EDITMACRO and RUNMACRO
Eddie Shore

11/11/2014

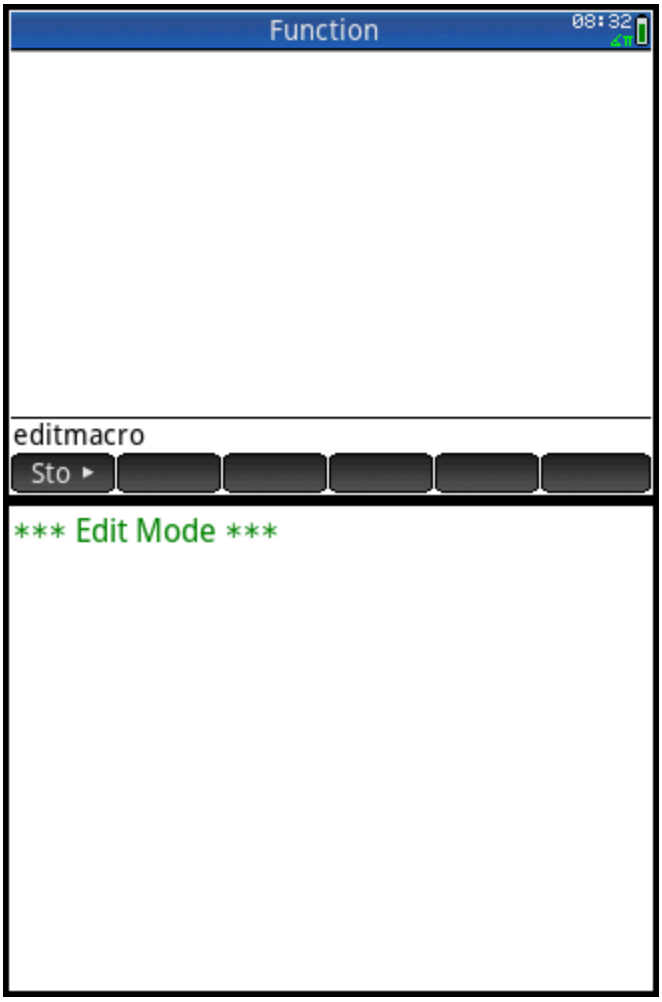
The programs **editmacro** and **runmacro** bring the powerful keystroke RPN (Reverse Polish Notation) programming to the HP Prime. If you are familiar with the Hewlett Packard programming calculators (HP 65, 67, 12C, 15C, 35S, and a whole lot more), you will feel at home using this program package.

Total size: about 24,000 bytes.

A four-level stack is used.

TABLE OF CONTENTS

How It Works

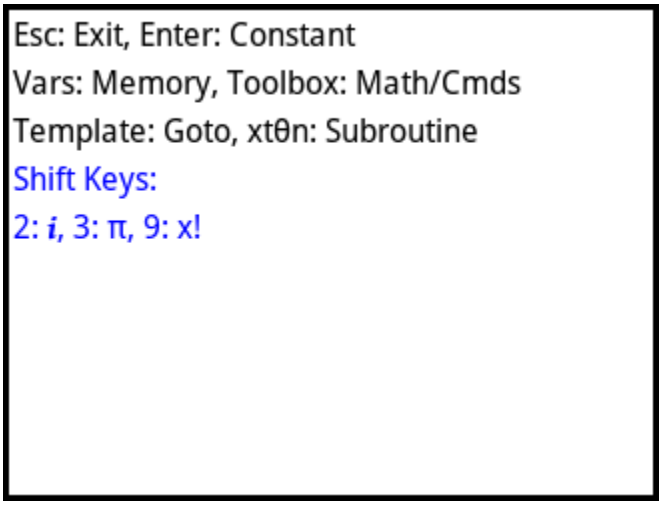


The first step is to run **editmacro**. You will to be taken to the edit mode where you enter various constants and commands using the keyboard. Once complete, press [Esc] and a list of your recorded steps are presented. This list can be saved to any of the default list variables L0 through L9, or a custom variable name.
Macros are ran through the runmacro(*macroname*) program. This is where you enter data and get results.

The only limit to how big a macro list can be is limited to the size of the memory. Hence, you can create short and quick macros, or elaborate macros with ease.

Running EDITMACRO and RUNMACRO for the First Time

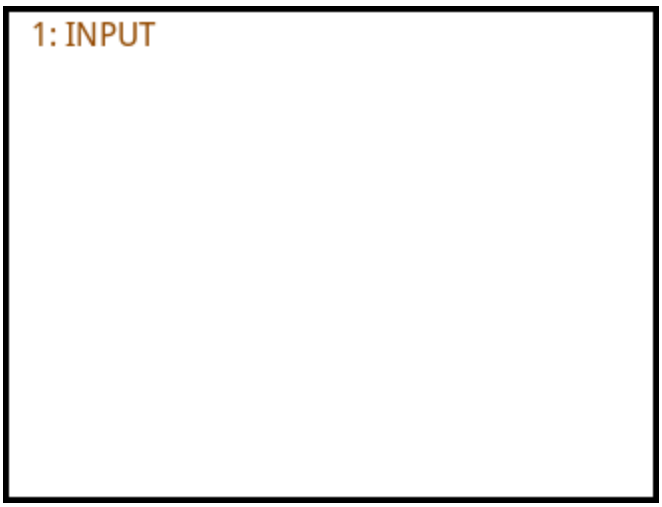
Start by running editmacro (all lowercase). You will be taken to the edit mode. Press the [? Help] key.



The help screen will give you a list of keys to press to access various commands. You can access the common scientific calculator functions by just using your keyboard. We will get to these in detail later. For now, just press [Esc] to exit the Help Screen.

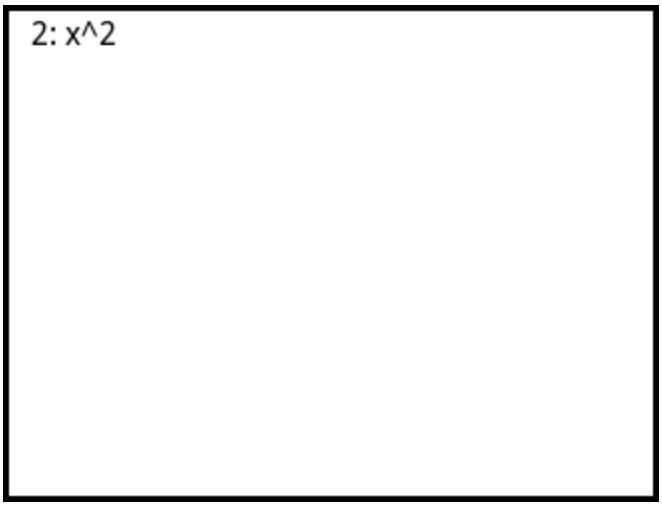
Let's start our first macro with something familiar: the area of a circle. The required formula is $A = \pi r^2$, where r is the radius.

We will let the user enter the radius. To allow the user to enter data, we will use the INPUT command. Press [Toolbox], [2] (for Control), and then [1] (for INPUT).



Note the screen will now read: "1: INPUT". This indicates that the INPUT command is the first step in this macro.

We will need to square this radius. Press [x^2].

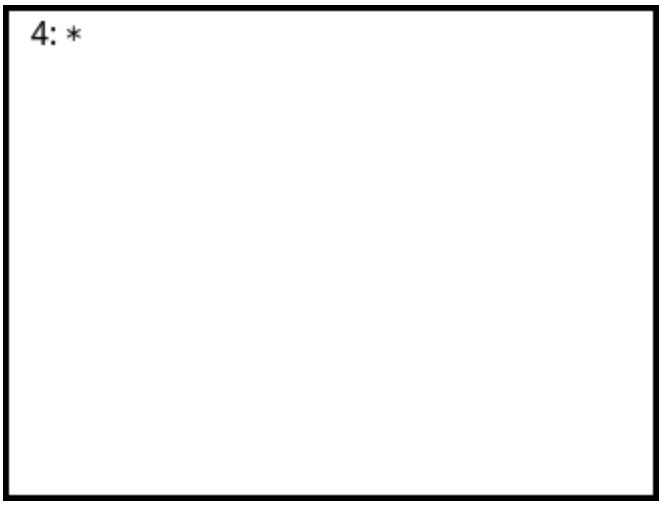


The “2: x^2” on the screen shows that we have entered the square command, and it is the second step in the macro.

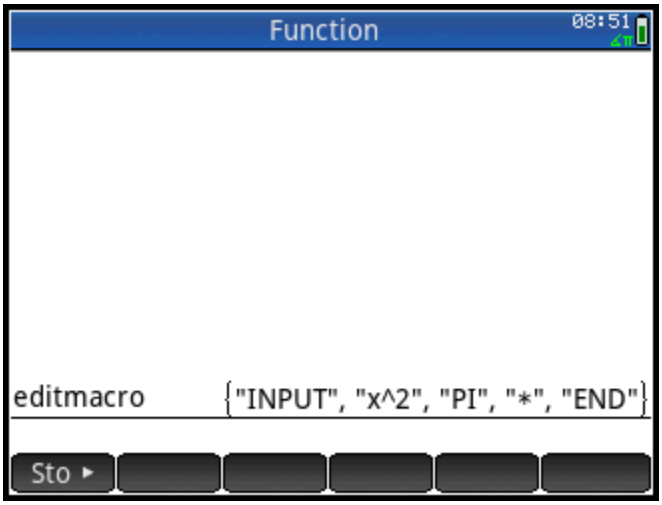
The next thing we need to do is to put π on the stack. Remember that **editmacro** and **runmacro** uses RPN logic. Press [Shift], [3] (for π).

The “3: PI” on the screen indicates that the third step of this macro is to enter π on the stack. Note that “3: PI” is in blue because we have used a shifted command.

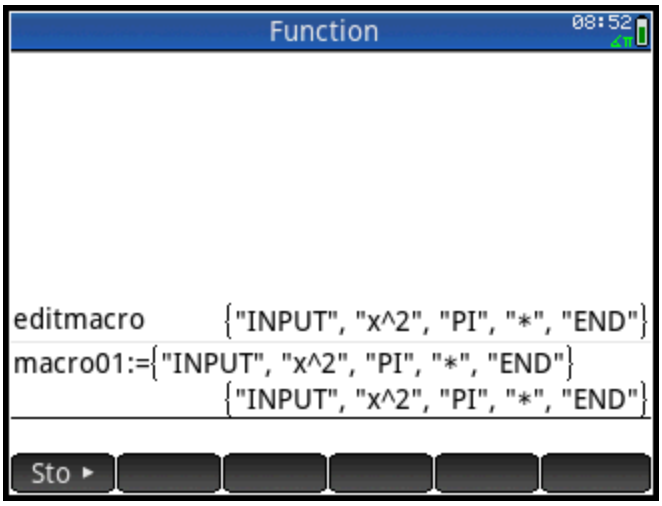
To finish, we need to multiply. This is accomplished by pressing [x].



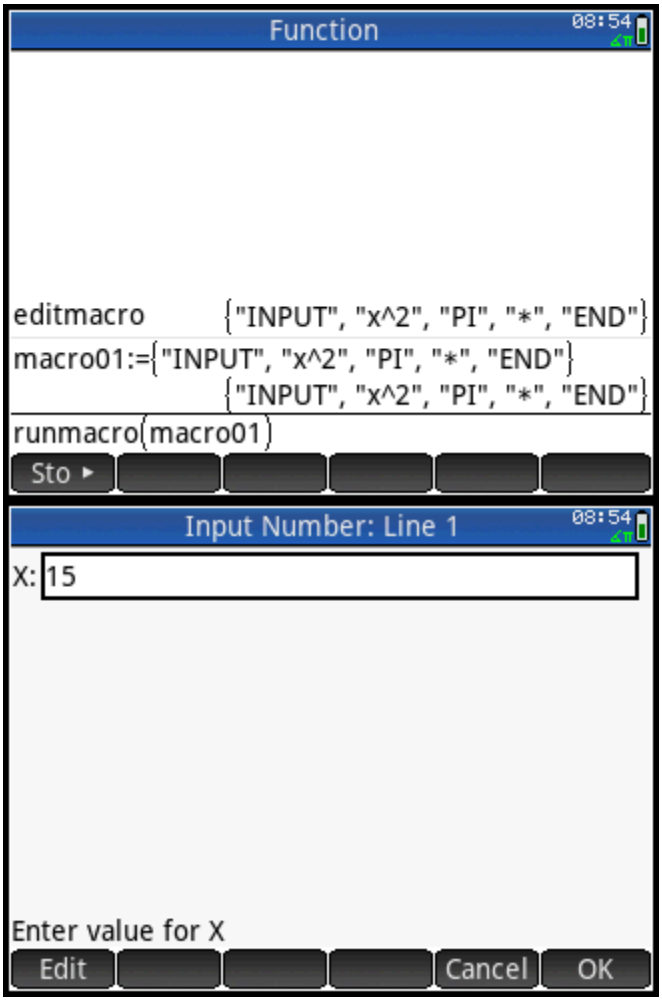
Now that our macro is done, press the [Esc] key.



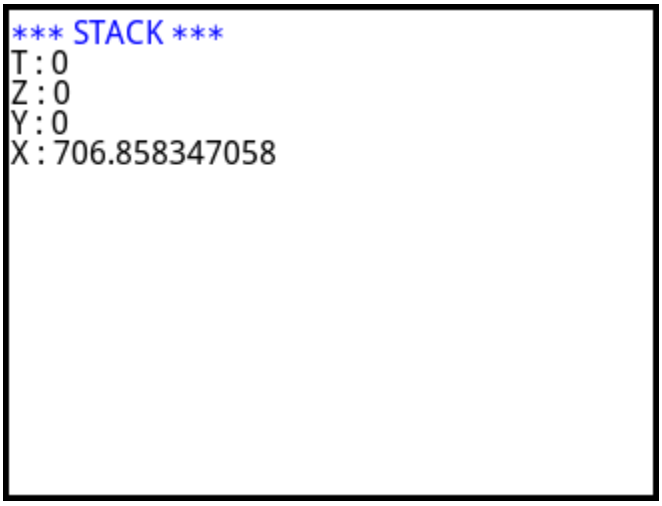
You are returned to the home screen with the list as the output: {“INPUT”, “x^2”, “PI”, “*”, “END”}. The last command, “END” is added automatically. You can run this list as is, or save it. For our example, I am going to save this macro to the variable name *macro01*.



Now it is time to run the macro. The program **runmacro** requires one argument: either a list (in the proper format) or the name of a macro. For our example, let's run *macro01*.

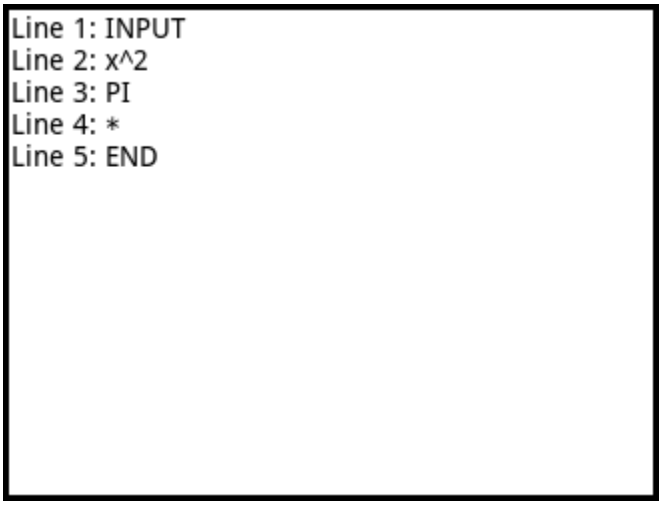


We are prompted for a number. The number must be a real number or an error occurs. Let's say we are trying to find out the area of a circle with radius of 15 units. Enter 15 and then press either (OK) or the [Enter] key.

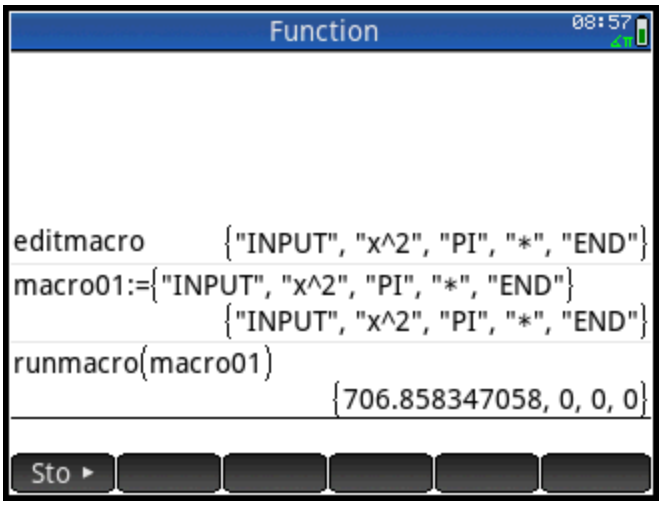


When the marco ends, you are shown the four-level stack. On a keystroke programming calculator with a one-line display, you are shown the contents of the X stack. In our example, the area of a circle with the radius of 15 units is just over 706.858 square units.

Press [Enter] and you will get this screen:



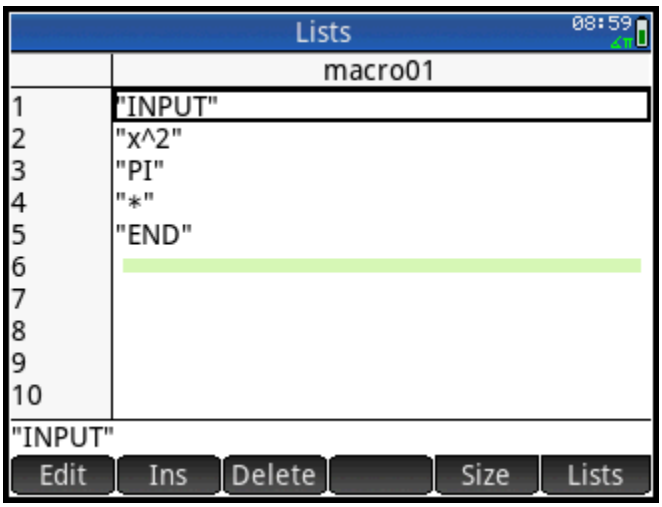
This screen shows the commands that macro used. You can use this screen for diagnostics. Or, you can press [Enter] once more:



You are now at the Home Screen. You will see a four-element list which shows the contents of the stacks X, Y, Z, and T.

Using EDITLIST to edit Macros

The **editmacro** program offers a nice interface to enter macros. However, you can use EDITLIST or the HP Prime's list editor to edit macros.



Note that each command, except for constants, are in strings. Each command has a specific string that is required to run the macros properly.

Entering Constants

One major difference between **editmacro** and RPN keystroke programmable calculators is the way that numerical constants are entered. In **editmacro**, you will need to first press the [Enter] key first. You will be asked for a constant to enter. You

can enter numbers or expressions that evaluate to a real number. Let's demonstrate that with our next macro:

You are purchasing the required textbooks to take classes at Generous University. The first two weeks of each semester at Generous, the bookstore gives students a 20% discount on all text books purchased. What is the discount, and then price after the discount?

Once again, we will ask the user to enter the costs of the textbooks. For this macro, let's make it pause to show us the discount first, and then final cost. Hence, the macro will be this:

1: "INPUT"
2: "DUP"
3: 0.2
4: "*"
5: "PAUSE"
6: "-"
7: "END"

"DUP" duplicates the contents on the X stack and moves a copy to the Y stack.

There is no percent function in **editmacro**. Hence, we will need to enter the decimal equivalent of 20% = 0.2.

Run **editmacro**.

1: "INPUT"
Press [Toolbox], [2] (Control), [1] (INPUT)

2: "DUP"
Press [Toolbox], [2] (Control), [2] (DUP)

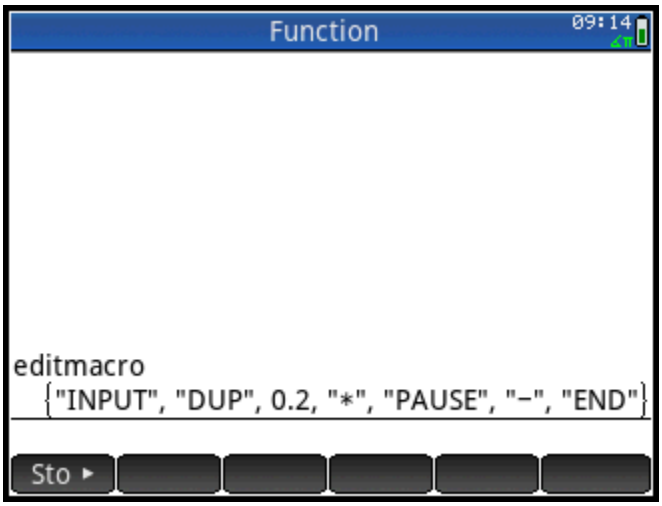
3: 0.2
Press [Enter], enter 0.2, press either (OK) or [Enter]

4: "*"
Press [x]

5: "PAUSE"
Press [Enter], [2] (Control), [Template] (or scroll up for option C: PAUSE)

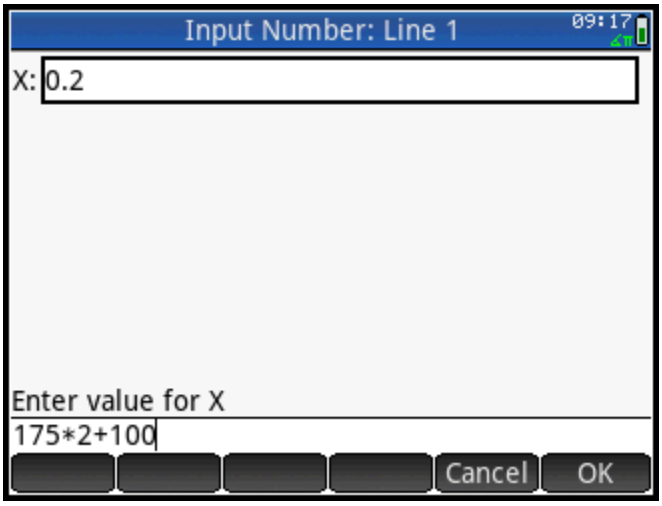
6: "-"
Press [-].

The last “END” is automatic when terminate an **editmacro** session. Press [Esc]. You should have this:

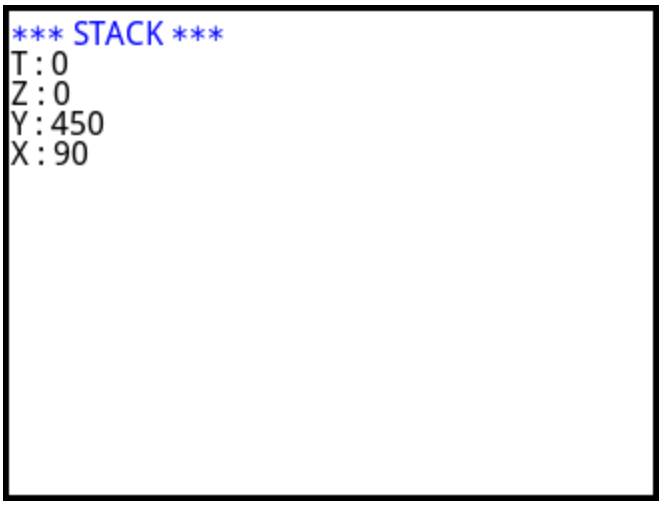


Save this macro if desired. I am going to use the variable name *macro02*. Run this macro.

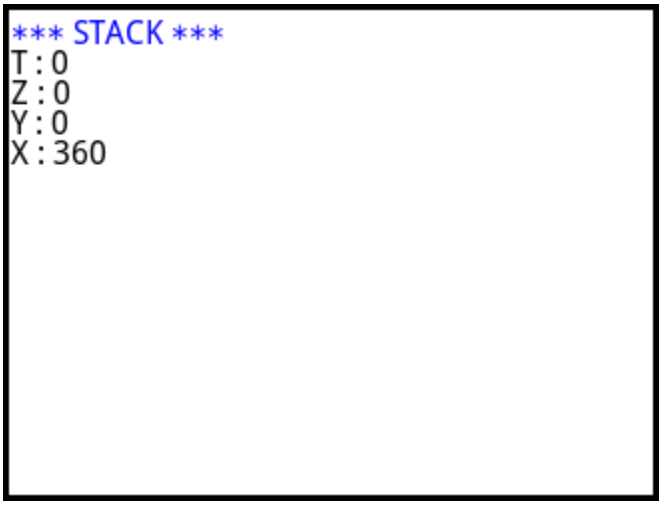
For our example, let’s say we had to buy 2 textbooks for $170 apiece and 1 at $100.



At the “Input Number: Line 1” prompt, enter 175*2+100. Remember we can enter numerical expressions at the INPUT and Constant prompts.



At this point, we are shown that the discount is $90 (and the original cost of the textbooks is $450). Press [Enter] again.



You are shown that the final cost of the textbooks is $360: still a lot of money after the discount.

Macro Structure

Each macro starts at Line 1. These code lines are important because they act as pointers to guide the macro as it is running. When executing jumps, testing, and subroutines, you will need to know where you direct the macro's pointer.

Each macro can run subroutines one level deep.

Variables

All variables in **editmacro** and **runmacro** are considered local. That is, nothing contained in the variables will never change any of the variable's contents outside of these programs. There are nine variables that can be used as temporary storage registers: 1 through 9. Everything is self-contained in the macros.

The default value of the storage registers is 0.

Store Arithmetic

You can store numbers to 9 temporary registers. You can also perform simple arithmetic functions:

STO+: add the number in the X stack to the register
STO-: subtract the number in the X stack *from* the register
STO*: multiply the number in the X stack to the memory register
STO/: divide the register by the number X in the stack
INC: increase the designated register by 1
DEC: decrease the designated register by 1

INC and DEC are useful in loops.

Comparison Testing

There are six comparison tests that are presented in **editmacro** and **runmacro**. Each of the comparison tests have the general format:

n: *comparison test*
n+1: *do this command if test is true*
n+2: *skip command at n+1 if test is false*

This scheme operates just as other RPN keystroke programming calculators.

The six comparison tests are:

TEST	TRUE	FALSE
x=0?	The X stack is zero.	The X stack is not zero.
x>0?	The X stack is positive.	The X stack is zero or negative.
x<0?	The X stack is negative.	The X stack is zero or positive.
x=y?	X = Y	X ≠ Y
x>y?	X > Y	X ≤ Y
x<y?	X < Y	X ≥ Y

Last X Function

Access the Last X function by pressing [Shift] [,]. Like classic RPN keystroke programmable calculators, **editmacro** and **runmacro** reserves a special register for storing the latest value of X prior to using a mathematical operation. Last X recalls this X onto the stack – which can be useful in programming.

The following commands store the contents of X in this register prior to execution: the arithmetic functions, EXP, ALOG, LN, LOG, the trigonometric functions, powers, roots, INT, FRAC, COMB, PERM, 1/x, and polar/rectangular conversions.

Forming Complex Numbers

Unfortunately, we can directly enter complex numbers in **editmacro**. The workaround is the [Shift] [2] key sequence. What this does is that the X stack is multiplied by $\sqrt{-1}$. To form a complex number, use this schematic:

1. Enter a constant or prompt for a number. This will be the real part of the complex number.
2. Enter a constant or prompt for a number. This will be the imaginary part of the complex number.
3. Press [Shift] [2], then press [+]

Note that this process requires two stack levels. If complex mode is turned on (“Allow complex output from real input”) in the Home Settings, you can execute calculations that return complex numbers.

Commands Catalog

Keyboard Commands

KEY	STRING CODE	DESCRIPTION CODE	RESULT STACK
[Del]	None	Deletes the last command in **editmacro**	
[Enter]	(numerical constant)*	Calls up the prompt to enter a numeric constant	Numeric constant, X, Y, T
[x^y]	"^"	Power function.	X^Y, T, Z, Z
[Shift], [x^y]	"XROOT"	Root function	X^(1/Y), T, Z, Z
[SIN]	"SIN"	Sine function.	SIN(X), Y, T, Z
[Shift] [SIN]	"ASIN"	Arcsine function.	ASIN(X), Y, T,Z
[COS]	"COS"	Cosine function.	COS(X), Y, T, Z
[Shift] [COS]	"ACOS"	Arccosine function.	ACOS(X), Y, T, Z
[TAN]	"TAN"	Tangent function.	TAN(X), Y, T, Z
[Shift] [TAN]	"ATAN"	Arctangent function.	ATAN(X), Y, T, Z
[LN]	"LN"	Natural logarithm function.	LN(X), Y, T, Z
[Shift] [LN]	"e^x"	Natural exponential function.	e^X, Y, T, Z
[LOG]	"LOG"	Common logarithm function.	LN(X)/LN(10), Y, T, Z
[Shift] [LOG]	"10^x"	Common antilog function.	10^X, Y, T, Z
[x^2]	"x^2"	Square function.	X^2, Y, T, Z
[Shift] [x^2]	"SQRT"	Square root function.	√X, Y, T, Z
[+/-]	"CHS"	Changes the sign (negates) of the X register.	-X, Y, T, Z
[Shift] [+/-]	"ABS"	Absolute value of the X register.	|X|, Y, T, Z,
[( )]	"DOWN"	Rolls down the stack one level. Puts the contents of Y on X, and so on.	Y, T, Z, X
[Shift] [( )]	"UP"	Rolls up the stack one level. Puts the contents of Z on X, and so on.	Z, X, Y, T
[,]	"SWAP"	Swap the contents of the X and Y stacks.	Y, X, T, Z
[Shift] [,]	"LAST"	The Last X function	Last X, X, Y, Z
[÷]	"/"	Division.	Y/X, T, Z, Z
[Shift] [÷]	"1/x"	Reciprocal	1/X, Y, T, Z
[x]	"*"	Multiplication	X*Y, T, Z, Z
[-]	"-"	Subtraction	Y-X, T, Z, Z
[+]	"+"	Addition	X+Y, T, Z, Z
[Shift] [9]	"FACT"	Factorial function. X does not need to be an integer.	X!, Y, T, Z
[Shift] [3]	"PI"	Enters π to the stack.	π, X, Y, Z

[Shift] [2]	"IMAG"	Multiplies X by √-1. This makes it possible to use complex numbers.	Xi, Y, Z, T

Note: Numerical constants are not in strings.

Memory Menu – [Vars] key

See the **Store Arithmetic** section for more details.

STRING CODE (n = 0 through 9)	DESCRIPTION
"STO n"	1. Store. Stores X in register n.
"RCL n"	2. Recall. Recalls register n to the stack. Result stack: register n, X, Y, Z
"SUM n"	3. STO+
"DIF n"	4. STO-
"PRD n"	5. STO*
"DIV n"	6. STO/
"INC n"	7. Increment
"DEC n"	8. Decrement

Math Menu – [Toolbox], [1]

STRING CODE	DESCRIPTION CODE	RESULT STACK
"INT"	1. INT Returns the integer part of X.	IP(X), Y, Z, T
"FRAC"	2. FRAC Returns the fractional part of X	FP(X), Y, Z, T
"RECT"	3. >Rect Polar to rectangular conversion.	X*COS(Y), X*SIN(Y), Z, T
"POLAR"	4. >Polar Rectangular to polar conversion	√(X^2+Y^2),ARG(X+Y*i), Z, T
"COMB"	5. COMB Combination function	Y nCr X, Z, T, T
"PERM"	6. PERM Permutation function	Y nPr X, Z, T, T
"RAND"	7. RAND Enters a random number from 0 to 1	'rand, X, Y, Z
"ARG"	8. ARG Argument or angle of a complex number	ARG(X), Y, Z, T
">D"	9. >Dec Deg Converts number from DMS (degree-minutes-seconds) format to full decimal	X>D, Y, Z, T
">DMS"	A. >Deg Min Sec Converts full number to DMS format	X>DMS, Y, Z, T

DMS format: DD.MMSSSS (degrees, minutes, seconds)

Control Menu – [Toolbox] [2]

STRING CODE	DESCRIPTION CODE	RESULT STACK
"INPUT"	1. INPUT When the macro runs, it will prompt the user for a number	entered data, X, Y, Z

“DUP”	2. DUP Duplicates the contents of the X stack	X, X, Y, Z
“CLx”	3. CLx Sets the value of the X stack to zero	0, Y, Z, T
“DEG”	4. DEG Sets Degrees mode	(no changes to the stack)
“RAD”	5. RAD Sets Radians mode	(no changes to the stack)
“x=0?”	6. x=0? Comparison test	(no changes to the stack)
“x>0?”	7. x>0? Comparison test	(no changes to the stack)
“x<0?”	8. x<0? Comparison test	(no changes to the stack)
“x=y?”	9. x=y? Comparison test	(no changes to the stack)
“x>y?”	A. x>y? Comparison test	(no changes to the stack)
“x<y?”	B. x<y? Comparison test	(no changes to the stack)
“PAUSE”	C. PAUSE Stops the execution of the macro. Shows the stack. Press [Enter] to continue.	(no changes to the stack)
“RTN”	D. RTN Sets a RTN (return) command. This ends a subroutine.	(no changes to the stack)
“END”	E. END Sets an END command. This ends a macro and stops execution.	(no changes to the stack)

Go To, Subroutine, and Labels

GTO n: Go To Command – [Template] key

You can transfer control to specific points in the macros by using labels. Each macro can contain up to 9 labels. When the macro runs and hits a GTO command, it will transfer the pointer to the specified label (LBL).

String Code: “GTO *n*” where n is the label number.

XEQ n: Subroutine Command - [x t θ n] key

You can temporary execute a subroutine. Subroutines end with a return (RTN) command.

String Code: “XEQ *n*” where n is the label number.

LBL n: Label Command - [a b/c] key

Press the fraction key into insert labels in the macro. Each macro can have up to 9 labels.

String Code: “LBL *n*” where n is the label number.

Sample Programs Catalog

Comments are followed by double slashes (//). Memory registers are designated by "R*n*" where *n* is the register number 1 through 9. Remember, the *last* END is inserted atomically when you exit **editmacro**. Also, constants (numbers not in strings) must be entered by pressing [Enter] first then enter the number when prompted.

Large Factorials

Based on HP 41C – Brian D Steel – 1/5/1984 – Datafile March & April 1984 Vol. 3 No. 2 Pg. 24
Output:
Y: exponent
X: mantissa

Where: x*10^y

Mantissa is an approximate!

1: "INPUT" // asks for a number
2: "INT" // automatically make it an integer
3: "STO 1"
4: 0 // set up the sum
5: "STO 2"
6: "LBL 1"
7: "RCL 1" // loop starts here
8: "LOG" // need LOG, not LN
9: "SUM 2"
10: "DEC 1" // M1 – 1 = M1
11: "RCL 1"
12: "x>0?"
13: "GTO 1"
14: "RCL 2"
15: "DUP"
16: "INT" // separate the exponent portion
17: "SWAP"
18: "FRAC"
19: "ALOG" // extract mantissa portion
20: "END"

Example:
12 -> {4.79001599997, 8, 0, 0}
125 -> {1.882677182, 209, 0, 0}

Loop Demonstration: Pascal's Triangle Loop

This program will show all of the entries of the nth row of Pascal's Triangle.

Registers: R1 = n, R2 = counter

Program:

1: "INPUT" // ask for n
2: "STO 1"
3: 0 // start a counter
4: "STO 2"
5: "LBL 1" // loop starts here
6: "RCL 1"
7: "RCL 2"
8: "COMB"
9: "PAUSE" // shows each entry on Pascal's Triangle
10: "INC 2" // increase counter by 1
11: "RCL 2"
12: "RCL 1"
13: "-"
14: "x>0?" // is counter>n?
15: "END" // if yes, then stop program
16: "GTO 1" // if no, run loop again
17: "END"

Quadratic Equation Solver

$ax^2 + bx + c = 0$

This program assumes that the “Allow Complex Output from Real Input” is checked. Go to Settings and check the box next to the Complex scroll box.

Program:

1: “INPUT” // input a
2: “STO 1”
3: “INPUT” // input b
4: “STO 2”
5: “INPUT” // input c
6: “STO 3”
7: “RCL 2” // -b/(2a)
8: “CHS”
9: 2
10: “RCL 1”
11: “*”
12: “/”
13: “STO 4”
14: “RCL 2” // √(b^2-4ac)/(2a)
15: “x^2”
16: “RCL 1”
17: “RCL 3”
18: “*”
19: 4
20: “*”
21: “-“
22: “SQRT”
23: 2
24: “RCL 1”
25: “*”
26: “/”
27: “STO 5”
28: “RCL 4” // root 1
29: “RCL 5”
30: “+”
31: “RCL 4” // root 2
32: “RCL 5”
33: “-“
34: “END”

Output: Y Stack: root 1, X Stack: root 2

Examples:
Variables: a = 1, b = 3, c = 6
Y: -1.5 + 1.93649167311i
X: -1.5 – 1.93649167311i

Variables: a = -2, b = 6, c = 7
Y: -0.89791576166
X: 3.89791576166

Solving a 2 x 2 Linear System

$a * x + b * y = e$
$c * x + d * y = f$

Solution:

$x = (d * e - b * f)/(a * d - b * c)$
$y = (a * f - c * e)/(a * d - b * c)$

Variables Used:

R1 = a, R2 = b, R3 = c, R4 = d, R5 = e, R6 = f, R7 = a*d-b*c

Input order: a, b, c, d, f, e

Program:

1: "INPUT" // input a
2: "STO 1"
3: "INPUT" // input b
4: "STO 2"
5: "INPUT" // input c
6: "STO 3"
7: "*"
8: "CHS"
9: "SWAP"
10: "INPUT" // input d
11: "STO 4"
12: "*"
13: "+"
14: "STO 7" // a*d-b*c
15: "RCL 1" // calculate y
16: "INPUT" // input f
17: "STO 6"
18: "*"
19: "RCL 3"
20: "INPUT" // input e
21: "STO 5"
22: "*"
23: "-"
24: "RCL 7"
25: "/"
26: "PAUSE"
27: "RCL 4" // calculate x
28: "RCL 5"

29: “*”
30: “RCL 2”
31: “RCL 6”
32: “*”
33: “-“
34: “RCL 7”
35: “/”
36: “END”

Example: a = 2, b = 1, c = -3, d = 2, f = 3, e = -1.

Results: x = -0.714285714286, y = 0.428571428571

Payment of a Loan

Source: HP 65 Standard Pac – June 1974

Formula: $PV = PMT * \frac{(1+\frac{I}{100N})^{YN}-1}{\frac{I}{100N}*(1+\frac{I}{100N})^{YN}}$

Input Order: I, N, Y, PV

Variables Used:
R1 = N
R2 = I/(100N)
R3 = Y*N

Program:

1: “INPUT” // I
2: “INPUT” // N
3: “STO 1”
4: 100
5: “*”
6: “/”
7: “STO 2”
8: 1
9: “+”
10: “INPUT”
11: “RCL 1”
12: “*”
13: “STO 3”
14: “^”
15: 1
16: “-“
17: “RCL 2”
18: 1
19: “+”
20: “RCL 3”
21: “^”
22: “RCL 2”
23: “*”
24: “/”
25: “1/X”
26: “INPUT” // PV
27: “*”
28: “END”

Subroutine Demonstration

Let $f(x) = \frac{x*\phi^2}{1-\phi}$ where $\phi = \sqrt{1-x^2}$

Registers: R1 = x, R2 = Φ

Program:

1: "INPUT" // enter x
2: "STO 1"
3: "XEQ 1" // run the subroutine starting at LBL 1
4: "x^2"
5: "*"
6: 1
7: "XEQ 1" // run the subroutine again
8: "-"
9: "/"
10: "END" // end the main program and show results
11: "LBL 1" // start the subroutine
12: 1
13: "RCL 1"
14: "x^2"
15: "-"
16: "SQRT"
17: "RTN" // RTN command needed – ends the subroutine
18: "END"

Example: f(0.5) = 2.79903810561

Numeric Derivative Approximation: Five Stencil Method

General Formula:

$$f'(x) = \frac{1}{12H} * \big(f(x-2H) - 8(f-H) + 8(f+H) - f(x+2H) \big) + \frac{H^4}{30}$$

H is a small number, used as a tolerance setting.

Source:
Burden, Richard L. and Douglas Faires. "Numerical Analysis 8th Edition" Thompson Brooks/Cole. Belton, CA 2005.

Example: Find the slope of f(x) = e^(sin x) at x = -1 and x = 1. Use H = 1*10^-6

Registers:
R1 = temporary – use Register 1 as your x in f(x)
R2 = H
R3 = temporary
R4 = X
R5 = f'(x)

Program:

1: "RAD" // set radians mode
2: "INPUT" // ask for X
3: "STO 1"
4: "STO 4"
5: 0.000001
6: "STO 2" // H
7: 2
8: "*"
9: "DOWN"
10: "STO 3"
11: "UP"
12: "+"
13: "STO 1"
14: "XEQ 1" // execute subroutine
15: "CHS"
16: "STO 5"
17: "RCL 3"
18: "RCL 2"
19: "+"
20: "STO 1"
21: "XEQ 1"

22: 8
23: “*”
24: “SUM 5” // STO+ 5
25: “RCL 3”
26: “RCL 2”
27: “-“
28: “STO 1”
29: “XEQ 1”
30: 8
31: “*”
32: “DIF 5” // STO- 5
33: “RCL 3”
34: “RCL 2”
35: 2
36: “*”
37: “-“
38: “STO 1”
39: “XEQ 1”
40: “SUM 5”
41: 12
42: “RCL 2”
43: “*”
44: “DIV 5” // STO/ 5
45: “RCL 5”
46: “END”
47: “LBL 1” // function starts here, f(R1)
48: “RCL 1”
49: “SIN”
50: “e^x”
51: “RTN” // Use RTN to end f(x)
52: “END”

Examples:
f’(-1) ≈ 0.232912 (Actual: 0.23291330138…)
f’(1) ≈ 1.2533875 (Actual: 1.25338076749…)