TN 50C

EduCALC TECHNICAL NOTES

27953 CABOT ROAD LAGUNA NIGUEL, CA 92677

INPUT - By Example

A program has many parts. A part common to most sophicated programs is prompting the user for the inputs required for the program to do its job. Simple programs may start assuming that the proper information is on the stack when the program is executed. An example is a program called 'M↑YR'. This is a calendar program that displays a calendar assuming that the month (number 1 thru 12) is on level two and the year (1993 for example) is on level one. This is the ultimate in simplicity - once you know the program and have run it a couple of times. The name of the program also contains the instructions on how to run it. The abbreviation YR identifies it in you mind. The letter M is associated with year as month. Most programs do not lend themselves to such simplicity. What you will normally use is a prompting display that also sets up the calculator to receive the inputs. You will also want to provide simple instructions to the user as to what to do. All of these things are taken care of by the INPUT command. This five letter command does so much it requires a few examples to illustrate.

In a program INPUT looks like this:

```
      << . . . "prompt-string" "command-line" INPUT  . . . >>
```

This is one of two variations. Let's explore this two string version first.

The INPUT command causes the program to stop running. It displays the the prompt-string and starts the command line with the command-line string. The user is expected to respond by pressing one or more keys followed by ENTER. Since the command line is activated by INPUT any keys that you normally press apply. Pressing TAN will place the function in the command line, pressing ON will clear the command line. Pressing ENTER will terminate the command line and place the command-string PLUS your keystrokes on level one AS A TEXT STRING. This string is called the result string. Let's see how this works with a simple program.

INPUT may be entered in a program by pressing: SX - PRG, CTRL, NXT; GX - PRG, NXT, IN.

1. DISPLAY THE INPUT PROMPT-STRING AS A SIMPLE MESSAGE.

```
'IN1'  << "Hello, How are you?" "" INPUT >>
           41.5 Bytes  # E968h
```

If you press the menu key IN1 you will see the "Hello . . . " message displayed on line three of the display. This is the first line of the normal display area. This area starts just below the line that seperates the message (status) area from the stack area. The command-line string is empty in this example.

You will notice that PRG shows in the upper right corner of the diaplay. This tells you that the program is temporarily halted but still running - and in a higher current use mode that is draining your batteries.

Also notice the blinking command line cursor in the lower left corner of the display just above the menu line. Run the program. What happens if you just press ENTER, press 77, press TAN, or press << SWAP >> ?

2. DISPLAY THE LONGEST ONE LINE MESSAGE POSSIBLE USING INPUT.

```
'IN2'   << "THIS LINE IS 27 CHARACTERS." "" INPUT >>
           49.5 Bytes  # 4256h
```

The display will show one line with three little dots at the end to tell you that there are more characters that can't be seen. If CHARACTERS. is shortened to CHARS. the line will just fit on the screen. The display is 22 characters across and that is the longest message that will fit ON ONE LINE. Change the 27 to 22 and delete CTER and 'IN2' will meet the requirement of example 2.

```
'IN2B'   << "THIS LINE IS 22 CHARS." "" INPUT >>
            44.5 Bytes  # 34DAh
```

3. DISPLAY THE LONGEST MESSAGE POSSIBLE USING INPUT.

```
'IN3'   <<  "Happy Birthday William┘from your Mom and Dad┘with
            our fondest love." "" INPUT >>
            89.5 Bytes  # B3DFh
```

The ┘ in the text is a new line character placed in the text in the 22nd character position. Up to three lines may be in the prompt-string of the INPUT command. The new line character is entered by pressing RIGHT SHIFT, DECIMAL POINT.

In order to display the message properly with minimum display confusion the ENTER key must be pressed. If ON is pressed the machine will show a partial Happy Birthday Wi... on level two and an empty text string on level one. This is not the ideal way for William to get the message. The ideal way would be to name the program 'WIL', and drop the empty text string after reading the message. To inform the new user to press ENTER we may use the prompt-string to tell him what to do - "PRESS ENTER TO CLEAR".

4. DISPLAY A NICE HAPPY BIRTHDAY MESSAGE FOR WILLIAM.

```
'WIL'   <<"Happy Birthday William┘from your Mom and Dad┘with our
          fondest love." "PRESS ENTER TO CLEAR" INPUT DROP >>
          112 Bytes  # 1D23h
```

You will notice that two text sizes are used. The command-string font is two dots higher than the prompt-string font - 5X7 Vs 5X9.

Thus far we have only used the prompting aspect of INPUT in a simple task of displaying messages rather than inputting data for a program. The keyed input is placed on level one as a string object. A more common use of inputs is numbers. Suppose we are writing a program to compute the area of a circle using the radius as an input.

4. WRITE A PROGRAM TO COMPUTE THE AREA OF A CIRCLE PROMPTING FOR RADIUS AS AN INPUT.

```
'CIR'  << "   Area of a Circle┘  Key Radius, ENTER" "" INPUT OBJ->
         SQ PI * >>
         73.5 Bytes  # FE6Fh
```

This example includes the instructions of what to do in the prompt-string. The command-line string has not been used. Our next example uses the command-line string.

5. CALCULATE THE FREQUENCY ERROR IN PARTS PER MILLION FOR A MEASURED CLOCK CRYSTAL.

The desired frequency is 32,768 hertz. Each crystal is measured and the frequency is compared with 32,768 to calculate the error. All measured crystals are within a few hertz of 32,768. We will use the command-line to enter 3276 for us so we don't have to key it each time.

```
'IN5'  << "┘   Input Frequency" "3276" INPUT OBJ-> 32768 SWAP OVER -
         SWAP / 1E6 * 2 RND >>        89 Bytes  # D02h
```

The new line character spaces the prompt one line down, the leading three spaces centers it. When the program is run the command line will show 3276 with a blinking cursor. You continue the number and press ENTER. Pressing 7 will enter 32767 which is one hertz low. This calculates to -30.52 parts per million low in frequency.

New line characters may also be used in the command-line string. The display area, however, may only show four lines. Our 'WIL' program used three for the prompt-string, a blank line due to the format of the INPUT command, and a one line command-line. If new line characters are used in the command-line the machine will "over write" the lines above from the prompt-string and they won't be displayed. The total lines that will be displayed from the two strings is four.

An exmple using this feature is a program that uses the command line for description and quantity for taking inventory. Move the cursor up and down and fill in the quantity.

The prompt-line is suitable for a few limited applications, but there are situations where more than completing the command line is desirable. For these applications the second variation of the INPUT command is used. This variation, instead of using two strings, uses a string and a list. In a program, this expanded variation looks like this:

```
<< . . . "prompt-string" {command-line} INPUT OBJ-> . . . >>
```

The command line list has one or more of the following components - in any order.

```
{ "command-line" cursor-position operating-options }
```

"command-line"	Optional. Behaves as described above.
cursor-position	Optional. May be a single number or a list of two {row-character} numbers. A single number positions the blinking cursor in the command line at that character position. The position is from the beginning of the string. A zero places the cursor at the end of the command-line string even if the new line character is used. If the number is positive the insert cursor is used, if negative the replace (over write) cursor is used. This is independent of how you have set your cursor. A cursor-position list specifies the row and character position. This is used when the new line character is used to display a multiline command-line string. A zero specifies the end of the row. A positive character position number specifies the insert cursor; a negative position number, the replace cursor.
operating-options	Optional, use zero or more of these unquoted names in any order. ALG - sets algebraic entry mode. Saves pressing the single tic key. å - alpha lock. V - varifies whether the string (without " " delimiters) is a valid object or sequence of objects. If invalid the Invalid syntax message is diaplayed and the prompt is repeated.

<u>6. USE QUAD TO SOLVE FOR THE ROOTS OF A QUADRATIC EQUATION.</u>

```
'quad'  << "Enter Expression in X" {"''" 2 å ALG} INPUT OBJ-> DUP -1
           CF 'X' QUAD DUP 1 's1' STO EVAL SWAP -1 's1' STO EVAL 's1'
           PURGE >>      140 Bytes  # A2EDh
```

The program is stored using lower case to avoid conflict with the reserved variable QUAD. An input 'X^2 - X - 6' => 'X = 3', 'X = -2'.

The program requires an algebraic input. The algebraic delimiters are input as a text string. The cursor position is

Note: å is HP 48 Greek letter Alpha: RIGHT SHIFT, A in alpha mode.

moved to the seccond position so the first key pressed is following the algebriac delimiter. Alpha mode is desired when keying alpha characters in an algebraic. The program is straight forward and may be followed after looking up QUAD in your 48S/SX Owners Manual or G Series User's Guide. Next is the shortest (not fancy) program to solve a quadratic.

```
'QD'     << 3 PICK / SWAP ROT -2 * / DUP SQ ROT - √ + LASTARG - >>
            Place coefficents on stack, example:  1, -1, -6  'QD'
```

7. PRELOAD A CHOICE THAT IS USED OFTEN. PURGE FILE? Y OR N?

The cursor position may be specified with the replace cursor to make a yes no response simpler. Key strokes are saved by setting alpha on.

```
'IN7'  << "┘      Purge File?" { "@ YorN @  N" -11 å } INPUT OBJ-> IF 'N'
          SAME THEN "File Saved" ELSE "File Purged" END >>
            126 Bytes  # CB7Eh
```

The program produces 'N' or 'Y' which is tested by the IF structure. If 'N', the text string "File Saved" is placed on level one. If 'Y', "File Purged" is placed on level one. The @ symbol brackets the YorN instruction part of the command-line string. OBJ-> strips away the @ symbols and the characters inbetween. This program does not provide a means of repeating the command-line if something other than an N or Y is entered.

8. REWRITE 'IN7' TO REPEAT THE QUESTION FOR ANYTHING EXCEPT Y OR N.

```
'IN8'  << "┘     Purge File?" { "@ YorN @  N" -11 å } INPUT OBJ-> DUP
          { N Y } SWAP POS IF THEN "File " SWAP IF 'N' SAME THEN
          "Saved" ELSE "Purged" END + ELSE IN8 END >>
            174 Bytes  # 4D22h
```

This version of the program uses the input to POS from the N Y list. If N, POS returns a 1 (non-zero), if Y POS returns a 2, (again non-zero). The first IF tests for non-zero. A 1 or 2 will cause the following THEN to be executed. This places the common word "File " on level one of the stack. A second IF checks for the input letter (saved by the DUP following OBJ->) to see if it is an N. If true (non-zero) place "Saved" on the stack. ELSE put "Purged" on the stack. The + after the END adds the two strings. If POS does not find an N or Y, a zero is placed on the stack. The first IF tests this and does the last ELSE which runs the program until the input is N or Y.

There is a program flaw. Can you see what it is? Run the program with various inputs and you will easily spot the problem. This example illustrates the difficulty of getting exactly what you want in the display, setting the machine to the desired modes for convenient (fewest keystrokes) input, and allowing for EVERYTHING the user may do.

9. REWRITE 'IN8' TO MEET ALL INPUT POSSIBILITIES.

Note: å is HP 48 Greek alpha character: RIGHT SHIFT, A in alpha mode.

```
'IN9'  << 0 WHILE DROP "┘      Purge File?"  { "@ YorN @  N" å V } INPUT
          OBJ-> DUP { N Y } SWAP POS 0 SAME REPEAT END "File " SWAP IF
          'N' SAME THEN "Saved" ELSE "Purged" END + >>
           178.5 Bytes  # B1D1h
```

This version uses a WHILE ... UNTIL ...END loop structure to repeat the INPUT command until either an N or a Y is keyed. Any other inputs (in alpha mode) cause INPUT to be repeated. Testing to produce the correct out put text string is the same as the previous programs. If a letter A is keyed the previous programs would enter the letter onto the stack. The WHILE loop-clause should remove this unacceptable input. The DROP at the beginning does this. The stack may be clear and this DROP will cause an error so the initial zero provides a "dummy" object to drop. Is this the best way to "fix" the problem?

CONCLUSION

Programs that provide a quality user interface that allows for every possible situation are difficult to write. This brief handout gives examples of the the essential elements of the INPUT acommand. Refer to your owners manual for additional details.

Often a complex user interface is called an "engine". It is a "front end" program that presents menus, dialog boxes, or multiple choices from lists. These programs are the type found on professional application cards. For the average programmer, INPUT, provides an efficient means for many applications.

EPILOG

The Purge File? problem above has many solutions. Here is one that that does not use INPUT.

```
'NY?' << 0 WHILE DROP "     Purge File?" CLLCD 3 DISP "Press N or Y" 7
         DISP DO UNTIL KEY END DUP { 32 52 } SWAP POS 0 SAME REPEAT
         END "File " SWAP IF 32 SAME THEN "Saved" ELSE "Purged" END + >>
           185.5 Bytes  # 27E8h
```

The WHILE ... REPEAT ... END structure and the selection-of-the-input letter logic is the same as described above. Two differences are the use of DISP and KEY. DISP provides the display prompts. KEY provides the input from pressing a SINGLE key. This is very nice because only one key pressing is required. The POS list uses the key codes instead of N or Y. This program is a better solution to the problem in terms of performance.

HOME WORK PROBLEM:

Write a program that displays a list A thru U in three columns. Put a function such as SIN, COS, etc next to the letter. Selecting the key should execute the function. Use the KEY command to provide the input. "NY' may be used as an example.