HEWLETT PACKARD

CALCULATOR SUPPORT
1000 N.E. Circle Boulevard, Corvallis, Oregon 97330-4239, Telephone 503 757-2004 8:00 AM - 3:00 PM Pacific Time

HP 28C/S PROGRAMMING EXAMPLES

The following examples have been developed as a continuing effort by Hewlett-Packard to meet the needs of our customers.

TABLE OF CONTENTS

EXAMPLE 1

This example illustrates the use of conditional tests and nesting, which means that the first conditional test contains another conditional test. The tests are the same, but they do not have to be.

The INP subroutine shows how to prompt for values within a program.

COM1

LEVEL 1	LEVEL 1
price ->	commission

The following HP-28C program calculates the amount of commission paid under these conditions:

* If the purchase price is less than $3,000, the commission is 5% of the price.

* If the purchase price is at least $3,000, but less than $10,000, the commission is 4% of the price, plus $30.

* If the purchase price is $10,000 or more, the commission is 3.8% of the price, plus $20.

```
        PROGRAM                              COMMENTS

<< "PRICE" INP 2 FIX      ! Put "PRICE" on stack, call INP, fix 2
  IF DUP 3000 < THEN      ! Dup value from INP program. If value is
    5 %                   !  less than 3000 computes 5% of value.
  ELSE                    ! Else.
    IF DUP 10000 < THEN   ! If value is 3000 or more but less than
      4 % 30 +            !  10000, computes 4% of value and adds $30.
    ELSE                  ! Else.
      3.8 % 20 +          ! If 10000 or more computes 3.8% and adds $20.
    END                   ! END for second IF statement.
  END                     ! END for first IF statement.
  "COMMISSION=" 36 CHR +  ! Appends "COMMISSION=" and character 36 ($).
  SWAP                    ! Swaps "COMMISSION=$" with value.
  ->STR +                 ! Puts value into string, adds levels 1 and 2.
>>                        ! Ends program.
```

[ENTER] 'COM1 [STO]

INP

```
-----------------------------------------------------------------
|                       LEVEL 1  |  LEVEL 1                     |
-----------------------------------------------------------------
|                        string  ->  n                          |
-----------------------------------------------------------------
```

The input routine INP takes the "PRICE" string off the stack. It stops the subroutine and prompts for the price. To proceed, press CONT. It then displays the information and returns to the calling program.

```
<< -> p                    ! Stores "PRICE" into local variable p.
  << "INPUT " p +          ! Appends contents of p to "INPUT ".
    HALT                   ! Halts program for input of price value.
    SWAP                   ! SWAP value with input prompt.
    DROP p "=" +           ! Drops "INPUT PRICE", adds "=" to "PRICE"
    OVER ->STR +           ! Copy value, put into string, add to "PRICE=".
    SWAP                   ! Swap value in level 2 with string in level 1
  >>                       ! Ends the second program.
>>                         ! Ends the first program.
```

[ENTER] 'INP [STO]

EXAMPLE 1B

Another approach for Example 1 is to use algebraic expressions for the different tests. The following program uses local variables (p), algebraic expressions within a program, and the short form of the If Then Else test.

COM2

LEVEL 1	LEVEL 1
price ->	commission

```
        PROGRAM                          COMMENTS

<< -> p                    ! Stores the price into local variable p.
  << p 3000 <              ! Performs test, puts true/false flag on stack.
     'C=.05*p'             ! Algebraic equation placed on stack.
    << p 10000 >           ! Tests, puts true/false flag on stack.
      'C=.038*p+20'        ! Algebraic equation placed on stack.
      'C=.04*p+30'         ! Algebraic equation placed on stack.
      IFTE                 ! If true flag on level 3, computes 3.8%+20,
                           !  if false flag on level 3 computes 4%+30.
    >>                     ! Ends third program.
    IFTE                   ! If true flag on 3, computes 5%, if false
                           !  flag on 3, value from first IFTE is returned.
  >>                       ! Ends second program.
>>                         ! Ends first program.
```

[ENTER] 'COM2 [STO]

SAMPLE PROBLEM: Using COM1, calculate the commission on an item costing $7000. Using COM2, calculate the commission on a $2500 item.

KEYS	DISPLAY
[USER]	
|COM1|	"INPUT PRICE"
7000 [] [CONT]	"COMMISSION=$310.00"
2500 |COM2|	'C=125'

EXAMPLE 2

The following program converts a 2's complement binary number into a signed real number.

SB->R

```
----------------------------------------------------------------
|                    LEVEL 1   |  LEVEL 1                       |
----------------------------------------------------------------
|               signed binary  ->  real number                  |
----------------------------------------------------------------
```

```
        PROGRAM                          COMMENTS

<< B->R DUP                   ! Convert binary to real and duplicate.
  IF 2 RCWS                   ! Put 2 on stack and recall wordsize.
    1 - ^                     ! Subtract 1 and raise 2 to this power.
    >                         ! Test and put true/false flag on stack.
    THEN 2 RCWS ^ -           ! If flag true, raise 2 to power indicated by
                              !  wordsize and subtract from signed binary.
  END                         ! End conditional test.
>>                            ! End program.
```

[ENTER] 'SB->R [STO]

Example: Convert the 8 bit wordsize #24(hex) into a signed real number.

```
  Press:   8  STWS        ! Set wordsize to 8 bits.
           HEX            ! Set hexadecimal entry/display mode.
           #24  ENTER     ! Put hex 24 on the stack.
           |SB->R|        ! Convert to signed real.
  Returns: 36             !
```

KEYS	DISPLAY	COMMENTS
8 [] [BINARY] |STWS|		Set 8 bit wordsize.
|HEX|		Set HEX mode.
#24		Put HEX 24 on stack.
[USER] |SB->R|	36.00	Convert.

EXAMPLE 2B

Converts a signed real number to a 2's complement binary number. This routine checks if the real number is in a valid range to be converted.

SR->B

LEVEL 1	LEVEL 1
real number ->	signed binary

```
        PROGRAM                              COMMENTS

<< DUP DUP DUP              ! Make copies of the real number, x.
  IF 0 < THEN               ! If x is negative, then
    ABS R->B 0 SWAP -       ! Make a two's complement binary.
  ELSE                      ! Else.
    R->B                    ! Just make a binary.
  END                       ! End conditional test.
  DUP SB->R                 ! Convert a copy of the binary into a real.
  ROT                       ! Rotate.
  IF == THEN                ! If the original x = the converted x, then
    SWAP DROP               !  leave the binary version on the stack.
  ELSE                      ! Else.
    "Out of Range" 1 DISP! Display error message
    1000 .1 BEEP DROP       ! Beep and leave original value on stack
  END                       ! End conditional test.
>>                          ! End program.
```

[ENTER] 'SR->B [STO]

Example: Convert 36 and -11 to an 8 bit 2's complement signed hexadecimal number (type binary).

```
  Press:    8  |STWS|       ! Set wordsize to 8 bits
            |HEX|           ! Set hexidecimal entry/display mode
            36 [ENTER]      ! Put a real 36 on the stack
            |SR->B|         ! Convert to signed binary
  Returns:  # 24            ! Hex 24

  Press:    -11 [ENTER]     ! Put a real -11 on the stack
            |SR->B|         ! Convert to signed real
  Returns:  # F5            ! Hex F5
```

KEYS	DISPLAY	COMMENTS
8 [][BINARY] |STWS|		Set 8 bit wordsize.
|HEX|		Set HEX mode.
36		Put 36 on stack.
[USER] |SR->B|	#24	Convert to 2's complement.
11 [CHS] |SR->B|	#F5	Convert to 2's complement.

EXAMPLE 3

The following program uses the PIXEL command to draw a circle. It uses the FOR/STEP programming technique to repeat the loop. The program also uses the local variable 'x' as the loop counter and as an input to calculate the dependent variable.

CIRCL

LEVEL 1	DISPLAY
	-> plot of circle

```
        PROGRAM                           COMMENTS

<< CLLCD DRAX               ! Clear LCD and draw axes.
   -1 1 FOR x x             ! Define FOR loop put x value on stack.
         1 x SQ - √         ! Compute imaginary part of complex number.
         R->C DUP NEG       ! Convert to complex, dup and negate level 1.
         PIXEL PIXEL        ! Turn on two pixels.
       .1 STEP              ! Increment loop.
>>                          ! End program.
```

[ENTER] 'CIRCL [STO]

To execute the program, press [USER] |CIRCL|.

EXAMPLE 4

This program sorts the numbers contained in a REAL number vector.

Note: Vector "PUT" and "GET" require a list object, so "->LIST" and "LIST->" are used frequently throughout this routine.

Description: A standard bubble sort algorithm is used. It compares two numbers then moves the greater value to the end of the pair. The inner loop, "k", controls the index value used by "GETI". The pairs are compared from first to last-1. The index does not take on the last value when the k'th and k+1'th values are compared, there is no last+1 value. The last pair is placed in order with each pass through the inner loop so the final index value of the next pass can be one less. This process continues until the comparison loop is only comparing one pair.

SORT

LEVEL 1	LEVEL 1
[vector] ->	[vector']

```
       PROGRAM                            COMMENTS

<< 0 0 -> n1 n2              ! Create two local variables.
  << DUP SIZE                ! Duplicate, then compute size of the vector.
     LIST-> -                ! Put list objects and size onto stack.
      1 FOR j                ! Outer loop controlling the ending index.
        1 j FOR k            ! Loop from 1 to decrementing ending index.
            k 1 ->LIST       ! Index the k'th value in vector
            GETI 'n1' STO    ! Store k'th value in n1
            GETI 'n2' STO    ! Store (k+1)'th value in n2
            DROP             ! Drop the index from the stack
            IF n1 n2 > THEN  ! If n1 is greater then n2 then
               k 1 ->LIST    ! Put n1 and n2 back in swapped positions
               n2 PUTI       !
               n1 PUT        !
            END              ! End IF THEN loop.
          NEXT               !
      -1 STEP                !
  >>                         ! End second program.
>>                           ! End first program.
```

Example: Sort a vector containing the values: [5 1 4 2 3]

Press:	[5 1 4 2 3] [ENTER]	Put vector on the stack
	|SORT|	Sort the vector
Leaves:	[1 2 3 4 5]	On the stack.

EXAMPLE 5

The following 28C STEP BY STEP Instruction will duplicate the example on pages 184-5 of the HP-15C Owner's Manual. The example reads:

Champion ridget hurler Chuck Fahr throws a ridget with an upward velocity of 50 meters/second. If the height of the ridget is expressed as

h=5000(1-e^(-t/20))-200t

How long does it take for it to reach the ground again? In this equation h is the height in meters and t is the time in seconds.

For reference purpose, the program in the HP-15C is as follows:

001- [f][LBL] [A]	008- 1	015- [x<>y]
002- 2	009- [+]	016- 2
003- 0	010- 5	017- 0
004- [-]	011- 0	018- 0
005- [CHS]	012- 0	019- [x]
006- [e^x]	013- 0	020- [-]
007- [CHS]	014- [x]	021- [g][RTN]

ON THE HP-28C

Using |ROOT| in the [SOLV] menu:

Key in: 'H=5000*(1-EXP(-T/20))-200*T'

Store the above equation with |STEQ|

Press |RCEQ|
key in:
level 2: 'T'
level 1: { 5 6 }   (your initial estimates for time)

press |ROOT| and the answer should be 9.2843

Using the SOLVR method:

Key in: 'H=5000*(1-EXP(-T/20))-200*T'

Store the above equation with |STEQ|

Press |SOLVR|

Store 0 into H
Store {5 6} into T
Solve for T by pressing [shift] T

Again, the answer should be 9.2843

EXAMPLE 6

The following HP-41 program for will find the two roots of the quadratic equation (a*X^2+b*X+c=0).

ROOT1=(-B+V B^2-4AC)/2A   ROOT2=(-B-V B^2-4AC)/2A

```
01 LBL "QROOT"
02 "a?"
03 PROMPT
04 STO 01
05 "b?"
06 PROMPT
07 STO 02
08 "c?"
09 PROMPT
10 STO 03
11 XEQ 01
12 -
13 RCL 01
14 2
15 *
16 /
17 PSE
18 XEQ 01
19 +
20 RCL 01
21 2
22 *
23 /
24 PSE
25 RTN
26 LBL 01
27 RCL 02
28 CHS
29 RCL 02
30 X^2
31 RCL 01
32 RCL 03
33 *
34 4
35 *
36 -
37 SQRT
38 END
```

This program can be written as follows on the HP-28C. The program, NU, on the HP-28C is comparable to the subroutine, LBL 01, on the HP-41.

QROOT

LEVEL 3	LEVEL 2	LEVEL 1	LEVEL 1	LEVEL 2
a	b	c	-> 2nd root	1st root

```
<< 'C' STO 'B' STO 'A' STO  ! Stores values into ordinary variables.
  NU -                      ! Calculate numerator of the first root.
  A 2 *                     ! Calculate denominator of the first root.
  / 'ROO1' STO              ! Calculate first root and store in ROO1.
  NU +                      ! Calculate numerator of second root.
  A 2 *                     ! Calculate denominator of second root.
  / 'ROO2' STO              ! Calculate second root and store in ROO2.
  CLEAR ROO1 ROO2           ! Clear, then put root 1 and 2 in display.
>>                          ! End program.
```

[ENTER] 'QROOT [STO]

```
<< B NEG                    ! Put B value on stack and negate.
  B SQ                      ! Put B value on stack and square it.
  A C *                     ! Put A and C on stack and multiply.
  4 *                       ! Multiply 4 times A*C.
  - V                       ! Subtract 4*A*C from B^2 and square root.
>>                          ! End program.
```

[ENTER] 'NU [STO]

Example:

```
Key in: 1 [ENTER] [ENTER]
        6 [CHS][ENTER] [USER] |QROO|       -3.00
                                            2.00
```

EXAMPLE 7

These are companion programs that give a user post-fix like storage to numbered registers 0-9. The register storage location is a list named 'REGS'. 'STOR' and 'RCLR' will prompt for a register number.

STOR

LEVEL 1	LEVEL 1
object	-> object

```
<< "Store in (0-9)?" 4 DISP ! prompt for the storage register
   CKREGS                   ! verify/create the register list
   'REGS'                   ! name of the register list
   GETNUM                   ! get the register number
   1 +                      ! register #'s start at 0 lists at 1
   3 PICK                   ! get a copy of the object to store
   PUT CLMF                 ! put object in list, restore display
>>                          ! ends program
```

RCLR

LEVEL 1	LEVEL 1
	-> object

```
<< "Recall From (0-9)?" 4 DISP   ! Prompt for the storage register
   CKREGS                        ! Verify/create the register list
   'REGS'                        ! Name of the register list
   GETNUM                        ! Get the register number
   1 +                           ! Register #'s start at 0 lists at 1
   GET                           ! Get object from list
   CLMF                          ! Restore display
>>                               ! Ends program
```

GETNUM: Returns a value 0-9 corresponding to a key press 0-9 or else it beeps.

GETNUM

```
-----------------------------------------------------------------
|                    LEVEL 1  |  LEVEL 1                         |
-----------------------------------------------------------------
|                            ->  n                               |
-----------------------------------------------------------------
```

```
<< DO                          ! Loop until a valid key press
    DO KEY UNTIL END           ! Loop waiting for a key
    NUM 48 - DUP DUP           ! Convert key press string into number.
   UNTIL                       ! Begin UNTIL clause.
    IF 0 < SWAP 9 > OR         ! If key was not in range
    THEN 1000 .2 BEEP DROP     ! Then beep.
      0                        ! Set key flag to false
    ELSE 1                     ! Else set valid key flag to true
    END                        ! End IF routine.
   END                         ! End DO loop.
>>                             ! End program.
```

CKREGS: Checks that the 'REGS' variable exists and creates it if absent. CKREGS assumes that, if the variable exists, it is the correct type and size.

CKREGS

```
-----------------------------------------------------------------
|                    LEVEL 1  |  LEVEL 1                         |
-----------------------------------------------------------------
|                            ->                                  |
-----------------------------------------------------------------
```

```
<< IF REGS TYPE 6 ==           ! If REGS is a name
   THEN                        ! Then user object doesn't exist, so
   i                             create one.
     { 10 } 0 CON ARRY->       ! Put 10 zeros and ten on the stack
     DROP 10                   ! Drop the array size.
     ->LIST                    ! Convert zeros to a list
     'REGS' STO                ! Store list in 'REGS'
   END                         ! End IF loop.
>>                             ! Ends program.
```

EXAMPLE 8

ERROR TRAPPING PROGRAM

The following program can be used in conjunction with function that cause a 'NON-REAL RESULTS' error message when plotting.

ERRT

```
--------------------------------------------------------------
|                       LEVEL 1  |  LEVEL 1                  |
--------------------------------------------------------------
|                                ->                           |
--------------------------------------------------------------
```

```
<< DEPTH -> D              ! Counts objects on stack, stores in D.
  << 31 SF                 ! Sets the LAST enable flag.
     IFERR ->NUM RE        ! Begins error trap routine.
     THEN DEPTH D SWAP     ! Error clause of error trap routine.
       DROPN MAXR          !
     END                   ! End IFERR routine.
  >>                       ! End second program.
>>                         ! End first program.

'ERRT' STO
```

To use ERRT, write a program such as:

```
          << 'LOG(X)' ERRT >>
```

Go to the PLOT menu, |STEQ| |DRAW|.

EXAMPLE 9

FCIRCLE

LEVEL 1	LEVEL 1
'symb'	-> (values returned from DGTZ)*

FCIRCLE takes the algebraic object 'symb' that defines a circular function, and produces a circle plot. 'symb' must use variables X and Y in the form:

'(X+5)^2+(Y+1)^2=25' or 'X^2+Y^2=9'

NOTE: This program does not modify the contents of PPAR, therefore, it may be necessary to adjust PPAR to obtain your desired plot.

This program leaves the following variables in USER:

CEQ - The algebraic object 'symb'. This permits editing of the original equation.
EQ - Current equation.

```
         PROGRAM                          COMMENTS

<<  36 SF              ! Set Result Mode
    { X Y } PURGE      ! Purge variables X and Y (if they exist)
    DUP                ! Duplicate 'symb' on the stack
    'CEQ' STO          ! Store one copy of 'symb' as CEQ (Circle EQuation)
    CLLCD              ! Clear the display
    DUP 1 DISP         ! DISP the equation.
    'Y' ISOL DUP       ! Isolate the Y variable 'symb', duplicate result
    -1 's1' STO        ! Store -1 as 's1'
    EVAL               ! Evaluate expression to get rid of 's1'
    SWAP               ! Place the expression in level 1
    1 's1' STO         ! Store 1 as 's1'
    EVAL               ! Evaluate expression
    =                  ! Set the expressions equal to each other
    STEQ DRAW          ! Store current equation, draw it
    's1' PURGE         ! Purge variable 's1'
    DGTIZ              ! * Digitize the plot
>>
```

[ENTER] 'FCIRCLE [STO]

* This command is valid only when using the HP-28S.

SAMPLE PROBLEM: Plot the expression X^2 + Y^2 = 9.

KEYS	DISPLAY
[USER]	
'X^2+Y^2=9'	'X^2+Y^2=9'
|FCIRCLE|	(plot)