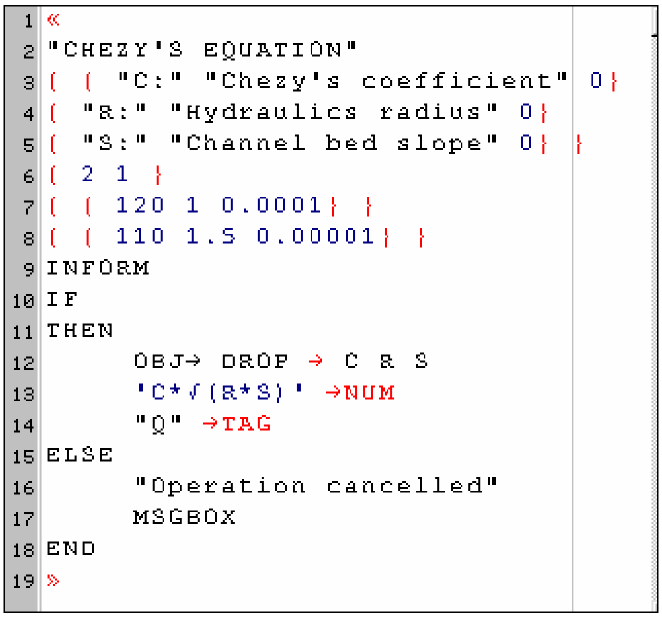
```
«
"CHEZY'S EQUATION"
{
{ "C" "Chezy's coefficient" 0 }
}
{ 2 1 }
{ 120 }
{ 110 }
INFORM
»
```

Editing the rest of the INFORM command in the editor, we get:



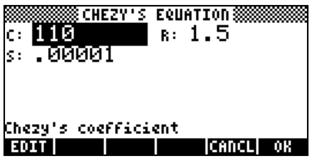
```
«
"CHEZY'S EQUATION"
{{ "C:" "Chezy's coefficient" 0 }
{ "R:" "Hydraulics radius" 0}
{ "S:" "Channel bed slope" 0} }
{ 2 1 }
{ 120 1 0.0001 }
{ 110 1.5 0.00001}
INFORM
»
```

Type in the rest of the program to read as shown in the following figure. Store the program into file *Prog6.hpe*. Load the file into the emulator as a string, and convert it from a string into a program („ ° ‚TYPE‚ ‚OBJ^2‚). Save the resulting program into variable P7.



```
«
"CHEZY'S EQUATION"
{ { "C:" "Chezy's coefficient" 0}
{ "R:" "Hydraulics radius" 0}
{ "S:" "Channel bed slope" 0} }
{ 2 1 }
{ { 120 1 0.0001} }
{ { 110 1.5 0.00001} }
INFORM
IF
THEN
      OBJ→ DROP → C R S
      'C*√(R*S)' →NUM
      "Q" →TAG
ELSE
      "Operation cancelled"
      MSGBOX
END
»
```

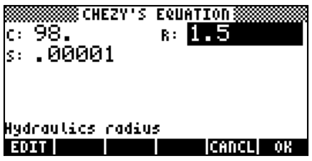
Run the program (J @@P7@@) to see the resulting INBOX:



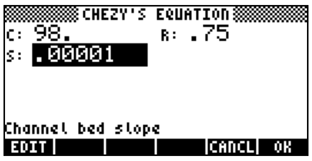
```
CHEZY'S EQUATION
C: 110          R: 1.5
S: .00001

Chezy's coefficient
EDIT |  |  |  |CANCL| OK
```

As the program gets activated, the INFORM box will show the three variables whose values are to be entered by the user. The values shown above are those in the default list, namely, {110 1.5 0.00001}. The field corresponding to C is highlighted first. At this point the user can type a new value and press @@OK@@, or simply press @@OK@@ to keep the default value. Afterwards, the value of R will be highlighted. Also, notice that the as a given variable's value is highlighted, the descriptive line at the bottom of the screen changes accordingly.

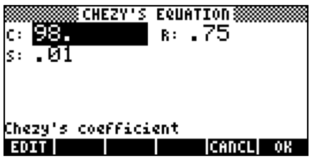
The next two figures show the highlighting of variables R and S. The values used for this exercise are as follows C = 98, R = 0.75, and S = 0.01.



```
CHEZY'S EQUATION
C: 98.          R: 1.5
S: .00001

Hydraulics radius
EDIT |  |  |  |CANCL| OK
```

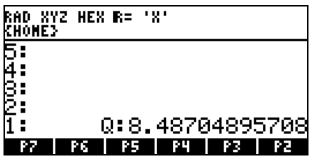


```
CHEZY'S EQUATION
C: 98.          R: .75
S: .00001

Channel bed slope
EDIT |  |  |  |CANCL| OK
```



```
CHEZY'S EQUATION
C: 98.          R: .75
S: .01

Chezy's coefficient
EDIT |  |  |  |CANCL| OK
```

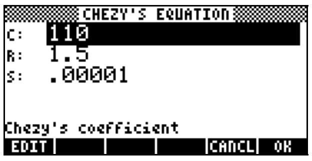
Pressing @@OK@@ one last time after highlighting and changing the value of S, the program following the INBOX command gets activated and a result is shown.



```
RAD XYZ HEX R= 'X'
{HOME}
5:
4:
3:
2:
1:        Q:8.48704895708
 P7 | P6 | P5 | P4 | P3 | P2
```

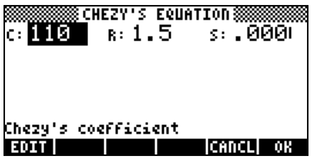
The FORMAT specification used in this case, namely { 2 1 }, produces the arrangement of input fields shown above. Other formats will produce different results as shown next:

- An empty format list, i.e., { }, or the format list {1 2}, produces



```
CHEZY'S EQUATION
C:  110
R:  1.5
S:  .00001

Chezy's coefficient
EDIT |  |  |  |CANCL| OK
```

- A format list containing {3 0} produces

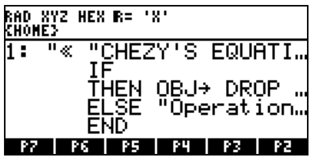


```
CHEZY'S EQUATION
C: 110   R: 1.5   S: .0001

Chezy's coefficient
EDIT |  |  |  |CANCL| OK
```

Example 8 - Exporting a program from the emulator

Suppose that you have edited a program in the emulator and would like to add additional commands in the editor (*HPUserEdit 4.0*). The simplest way to proceed is to list the program in the emulator's screen and then use the option *Edit>Save Object*. The next step is to save the object to a file. Then, move to the editor, press the *Import* button, and the contents. The program file will be listed in the editor's screen.

As an example, consider the latest version of program *P7* shown above, i.e., the one with the format list {3 0}. Listed in the emulator screen the program looks as follows (the format list is not visible, but it has been changed from the original {2 1} in *P7*):



```
RAD XYZ HEX R= 'X'
{HOME}
1: "« "CHEZY'S EQUATI…
      IF
      THEN OBJ→ DROP …
      ELSE "Operation…
      END
 P7 | P6 | P5 | P4 | P3 | P2
```

Use the option *Edit>Save Object*, and save it as *Prog8.hpe*. Move to the *HPUserEdit 4.0* editor, open a new file (*File>New*), and use press the *Import* button. The resulting program is shown next:

```
« "CHEZY'S EQUATION" { { "C:" "Chezy's coefficient" 0 } { "R:"
"Hydraulics radius" 0 } { "S:" "Channel bed slope" 0 } } { 3 0 } { 120
1 .0001 } { 110 1.5 .00001 } INFORM
  IF
  THEN OBJ→ DROP → C R S 'C*√(R*S)' →NUM "Q" →TAG
  ELSE "Operation cancelled" MSGBOX
  END
»
```

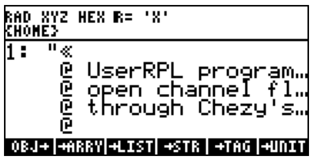
Example 9 - Using comment lines in your program

Comment lines in UserRPL start with an ampersand (@). These lines are informational only and will be ignored by the emulator or calculator. In this example we take the file exported to the editor in Example 7 and add a few comment lines as shown in the following screen:

```
«
@ UserRPL program to calculate
@ open channel flow velocity Q
@ through Chezy's equation
@
@ INFORM specifications
"CHEZY'S EQUATION"
{ { "C:" "Chezy's coefficient" 0 }
  { "R:" "Hydraulics radius" 0 }
  { "S:" "Channel bed slope" 0 } }
  { 2 1 } { 120 1 .0001 }
  { 110 1.5 .00001 }
  INFORM
@ INFORM operation
  IF    @if a 1 is received
  THEN  @decompose input and
        @calculate Q
       OBJ→ DROP
       → C R S 'C*ƒ(R*S)'
       →NUM "Q" →TAG
  ELSE
       @if a 0 is received
       @report cancel operation
       "Operation cancelled"
       MSGBOX
  END
»
```

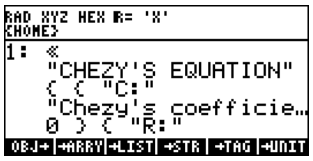
Notice that the comment lines are shown in italic format, thus making them easy to distinguish from programming statements. A line that starts with an ampersand (@) is a comment line in its entirety (i.e., no executable commands are present in that line). However, a comment can be added at the end of a command in a given line, thus producing a combined command-comment line. Anything after the ampersand in that line will be ignored during execution.

The program listed above will be stored into file *Prog7new.hpe*. Load this file into the emulator as a string. At this point the emulator screen will look as follows:



```
RAD XYZ HEX R= 'X'
{HOME}
1: "«
    @ UserRPL program…
    @ open channel fl…
    @ through Chezy's…
    @
OBJ→ →ARRY →LIST →STR →TAG →UNIT
```

Convert this string into a program by using „ ° @TYPE@ @OBJ²@. Notice that this conversion removes the comment lines from the program:



```
RAD XYZ HEX R= 'X'
{HOME}
1: «
   "CHEZY'S EQUATION"
   { { "C:"
   "Chezy's coefficie…
   0 } { "R:"
OBJ→ →ARRY →LIST →STR →TAG →UNIT
```

Save the resulting program into variable *P7n*. Press @@P7n@@ to run the program.

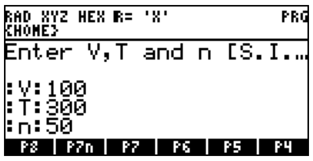
NOTE: Comment lines are only useful in files open in the editor. Once the string containing the program is transferred to the emulator and converted into a program, the comment lines will be removed.

Example 10 - Using INPUT for input and MSGBOX for output

This example, taken from the *hp 49g+ User's Guide* (Chapter 21) uses an INPUT command to enter data, and creates a long output string that is displayed in a message box (command MSGBOX). The specifications for command INPUT are listed in lines 2 through 7. Line 10 contains the command to process the input data. Lines 13 and 14 produce a list of the values of V, T, and n with attached units. Lines 17 through 22 are used to put together the output string. The value of *p* is calculated in lines 21 and 22. Command MSGBOX, in line 23 produces the output. Lines 24 through 26 show the three levels of subprograms used in this program. The listing shown below includes comment lines that are not contained in the *User's Guide* listing.

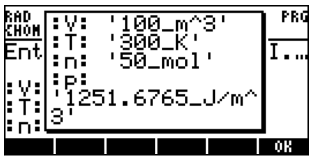
```
1.  «
2.  @ INPUT specifications
3.  "Enter V,T and n [S.I.]:"
4.  { "
5.  :V:
6.  :T:
7.  :n:" { 2. 0. } V }
8.  INPUT
9.  @ Processing of input data
10. OBJ→ → V T n
11. @ Subprogram to add units to
12. @ variables
13.    « V '1_m^3' * { } + T '1_K'
14.    + n '1_mol' * + EVAL → V T n
15. @ Subprogram to create output
16. @ string
17.       « V "V" →TAG →STR "
18.     " + T "T" →TAG →STR "
19.     " + n "n" →TAG →STR "
20.     " +
21.     '(8.34451_J/(K*mol))*(n*T/V)'
22.      EVAL "p" →TAG →STR + + +
23.      MSGBOX
24.       »
25.    »
26. »
```

Once the program is transferred to the emulator as a string, converted to a program, stored (say in *P8*) and executed, the following input string will be used for input:



```
RAD XYZ HEX R= 'X'          PRG
{HOME}
Enter V,T and n [S.I.…

:V:100
:T:300
:n:50
```

After pressing ` , the result is show in the following screen:


```
:V: '100_m^3'
:T: '300_K'
:n: '50_mol'
:p:
'1251.6765_J/m^
3'
```

Press **OK** to exit the program.

Example 11 - Using list operations for discrete probability distributions - modular programming

A discrete probability distribution is typically defined by a *probability mass function (or pmf)* given as a table of values (x_i, f_i), for *i = 1, 2, …, n*. For the development of the program in this example we will enter the values of *x* and *y* as lists of values, say:

x = { 1, 2, 3, 4, 5}
f = { 0.1 0.2 0.3 0.1 0.3}

The lists x and f will be stored in variables called **x** and **f**, respectively. The program will perform the following tasks:

1. Check that $\sum_{i=1}^{n} f_i = 1$. If it is true, continue with step 2, otherwise stop procedure and report result.
2. Calculate the mean of the distribution, i.e., $\mu = \sum_{i=1}^{n} x_i \cdot f_i$.
3. Calculate the variance and the standard deviation of the distribution, i.e., $Var(X) = \sum_{i=1}^{n} (x_i - \mu)^2 \cdot f_i$, and $\sigma = \sqrt{Var(X)}$.
4. Calculate the skewness of the distribution, i.e., $\alpha_3 = \frac{1}{\sigma^3} \sum_{i=1}^{n} (x_i - \mu)^3 \cdot f_i$.
5. Calculate the skewness of the distribution, i.e., $\alpha_4 = \frac{1}{\sigma^4} \sum_{i=1}^{n} (x_i - \mu)^4 \cdot f_i$.

The first thing to notice is that the calculation of variance (step 4), skewness (step 5), and kurtosis (step 6), involve similar summations. Thus, we could develop a program called SUMN to calculate the summations involved in those steps. The following program can be used for that purpose:

```
« MUCALC → n mu
« x mu - n ^ f * ΣLIST
»
»
```

The program SUMN, listed above, calls another program MUCALC, used to calculate the mean value, which is listed below:

```
« CHKF
IF
THEN x f * ΣLIST
END
»
```

The following program is used to check if the sum of the values in *f* adds to 1.0. Instead of seeking an exact equality we will check instead that $\left|\left(\sum_{i=1}^{n} f_i\right) - 1\right| \leq \varepsilon$, where ε is a small quantity, say ε = *0.00001*. The value of ε will be stored in variable @@@‾@@@.The program will be called CHKF:

```
«f ΣLIST 1 - ABS ε >
IF
THEN
    "f is not a pmf" MSGBOX
    0
ELSE
    1 "Σf" →TAG MSGBOX
END»
```

Notice that if $\left|\left(\sum_{i=1}^{n} f_i\right) - 1\right| > \varepsilon$, the program reports that list *f* is not a probability mass function (pmf) and returns a zero, otherwise it returns a one.

The functions VARX, STDEV, SKEW, and KURT calculate, respectively, the variance, standard deviation, skewness and kurtosis of the distribution. The corresponding programs are listed next:

VARX:

```
« CHKF
IF
THEN 2 SUMN
END
"Var(X)" →TAG
»
```

STDEV:

```
«VARX √ "σ" →TAG
»
```

SKEW:

```
«CHKF
IF
THEN 3 SUMN STDEV 3 ^ /
END
"α3" →TAG
»
```

KURT:

```
«CHKF
IF
THEN 4 SUMN STDEV 4 ^ /
END
"α4" →TAG
»
```

NOTE: Since we are dealing with a number of small programs, instead of saving the programs in files from within the editor and then loading those files in the emulator, we can simply copy the programs in the editor (*Edit>Select All* followed by *Edit>Copy*) and paste them in the emulator (*Edit>Paste String*). The strings thus pasted can be converted into programs with function OBJ→ („ ° @TYPE@ @OBJ²@), and saved in the emulator for debugging and testing.

To enter data we can use the following program entitled DATAIN:

```
«
"Discrete prob. distribution"
{ { "x:" "values of random variable" 5}
{ "f:" "values of probability" 5}
{ "ε:" "tolerance" 0 } }
{ 3 0 }
{ { 1 2 3 4 5 } { 0.2 0.2 0.2 0.2 0.2 } 0.0001 }
{ { 1 2 3 4 5 } { 0.2 0.2 0.2 0.2 0.2 } 0.0001 }
INFORM
IF
THEN
    OBJ→ DROP
    'ε' STO 'f' STO 'x' STO
    "Data stored" MSGBOX
ELSE
    "Operation cancelled"
    MSGBOX
END
»
```

Finally, to control the entire operation we create the following program called DPCALC (which stands for Discrete Probability CALCulation):

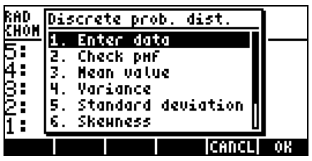
```
«"Discrete probability dist."
{ { "1. Enter data" DATAIN}
  { "2. Check pmf" CHECKF }
  { "3. Mean value" MUCALC }
  { "4. Variance" VARX}
  { "5. Standard deviation" STDEV}
  { "6. Skewness" SKEW}
  { "7. Kurtosis" KURT}
  { "8. Exit" } }
1
CHOOSE
IF
THEN EVAL
ELSE "Operation cancelled" MSGBOX
END
»
```

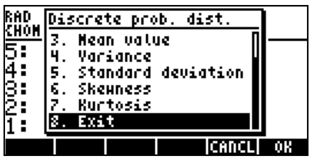
Thus, you should have loaded in your emulator nine programs, namely:

• DPCALC:	main program
• DATAIN:	data input routine
• CHKF:	routine to check property that sum of probabilities is equal to 1
• MUCALC:	routine to calculate the mean value
• SUMN:	routine to calculate the summation needed for VARX, SKEW, KURT

- VARX: routine to calculate the variance
- STDEV: routine to calculate the standard deviation
- SKEW: routine to calculate the skewness
- KURT: routine to calculate the kurtosis

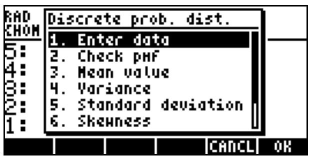
To facilitate the calculation, it is recommended that you copy all your programs into a sub-directory, say, one called DPDIST (for Discrete Probability DISTribution). The main program is DPCALC. When you launch this program the following CHOOSE box is provided:

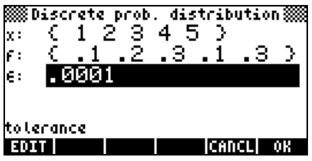




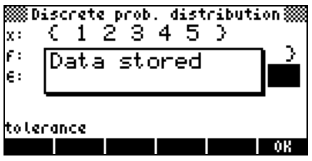
You can use the vertical arrow keys to highlight the program of interest, or simply press the corresponding number in the keyboard. After highlighting a selection, press OK to execute that program. The following screen shots show the different programs in action:

- Option 1 - Enter data (DATAIN)



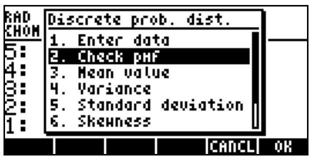


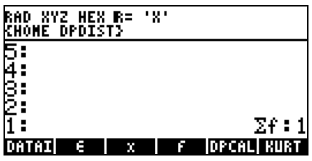
```
Discrete prob. distribution
x: { 1 2 3 4 5 }
f: { .1 .2 .3 .1 .3 }
ε: .0001
tolerance
```



```
Discrete prob. distribution
x: { 1 2 3 4 5 }
f: Data stored }
ε:
tolerance
```

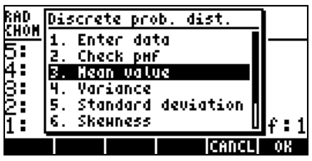
- Option 2 - Check pmf (CHKF)

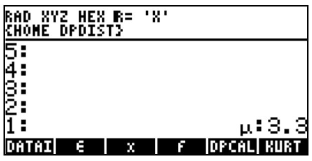




```
RAD XYZ HEX R= 'X'
{HOME DPDIST}
5:
4:
3:
2:
1:                Σf:1
```

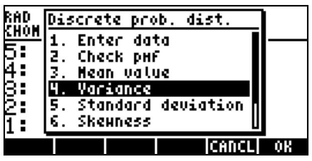
- Option 3 - Mean value (MUCALC)

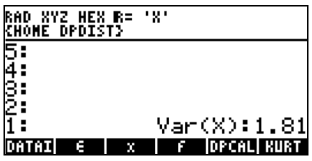




```
RAD XYZ HEX R= 'X'
{HOME DPDIST}
5:
4:
3:
2:
1:              μ:3.3
```

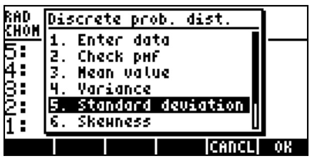
- Option 4 - Variance (VARX)



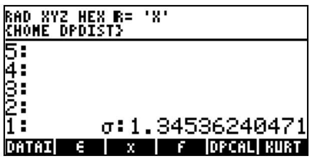


```
RAD XYZ HEX R= 'X'
{HOME DPDIST}
5:
4:
3:
2:
1:         Var(X):1.81
```

- Option 5 - Standard deviation (STDEV)

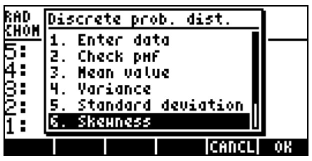


```
RAD  Discrete prob. dist.
{HOM 1. Enter data
5:   2. Check pmf
4:   3. Mean value
3:   4. Variance
2:   5. Standard deviation
1:   6. Skewness
                    CANCL  OK
```

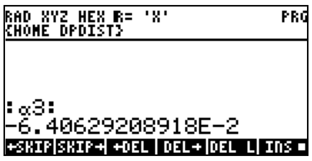


```
RAD XYZ HEX R= 'X'
{HOME DPDIST}
5:
4:
3:
2:
1:        σ:1.34536240471
DATA1  ∈   x   f  DPCAL KURT
```

- Option 6 - Skewness (SKEW)

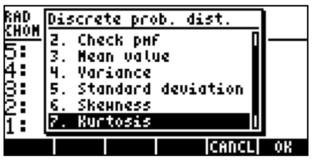


```
RAD  Discrete prob. dist.
{HOM 1. Enter data
5:   2. Check pmf
4:   3. Mean value
3:   4. Variance
2:   5. Standard deviation
1:   6. Skewness
                    CANCL  OK
```

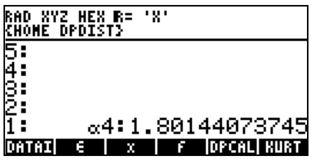


```
RAD XYZ HEX R= 'X'            PRG
{HOME DPDIST}


:α3:
-6.40629208918E-2
←SKIP SKIP→ ←DEL DEL→ DEL L INS ■
```

- Option 7 - Kurtosis (KURT)

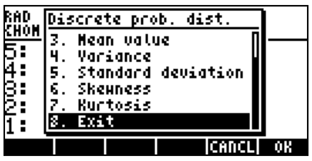


```
RAD  Discrete prob. dist.
{HOM 2. Check pmf
5:   3. Mean value
4:   4. Variance
3:   5. Standard deviation
2:   6. Skewness
1:   7. Kurtosis
                    CANCL  OK
```

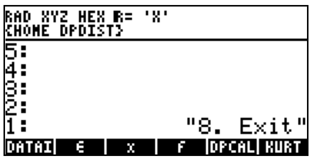


```
RAD XYZ HEX R= 'X'
{HOME DPDIST}
5:
4:
3:
2:
1:      α4:1.80144073745
DATA1  ∈   x   f  DPCAL KURT
```

- Option 8 - Exit (no action)

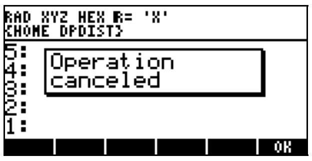


```
RAD  Discrete prob. dist.
{HOM 3. Mean value
5:   4. Variance
4:   5. Standard deviation
3:   6. Skewness
2:   7. Kurtosis
1:   8. Exit
                    CANCL  OK
```



```
RAD XYZ HEX R= 'X'
{HOME DPDIST}
5:
4:
3:
2:
1:                 "8. Exit"
DATA1  ∈   x   f  DPCAL KURT
```

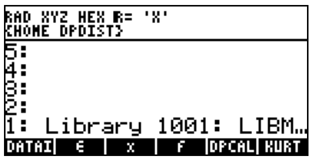
- Cancel operation - when you press **CANCL**, the result is this screen:



```
RAD XYZ HEX R= 'X'
{HOME DPDIST}
5: Operation
4: canceled
3:
2:
1:
                         OK
```

Example 12 - Creating a library out of your UserRPL programs

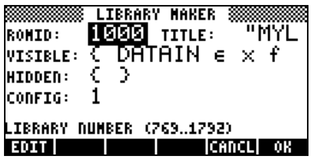
A library is a binary program that is available for use from any directory in your memory. The easiest way to produce a library is by using the library called LIBRARY MAKER available under the name *libmaker.zip* in http://www.hpcalc.org. The zip file for LIBRARY MAKER contains three files: *lbmkr.lib*, *lbmkr.src*, and *lbmkr.txt*. Extract those files to a folder of your choosing. Then load *lbmrk.lib* onto the emulator (or calculator), which will produce the following screen:



```
RAD XYZ HEX R= 'X'
{HOME DPDIST}
5:
4:
3:
2:
1: Library 1001: LIBM…
DATA1  ∈   x   f  DPCAL KURT
```

The library contents are listed in stack level 1. At this point it is necessary to store this contents into port 0 by typing the number 0 and pressing K . To load the library into memory it is necessary to press the keys $ and C , simultaneously. This will re-start the emulator (or calculator) and make the library available for use.

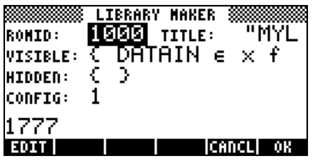
The exercise proposed herein will create a library out of the collection of programs defined in the previous example (Example 11). We assume that all the programs are gathered in a sub-directory. From within that sub-directory, press , á , and then press @)LIBMA and @LBMKR@. These actions produce the following screen:

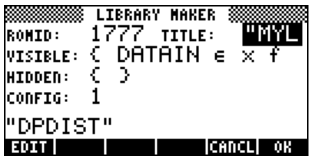

```
LIBRARY MAKER
ROMID:  1000  TITLE:  "MYL
VISIBLE: { DATAIN ∈ x f
HIDDEN:  { }
CONFIG:  1
LIBRARY NUMBER (769..1792)
EDIT |   |   |   |CANCL| OK
```

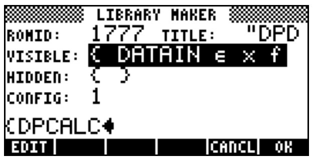
The form contain the following fields:

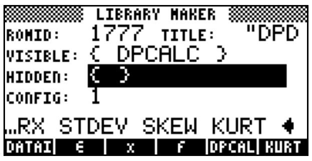
- ROMID: an integer number between 769 and 1792 that identifies the library about to be created (for example, library LBMKR has ROMID = 1001, as shown above).
- TITLE: a string that identifies the library.
- VISIBLE: the list of the programs or variables that will be visible to the user.
- HIDDEN: the list of the programs or variables that will be hidden from the user.
- CONFIG: use the default value of 1

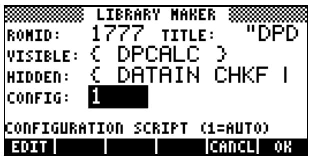
For the present case we will use: ROMID = 1777, TITLE = "DPDIST", VISIBLE = {DPCALC}, HIDDEN = {DATAIN, CHKF, MUCALC, SUMN, VARX, STDEV, SKEW, KURT}, and CONFIG = 1. Thus, the input form for our case will show the following options:


```
LIBRARY MAKER
ROMID:  1000  TITLE:  "MYL
VISIBLE: { DATAIN ∈ x f
HIDDEN:  { }
CONFIG:  1
1777
EDIT |   |   |   |CANCL| OK
```

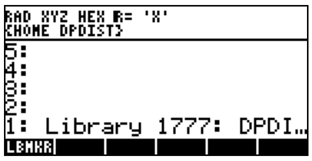

```
LIBRARY MAKER
ROMID:  1777  TITLE:  "MYL
VISIBLE: { DATAIN ∈ x f
HIDDEN:  { }
CONFIG:  1
"DPDIST"
EDIT |   |   |   |CANCL| OK
```


```
LIBRARY MAKER
ROMID:  1777  TITLE:  "DPD
VISIBLE: { DATAIN ∈ x f
HIDDEN:  { }
CONFIG:  1
{DPCALC◆
EDIT |   |   |   |CANCL| OK
```

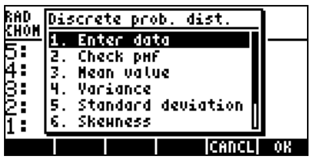

```
LIBRARY MAKER
ROMID:  1777  TITLE:  "DPD
VISIBLE: { DPCALC }
HIDDEN:  { }
CONFIG:  1
…RX STDEV SKEW KURT ◆
DATAI| ∈ | x | f |DPCAL|KURT
```


```
LIBRARY MAKER
ROMID:  1777  TITLE:  "DPD
VISIBLE: { DPCALC }
HIDDEN:  { DATAIN CHKF I
CONFIG:  1
CONFIGURATION SCRIPT (1=AUTO)
EDIT |   |   |   |CANCL| OK
```

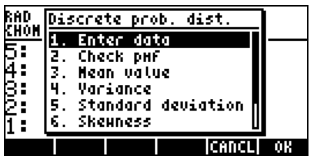
After entering all the information in the input form, press @@OK@@ to obtain the following screen:

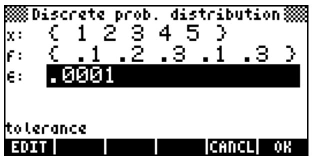

```
RAD XYZ HEX R= 'X'
{HOME DPDIST}
5:
4:
3:
2:
1: Library 1777: DPDI…
LBMKR|   |   |   |   |
```

The library is now ready to be stored in port 0 (0 K) and loaded into memory it is ($ and C , simultaneously). After the emulator re-start we can press , á , and then press @)PDIS and @DPCAL@ to activate the library. Before we do that, however, let's move to the HOME directory by pressing „ § , simultaneously (to hold down the „ key in your emulator press it using the right mouse button and then press §). Now, press , á PDIS and DPCAL. The result is the following:

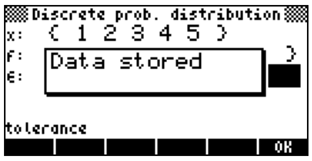


which is what we programmed in the previous example. Thus, let's repeat the entries in Example 11 to verify that the library program is performing the expected tasks:

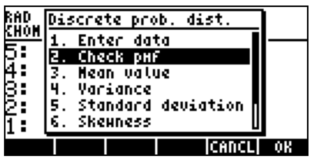


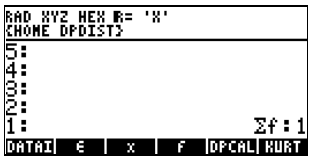


```
Discrete prob. distribution
x: { 1 2 3 4 5 }
f: { .1 .2 .3 .1 .3 }
ε: .0001
tolerance
```

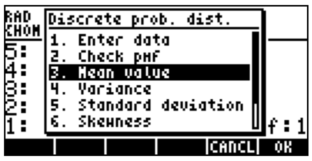


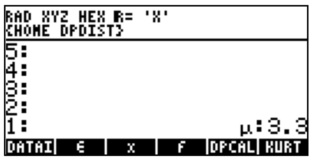
```
Discrete prob. distribution
x: { 1 2 3 4 5 }
f: Data stored
ε:
tolerance
```



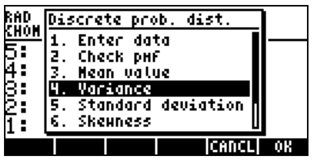


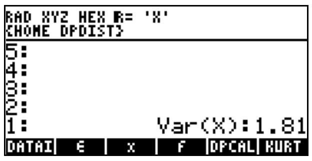
```
1:                 Σf:1
```



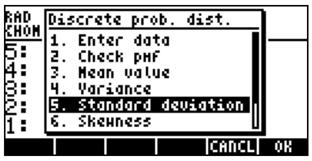


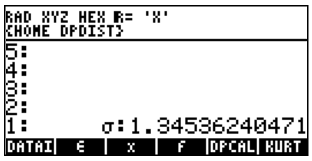
```
1:                 μ:3.3
```



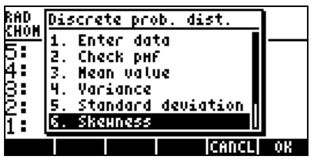


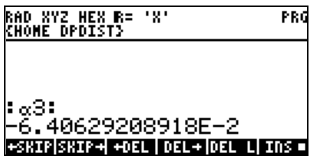
```
1:           Var(X):1.81
```



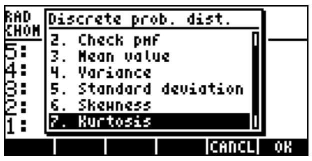


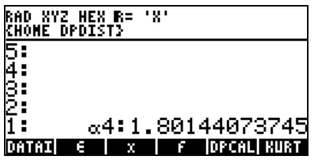
```
1:      σ:1.34536240471
```



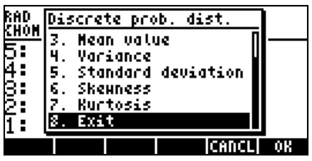


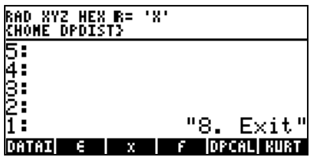
```
:α3:
-6.40629208918E-2
```



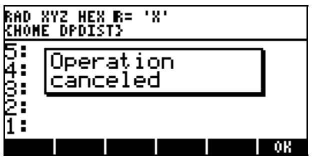


```
1:      α4:1.80144073745
```

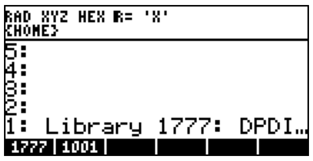




Finally, when you press @CANCL, the result is this screen:



Having verified that the library works properly, we need to export it from the emulator by storing it into a variable. To get it listed in the screen use , á @)@0@@ and , @1777@. The resulting screen is:



To store it into variable DPD, type 'DPD' and press K . Press J to verify the existence of variable @@DPD@ in your list of variables. Press @@DPD@ to list the contents of the variable in the screen once more, and then use the option *Edit>Save Object…* to save the library to a computer file, say *DPD*. This file can then be transferred to your calculator in order to store the library into port 0 (or port 1 or port 2) and make it available for use in your calculator.

SUMMARY

In conclusion, in this document we presented several exercises to demonstrate the use of the editor HPUserEdit 4.0 and the emulator Emu48 (with an hp 49 ROM) to develop UserRPL programs. Also, we demonstrated the use of the library LIBRARY MAKER to convert a directory of UserRPL programs into an hp 49 library.

For additional information visit the web page:

http://www.engineering.usu.edu/cee/faculty/gurro/Software_Calculators/Calculators.htm

This document was prepared by Gilberto E. Urroz, Ph.D., P.E., Associate Professor, Department of Civil and Environmental Engineering and Utah Water Research Laboratory, Utah State University, Logan, Utah, March 12, 2006.