

Programming the Texas Instruments TI-83/TI-84+ Graphics Calculator.

by

Anthony S. Pyzdrowski Ph.D.
Department of Mathematics and Computer Science
California University of Pennsylvania

and

Laura J. Pyzdrowski, Ed.D.
Department of Mathematics
West Virginia University

Why Write a Program?	1
The PRGM menu	1
Creating a Program on the Calculator	2
The Circle Area Program	2
Executing the Circle Area Program	5
Clearing the Display	6
Drawing a Circle Program	7
Program Accessing Other Programs: prgm	10
Useful Programming Statements	11
Program Termination Statements: Stop and Return	11
Input Statements: Input, Prompt, and getKey	11
Output Statements: Output and Disp	12
Display Control Statements: ClrHome and Pause	14
Conditions	15
Decision Statement: If Then Endif and If Then Else End	15
Loop Statements: For, While, Repeat	16
Menu Statement	19
Variables: Numeric, Strings, and Lists	21
System Variables	22
Clearing the Draw Screen	22
Displaying the Draw Screen	22
PlotsOn PlotsOff Statements	23
FnOn FnOff Statements	23
Configuring a Stat Plot	23
Drawing Statements	23
Sample Program Draws Coordinates of a Point Moving Around the Circumference of a Circle	25

© Copyright by Pyzdrowski 2007, 2009
All rights reserved

Why Write a Program?

Calculator programs enable the user to automate a sequence of steps or calculations. Whenever the user repeatedly performs the same or similar task a program should be considered as a viable option to re-entering the possibly complex sequence of steps or calculations. A program is also useful whenever a complex task is desired to be used by individuals for the results and not necessarily used to demonstrate the ability to enter the complex sequence into the calculator. After the sequence of statements are entered into a program and the program is saved, the sequence of statements can be executed without re-entering the statements each time. Whenever a program does not terminate, pressing the [ON] key will bring up a break menu:

ERR: BREAK

1: Quit

2: Goto

Pressing the [1] key will terminate the program. Pressing the [2] key will open the program in the editor and position the cursor at the statement where the program was terminated.

The program statements must be the statements accessed through the command menus and can not be typed as text. For example the **Input** statement selected from the I/O sub menu of the [PRGM] key is not the same as typing the text INPUT.

The PRGM menu

Programs are executed, edited, and created through the PRGM menu. Access the PRGM menu by pressing the [PRGM] key. Three sub menus are available, EXEC, EDIT, and NEW. When the EXEC or EDIT sub menu is selected, a list of the programs currently in the calculator's memory is displayed. An existing program can be executed or edited by scrolling down to the desired program and pressing [ENTER] or by pressing the number associated to the program.



Figure 1 PRGM menu.

When in the EXEC sub menu and a program is selected to be executed its name will be placed on the display and the calculator is waiting for confirmation by pressing the [ENTER] key.



Figure 2 Confirm Execute.

When in the EDIT sub menu and a program is selected for editing the contents of the program will be displayed. When finished editing a program press [QUIT], [2ND][MODE], to save and close the program.

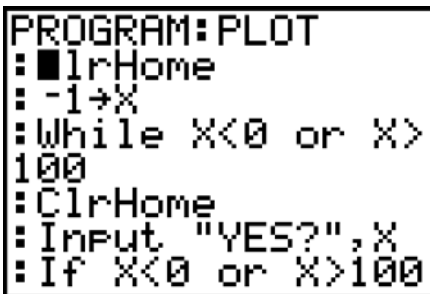


Figure 3 Program Edit.

The NEW sub menu only offers one option, 1: Create New. When Create New is selected with either the [ENTER] or [1] key, the calculator will prompt for a name of the new program.



Figure 4 Program New.



Figure 5 Program Name.

Creating a Program on the Calculator

The Circle Area Program

When creating a program on the calculator, select the program menu with the [PRGM] key. Select NEW with the cursor keys then either press the [1] or the [ENTER] key. Enter a name for this program, the calculator is already set to the ALPHA mode. The program name can be up to eight characters or digits in length containing. The

program name must begin with a character. As an example enter CIRCLEAR for a new program name. This program will calculate the area of a circle given its radius.



Figure 6 New Program Name.

After pressing the [ENTER] key, the program edit screen will be displayed with the program name at top and a cursor waiting for statements to be entered into the calculator.

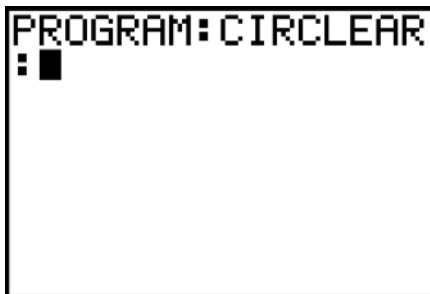


Figure 7 New Circle Area Program.

The statements used in the programs are case sensitive and are best entered from either the PRGM menu or the CATALOG menu.

The equation used to calculate the area of a circle is $A = \pi \times R^2$. The radius, independent variable, is required to calculate the area. One way to prompt the user for the data and to enter the data from the user is to use the **Input** statement. Enter the **Input** statement by pressing the [PRGM] key then scroll over to the I/O sub menu and select **Input** with the [ENTER] key or the [1] key. **Input** will now be displayed on the first line of the program. The first parameter for the **Input** statement is the prompt. The prompt is text that will be displayed on the calculator's screen and the text must be enclosed in quotes. Enter the text "ENTER R:" by pressing:

[ALOCK] with the [2ND][ALPHA] keys, the cursor is a blinking A
["] with the [+] key
[E][N][T][E][R]
[] space with the [0] key
[R]
[:] with the [.] key
["] with the [+] key

The second parameter of the **Input** statement is the variable where the input data will be stored. A comma is used to separate the two parameters of the **Input** statement. Enter **,R** by pressing:

[ALPHA] the cursor changes back to a blinking solid rectangle
 [,]
 [ALPHA] [R]

Complete the **Input** statement by pressing the [ENTER] key. The first line of the program now contains the statement **Input "ENTER R:",R**. This statement will display the text *ENTER R:* on the screen and wait for the user to enter a value. When a number is entered it will be placed into the variable R. The variables on the TI-83/TI-84 calculators can only be single characters.

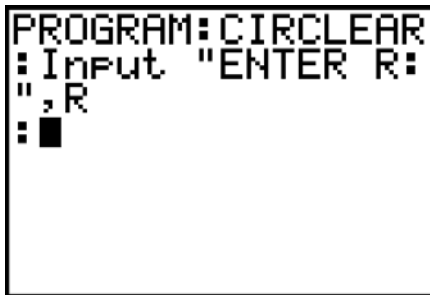


Figure 8 Input Statement.

Now enter the equation which will calculate the area of a circle and store the result into the variable **A**. Enter $\pi * R ^ 2 \rightarrow A$ by pressing:

[π] with the [2ND][\wedge] keys
 [x]
 [ALPHA][R]
 [\wedge]
 [2]
 [STO>] \blacktriangleright
 [ALPHA][A]

Complete the statement by pressing the [ENTER] key. The second line of the program now contains the statement $\pi * R ^ 2 \rightarrow A$. This statement will calculate the area of the circle whose radius is stored in the variable **R** and store the result in the variable **A**.

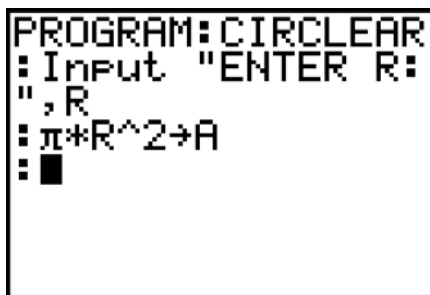


Figure 9 Circle Area Statement.

Now that the area of the circle is calculated, the result needs to be displayed to the user. One way to output information is to use the **Disp**, display, statement. Enter the **Disp** statement by pressing the [PRGM] key then scroll over to the I/O sub menu and select **Disp** with the [ENTER] key or the [3] key. **Disp** will now be displayed on the third line of the program. The parameters for the **Disp** statement follow and are separated by commas. The first parameter will be a string which indicates what the number is, "AREA:". The next and final parameter will be the variable **A**. Enter the text "AREA:", **A** by pressing:

[ALOCK] with the [2ND][ALPHA] keys, the cursor is a blinking A
 ["] with the [+] key
 [A][R][E][A]
 [:] with the [.] key
 ["] with the [+] key

[ALPHA] the cursor changes back to a blinking solid rectangle
 [,]
 [ALPHA] [A]

Complete the **Disp** statement by pressing the [ENTER] key. The third line of the program now contains the statement **Disp "AREA:",A**. This statement will display the text *AREA:* followed by the value stored in the variable **A** on the screen.

```
PROGRAM:CIRCLEAR
:Input "ENTER R:
",R
: $\pi$ *R^2→A
:Disp "AREA:",A
:█
```

Figure 10 Circle Area Display.

Complete the program by pressing [QUIT], [2ND][MODE]. The program will be saved and the calculator will return to the calculation screen. This concludes a very basic program written on the calculator.

Executing the Circle Area Program

To execute the CIRCLEAR program, select [PRGM] and from the EXEC sub menu select the CIRCLEAR program and press [ENTER]. prgmCIRCLEAR will be displayed on the screen. Press [ENTER] to execute the CIRCLEAR program. The prompt ENTER R: will be displayed on the screen. The calculator is now waiting for user input.


```
Pr9mCIRCLEAR
ENTER R:█
```

Figure 11 Circle Area Program Input.

Enter a value for the radius, 2.25, and press [ENTER]. The program will calculate the area of the circle and display the area of 15.90431281.

```
Pr9mCIRCLEAR
ENTER R:2.25
AREA:
      15.90431281
                Done
█
```

Figure 12 Circle Area Area.

Clearing the Display

Sometimes it is desirable to clear the calculator's text, Home, screen before displaying information. The **ClrHome** statement will clear the Home screen and reposition the text cursor at the upper left corner of the display. Edit the CIRCLEAR program by pressing the [PRGM] key and from the EDIT sub menu select the CIRCLEAR program and press the [ENTER] key. Position the cursor on the I of **Input** in the first line of the program, the cursor begins in this position. Change the edit mode to insert by pressing the [INS] key, [2ND][DEL]. When the calculator is in the insert mode the cursor changes from a blinking solid rectangle to a blinking underscore. Enter the **ClrHome** statement by pressing the [PRGM] key and select **ClrHome** from the I/O sub menu, press [ENTER] to add **ClrHome** to the program. Move the Input statement to the second line by pressing the [ENTER] key.

Complete the program by pressing [QUIT], [2ND][MODE]. The program will be saved and the calculator will return to the calculation screen.

```

PROGRAM:CIRCLEAR
:ClrHome
:Input "ENTER R:
",R
: $\pi$ *R^2→A
:Disp "AREA:",A
:█

```

Figure 13 Circle Area Clear Home.

Execute the program again and notice that the prompt to ENTER R: is on a cleared screen and begins on the first row. Whatever information that was previously on the screen has been cleared before the **Input** statement is processed.

```

ENTER R:█

```

Figure 14 Cleared Display Prompt.

Drawing a Circle Program

Begin a new program named CIRCLE. This program will make use of the graphics screen instead of the normal calculator text screen. Begin by clearing the home screen. Prompt the user to enter a radius for the circle. Store this radius into the variable **R**, use the **Input** statement. These statements were used in the previous CIRCLEAR program example. The program contains:

```

:PROGRAM:CIRCLE
:ClrHome
:Input "ENTER R:",R

```

All statistics plots and functions need to be turned off so that any plots or functions will not be displayed along with the circle. The **PlotsOff** statement will turn off all of the statistics plots. Enter the **PlotsOff** statement from the [STAT PLOT] menu, [2ND][Y=], then select **PlotsOff**. The **FnOff** statement will turn off all Y= functions. Enter the FnOff statement from the [VARS], Y-VARS submenu, then select On/Off and finally select **FnOff**. Initially the X and Y axis will be turned on with the **AxesOn** statement. The **AxesOn** statement is selected from [FORMAT], [2ND][ZOOM]. The program now contains:

```

:PROGRAM:CIRCLE
:ClrHome
:Input "ENTER R:",R
:PlotsOff
:FnOff
:AxesOn

```

The graph window parameters need to be defined. All of the window parameters are selected from the [VARS] then Window sub menu. The **Xmin**, **Xmax**, and **Xscl** parameters determine the horizontal window size for the graph and the spacing of the tick marks. The **Ymin**, **Ymax**, and **Yscl** parameters determine the vertical window size for the graph and the spacing of the tick marks. The parameter **Xres** should be set to 1 for normal graphing purposes. Enter the following assignment statements into the program.

```

:-1.5*R→Xmin
:1.5*R →Xmax
:1→Xscl
:-1.5*R→Ymin
:1.5*R →Ymax
:1→Yscl
:1→Xres

```

The parameters of the graph screen are now set. Next the graph screen should be cleared to remove any old graphical information that was previously on the graph screen. The **ClrDraw** statement is selected from the [DRAW], [2ND][PRGM] menu. The statement that will draw the circle of a specified radius whose center is specified at a specific x, y location is **Circle** selected from the [DRAW], [2ND][PRGM] menu. The first parameter of the **Circle** statement specifies the x location of the center of the circle. The second parameter specifies the y position of the center of the circle. The x and y location are with respect to the graph coordinates and are not screen locations. The third parameter specifies the radius of the circle. Enter the statement **Circle(0,0,R)**. This will draw a circle of radius R centered on the x and y axis. After the circle is drawn, the calculator program can wait until the user presses the [ENTER] key by using the **Pause** statement. The **Pause** statement is selected from the [PRGM] menu. After the user presses the [ENTER] key, the processing will continue. Turn the axis off with the **AxisOff** statement located under the [FORMAT] menu. Redraw the circle because turning the axis on/off will clear the graph screen. After the circle is drawn, draw a line segment from the origin to the edge of the circle along the x axis. Use the **Line** statement which is selected from the [Draw], [2ND][PRGM], menu. The **Line** statement has four parameters. The first two parameters are the starting x, y location of the line. The second two parameters are the ending x, y location of the line. Draw the **Line** from location **0,0** to location **R,0** with **Line(0,0,R,0)**. Pause the program after the circle and the radius line are drawn. Finally turn the axis on with the **AxisOn** statement. The final program is:

```

:PROGRAM:CIRCLE
:ClrHome
:Input "ENTER R:",R
:PlotsOff
:FnOff
:AxesOn

```

```

:-1.5*R → Xmin
:1.5*R → Xmax
:1→Xscl
:-1.5*R → Ymin
:1.5*R → Ymax
:1→Yscl
:1→Xres
:ClrDraw
:Circle(0,0,R)
:Pause
:AxesOff
:Circle(0,0,R)
:Line(0,0,R,0)
:Pause
:AxesOn

```

Run the CIRCLE program. The program will prompt for a radius. After entering a radius and pressing the [ENTER] key, graph screen will contain the x and y axis and draw the circle of your specified radius.

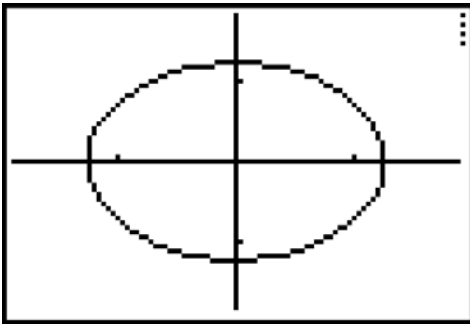


Figure 15 Circle drawn by the program.

After pressing the [ENTER] key, in response to the **Pause**, the graph screen will be cleared, and the circle will be re-drawn with a radius line going from the center of the circle to the edge of the circle along the x axis. No x or y axis will be drawn so the radius line can be seen.

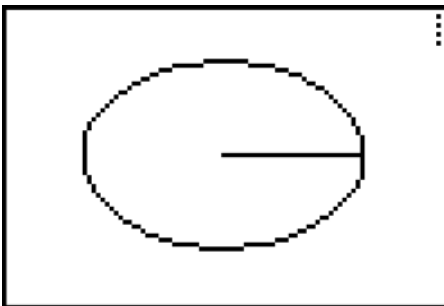


Figure 16 Circle and radius line drawn by the program.

Pressing the [ENTER] key a second time will clear the display and display the x and y axis. Return to the home screen by pressing the [CLEAR] key.



Figure 17 Final graph screen of the CIRCLE program.

Notice the circle is not displayed as a symmetrical circle, it appears as an ellipse. The reason for this distortion is the graph screen is 95 pixels, dots, wide and 63 pixels high, rectangular not square, and the x and y minimums and maximums were set to the same value, ± 1.5 . This distortion can be corrected by setting the domain and range to be proportional to the screen width and height. The screen proportion is 1.5 to 1. Change the **Ymin** to -1.0 and **Ymax** to 1.0 to match the proportions with the **Xmin** and **Xmax** remaining at ± 1.5 . The modified program will draw a circle now and not an ellipse.

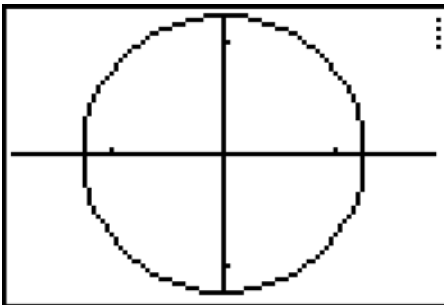


Figure 18 Circle drawn with axis proportional to screen size.

Program Accessing Other Programs: prgm

Often small programs are written which perform a specific task, a module, that will be used many times by a variety of programs. Larger programs can access, call, these programs. The **prgm** statement followed by the name of the program will call the program. When the called program is finished, processing returns and continues with the next statement in the larger program. This statement is accessed from the [PROG], CTL menu. The [PROG] menu can only be accessed when editing a program.

To prompt the user for an age, read the age, and return the age through the variable A, a subprogram:

```
:PROGRAM:GETAGE
```

```
[PRGM], NEW,  
1:Create New, [G],
```

:ClrHome	[E], [T], [A], [G], [E], [ENTER] [PRGM], I/O,
:Input "ENTER AGE:",A	8:ClrHome, [ENTER] [PRGM], I/O, 1:Input, [2ND][ALPHA], ["], [E], [N], [T], [E], [R], [], [A], [G], [E], ["], [ALPHA], [], [ALPHA], [A], [ENTER]
:Return	[PRGM], CTL, E:Return, [ENTER]

A main program that calls the program GETAGE and then displays the data retrieved by GETAGE.

:PROGRAM:PROG	[PRGM], NEW, 1:Create New, [P], [R], [O], [G], [ENTER]
:prgm GETAGE	[PRGM], CTL, D:prgm, [2ND][ALPHA], [G], [E], [T], [A], [G], [E], [ENTER]
:Disp "AGE IS ", A	[PRGM], I/O, 3:Disp, [2ND][ALPHA], ["], [A], [G], [E], [], [I], [S], ["], [ALPHA], [], [ALPHA], [A], [ENTER]
:Stop	[PRGM], CTL, F:Stop, [ENTER]

Executing the program PROG will call the program GETAGE. The program GETAGE will prompt for and read an age then store the age in the variable A. The program GETAGE returns control to the calling program PROG. The program PROG displays the message AGE IS with the value entered into the GETAGE program.

Useful Programming Statements

The following are some useful programming statements and examples.

Program Termination Statements: Stop and Return

There are two statements that can be used to properly terminate a program: **Stop** and **Return**. The **Stop** Statement is the proper way to terminate a program. When the **Stop** statement is encountered, processing will terminate and the current screen will be displayed. The same will happen if no terminating statement appears at the end of the program. The **Return** statement is used when the program is intended to be used by another program, a subprogram. The **Return** statement will transfer control back to the program that accessed, called, this program. These statements are accessed from the [PROG], CTL menu. The [PROG] menu can only be accessed when editing a program.

Stop	[PRGM], CTL, F:Stop
Return	[PRGM], CTL, E:Return

Input Statements: Input, Prompt, and getKey

There are two data input statements: **Input** and **Prompt**. Both statements are used to enter data into a program from the user. The **Input** statement has three variations. The **Input** statement without any parameters will display the graph screen. **Input** followed by a variable, **Input X**, will display a question mark in the first column of the next row and wait for data to be entered followed by the [ENTER] key. A prompt can be included in the third form, **Input "ENTER X ",X**. The prompt ENTER X will be displayed at the beginning of the next row on the home screen. The statement will wait until the user enters data and presses the [ENTER] key. The **Prompt** statement enables the user to enter a list of data. Unlike the **Input** statement, a message can not be included with the **Prompt** statement. The program will display a question mark, ?, at the beginning of the next line on the home screen and wait for the user to enter data followed by the [ENTER] key. The process will be repeated for each parameter listed with the **Prompt** statement. These statements are accessed from the [PROG], I/O menu. The [PROG] menu can only be accessed when editing a program.

To display the graph screen and enable the coordinates and cross hairs:

Input [PRGM], I/O, 1:Input

To prompt for and read data into the variable N:

Input N [PRGM], I/O, 1:Input, [ALPHA] ,[N]

To prompt for N and read data into the variable N:

Input "ENTER N", N [PRGM], I/O, 1:Input, [2ND],[ALPHA],
["],[E],[N],[T],[E],[R],[],[N],["],[ALPHA],
[,]],[ALPHA],[N]

To prompt for and read data into the variable N:

Prompt N [PRGM], I/O, 2:Prompt, [ALPHA],[N]

To prompt for and read data into the variables X, Y, and Z:

Prompt X,Y,Z

[PRGM], I/O, 2:Prompt, [ALPHA],[X],
[,],[ALPHA],[Y],[,],[ALPHA],[Z]

Another type of input statement is **getKey**. This statement reads the key code of the key pressed. Processing does not stop for the **getKey** statement as it does for the data input statements. Instead, the keys are read, if no key is pressed a zero, 0, is returned and the processing continues. When a **getKey** is processed and a key is pressed the key code corresponding to the key is returned. The keys are coded by row and column. The row is the tens position of the code and the column is the units position of the code. The rows and columns start with one. For example, the [Y=] key is code 11, row 1 column 1. The [DEL] key is 23, the [STO] key is 91, the [1] key is 92, the [+] key is 95, and the [ENTER] key is 105. In order to identify the shifted keys like [LIST], the two key sequence must be identified of [2ND][STAT] or 21 then 33. The only key that **getKey** does not return a code for is the [ON] key. The position of the [ON] key would be 101; but, the [ON] key always terminates the program for the ERR:BREAK 1:Quit 2:Goto menu. This statement is accessed from the [PROG], I/O menu. The [PROG] menu can only be accessed when editing a program.

To read the key code into the variable K:

getKey → K

[PRGM], I/O, 7:getKey, [STO], [ALPHA],
[K]

Output Statements: Output and Disp

The home screen contains character positions of eight rows of sixteen columns. The output will use these character positions. There are two output statements: **Output** and **Disp**. The **Output** statement places the desired information starting at a specified row and column position. The information can be either text or any of the data types i.e. a number, a string, a list, etc. The **Disp** statement starts at the current cursor position and will display a sequence of text and/or data. The following grid shows the layout of the home screen. These statements are accessed from the [PROG], I/O menu. The [PROG] menu can only be accessed when editing a program.

1																
2																
3																
4																
5																
6																
7																
8																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

To display the text starting at a specific row and column position on the screen, i.e. HELLO starting in row 2 of column 4:

Output(2, 4, "HELLO")

[PRGM] , I/O, 6:Output
[2],[,],[4],[,],[2ND][ALPHA]
["][H][E][L][L][O]["]
[ALPHA][)]

To display the value of N at a specific row and column position on the screen, i.e. N starting at row 3, of column 4:

Output(3,4,N)

[PRGM] , I/O, 6:Output
[3][,][4][,][ALPHA][N]
[ALPHA][)]

To display the text starting at the current cursor position on the screen, i.e.
HELLO:

Disp "HELLO"

[PRGM], I/O, 3:Disp, [2ND],
[ALPHA], ["], [H], [E], [L],
[L], [O], ["]

To display the value stored in the variable N:

Disp N

[PRGM], I/O, 3:Disp,
[ALPHA], [N]

To display the test VALUE OF N: followed by the value stored in the variable N:

Disp "VALUE OF N", N

[PRGM], I/O, 3:Disp, [2ND],
[ALPHA], ["], [V], [A], [L],
[U], [E], [], [O], [F], [], [N],
["], [ALPHA], [,], [ALPHA],
[N]

To display a sequence of parameters of text or variables:

Disp "VALUE 1", X,"VALUE 2",Y, "VALUES 3 AND 4", A,B

To display the Home screen:

Disp

[PRGM], I/O, 3:Disp

Display Control Statements: ClrHome and Pause

The **ClrHome** statement clears the home screen and places the cursor back to the first column of the first row.

To clear the Home display:

ClrHome

[PRGM], I/O, 8:ClrHome

The **Pause** statement stops processing and waits for the user to press the [ENTER] key. There are two forms of the **Pause** statement. The **Pause** statement without a text parameter will stop processing and wait for the [ENTER] key to be pressed before resuming program execution. The **Pause** statement with a text string will stop processing, display the text string starting at the beginning of the next row, and wait for the [ENTER] key to be pressed before resuming program execution.

These statements are accessed from the [PROG], I/O and CTL menus. The [PROG] menu can only be accessed when editing a program.

To pause for the ENTER key to be pressed without a message on the screen:

Pause

[PRGM], CTL, 8:Pause

To pause for the ENTER key to be pressed with a message on the screen:

Pause "Press ENTER to continue"

[PRGM], CTL, 8:Pause

Conditions

Logical conditions are used to control many programming statements. The logical conditions evaluate either true or false. Relational operators are used to translate numeric data into a logical result. There are six relational operators: =, ≠, >, ≥, <, and ≤. The relational operators are located under the [TEST], [2ND][MATH], menu's TEST submenu. Multiple relational operations can be connected with the logical operations of **and**, **or**, **xor**, and **not**. The logical operations are located under the [TEST], [2ND][MATH], menu's LOGIC submenu.

To test the contents of X to be less than 5:

X < 5

[ALPHA], [X], [TEST]
([2ND][MATH]), TEST, 5:<,
[5]

To test the contents of X to be less than 5 and greater than 0:

X < 5 and X > 0

[ALPHA], [X], [TEST]
([2ND][MATH]), TEST, 5:<,
[5], [TEST]
([2ND][MATH]), LOGIC,
1:and, [ALPHA], [X],
[TEST] ([2ND][MATH]),
TEST, 3:>, [0]

To test the contents of X to be less than 5 and greater than 0 is not true:

not(X < 5 and X > 0)

[TEST] ([2ND][MATH]),
LOGIC, 4:not ([ALPHA],
[X], [TEST]
([2ND][MATH]), TEST, 5:<,
[5], [TEST]
([2ND][MATH]), LOGIC,
1:and, [ALPHA], [X],
[TEST] ([2ND][MATH]),
TEST, 3:>, [0], [)]

Decision Statement: If Then Endif and If Then Else End

Decision statements are used to conditionally process a set of statements based on the outcome of a logical condition. There are two forms of the decision statement **If**. The simplest form is the **If condition Then statements... End** where **condition** is any expression the results in true or false and **statements...** are any number of programming statements. The statement **Then** separates **condition** from the **statements...** The statement **End** terminates the block of statements associated with the **Then** statement. The **statements...** will be processed only when the **condition** evaluates to true. When the **condition** evaluates false no statements are processed and processing continues with the statement following the **End** statement.

The more complete form of the decision statement is the **If condition Then statements... Else statements... End**. The first part is the same as the **If condition Then statements... End** form. What is added is the **Else statements... End** portion. When the **condition** evaluates true the **statements...** between the **Then** and **Else** are processed. When the **condition** evaluates false the **statements...** between the **Else** and **End** are processed. The decision statements are accessed from the [PROG], CTL menu. The [PROG] menu can only be accessed when editing a program.

To create a decision statement which displays OK if N is greater than 10 and displays NOT OK otherwise, i.e. N is less than or equal to 10:

If N > 10	[PRGM], CTL, 1:If, [ALPHA], [N], [TEST] ([2ND][MATH]), TEST, 3:>, [1], [0][ENTER]
Then	[PRGM], CTL, 2:Then, [ENTER]
Disp "OK"	[PRGM], I/O, 3:Disp, [2ND], [ALPHA], ["], [O], [K], ["], [ENTER]
Else	[PRGM], CTL, 3:Else, [ENTER]
Disp "NOT OK"	[PRGM], I/O, 3:Disp, [2ND], [ALPHA], [N], [O], [T], [], [O], [K], ["], [ENTER]
End	[PRGM], CTL, 7:End, [ENTER]

Loop Statements: For, While, Repeat

Loop statements are used when a task is desired to be performed multiple times. There are three types of loops. The **For** loop, the **While** loop, and the **Repeat** loop. The loop statements are accessed from the [PROG], CTL menu. The [PROG] menu can only be accessed when editing a program.

The **For** loop uses a counter variable which starts at a specified start value and continues to loop as long as the variable is less than or equal to the ending value. When the counter variable is less than or equal to the ending variable, the statements in the body of the loop will be processed. After processing the statements the counter variable will have the specified step added to the counter. After the counter variable is updated, processing loops back to the beginning of the **For** loop and the test is evaluated in order to determine if the body statements will be processed again or not. When the test fails, the loop will terminate and the processing will continue with the first statement after the **For's End** statement. When no step value is specified, the step on one is used. When the step value is negative, the **For** statement will process the statements in the body of the loop as long as the counter variable is greater than or equal to the end value. This type of loop is a pre-test loop since the condition is tested before the body of the loop. A pre test loop has the ability to loop zero or more times. It has the form of **For (var, start, end, step) Statements... End**

To display the values from 0 through 5:

For (X, 0, 5)

Disp X

End

[PRGM], CTL, 4:For,
[ALPHA], [X], [,],
[0], [,], [5], [)],
[ENTER]
[PRGM], I/O, 3:Disp,
[ALPHA],
[X],[ENTER]
[PRGM], CTL,
7:End, [ENTER]

To display the values from 5 through 0:

For (X, 5, 0, -1)

Disp X

End

[PRGM], CTL, 4:For,
[ALPHA], [X], [,],
[5], [,], [0], [,], [(-)],
[1], [)], [ENTER]
[PRGM], I/O, 3:Disp,
[ALPHA],
[X],[ENTER]
[PRGM], CTL,
7:End, [ENTER]

To display the values from 8 through 16 in 2's:

For (X, 8, 16, 2)

Disp X

End

[PRGM], CTL, 4:For,
[ALPHA], [X], [,],
[8], [,], [1], [6], [,],
[2], [)], [ENTER]
[PRGM], I/O, 3:Disp,
[ALPHA],
[X],[ENTER]
[PRGM], CTL,
7:End, [ENTER]

The **While** statements is also a pre test loop. The condition of the loop is tested before the possible processing of the body statements. When the condition evaluates true, the statements will be processed. At the end of the body, after all statements have been processed, the processing will loop back to the beginning of the **While** statement and the condition will be evaluated in order to determine if the body will be processed. When the condition evaluates false, the loop will terminate and the processing will continue with the first statement after the **While's End** statement. The **While** statement requires the condition to be initiated before the loop begins. The **While** statement also requires the condition to be updated before the statement loops back to the condition. Like the **For** statement, the **While** statement is a pre-test loop. It has the form of **While (condition) statements... End**.

To display the values from 0 through 5:

0 → X

While (X ≤ 5)

[0], [STO], [ALPHA],
[X], [ENTER]
[PRGM], CTL,
5:While, [(],
[ALPHA], [X],
[TEST] ([2ND],

Disp X

X + 1 → X

End

[MATH]), TEST,
6: ≤, [5], [)],
[ENTER]
[PRGM], I/O, 3:Disp,
[ALPHA],
[X],[ENTER]
[ALPHA], [1], [+],
[1], [STO], [ALPHA],
[X], [ENTER]
[PRGM], CTL,
7:End, [ENTER]

To continue to read the keys until a key is pressed, then display the code of the key, this will be made as a subprogram:

PROGRAM:KEY

getKey → K

While (K = 0)

getKey → K

End

Disp "KEY CODE", K

Return

[PRGM], NEW,
1:Create New, [K],
[E], [Y], [ENTER]
[PRGM], I/O,
7:getKey, [STO],
[ALPHA], [K],
[ENTER]
[PRGM], CTL,
5:While, [(],
[ALPHA], [K],
[TEST]
([2ND])[MATH]), 1:=,
[0], [)], [ENTER]
[PRGM], I/O,
7:getKey, [STO],
[ALPHA], [K],
[ENTER]
[PRGM], CTL,
7:End, [ENTER]
[PRGM], I/O, 3:Disp,
[2ND], [ALPHA], ["],
[K], [E], [Y], [], [C],
[O], [D], [E], ["],
[ALPHA], [,],
[ALPHA], [K],
[ENTER]
[PRGM], CTL,
E:Return, [ENTER]

The **Repeat** statements is a post test loop. The condition of the loop is tested after processing the body statements before processing loops back to the beginning. Think of the **Repeat** as a Repeat until the condition evaluates true. When the condition evaluates false, the **Repeat** will loop and the statements will be processed. When the condition is true, the loop will terminate and the processing will continue with the first statement after the **Repeat's End** statement. Since the **Repeat** is a post test loop, the condition only needs to be updated before the end of the loop where the condition will be

evaluated. Initialization of the condition before the loop is still a good idea. It has the form of **Repeat (condition) statements... End**.

To display the values from 0 through 5:

0 → X

Repeat (X > 5)

Disp X

X + 1 → X

End

[0], [STO], [ALPHA],
[X], [ENTER]
[PRGM], CTL,
6:Repeat, [(],
[ALPHA], [X],
[TEST ([2ND],
[MATH]), TEST, 3:>,
[5], [)], [ENTER]
[PRGM], I/O, 3:Disp,
[ALPHA],
[X],[ENTER]
[ALPHA], [1], [+],
[1], [STO], [ALPHA],
[X], [ENTER]
[PRGM], CTL,
7:End, [ENTER]

To continue to read the keys until a key is pressed, then display the code of the key until the key pressed was the [ENTER] key. This program will call the KEY subprogram:

PROGRAM:KEYS

Repeat (K = 105)

prgm KEY

End

Stop

[PRGM], NEW,
1:Create New, [K],
[E], [Y], [S],
[ENTER]
[PRGM], CTL,
6:Repeat, [(],
[ALPHA], [K],
[TEST]
([2ND][MATH]), 1:=,
[1], [0], [5],[ENTER]
[PRGM], CTL,
D:prgm,
[2ND][ALPHA], [K],
[E], [Y], [ENTER]
[PRGM], CTL,
7:End, [ENTER]
[PRGM], CTL,
F:Stop, [ENTER]

Menu Statement

The **Menu** statement is used to create up to a seven line text selection menu. The menu contains a title and up to seven pairs of menu text items with associated labels. Menus are typically used when a selection of choices are needed to be presented to the user. Additional statements are needed in order for the **Menu** statement to work. They are the **Lbl**, label, statement and the **Goto**, go to, statement. The **Goto** statement allows a

programmer to develop unstructured code and it is strongly encouraged not to use the **Goto** statements in programs. However; the **Menu** is structured in a way that the selection will transfer control to the specified label, **Lbl**. Once processing enters the statements associated with a label, **Lbl**, the only way to branch around the next block of statements beginning with a different label, **Lbl**, is to use the **Goto** statement. The **Menu** statement only processed valid selection entries.

The **Lbl** statement requires a label of one or two characters. This label becomes an entry point for processing whenever there is a branch to the specified label. The branch can be from the **Menu** statement or **Goto** statements. The **Goto** statement requires a label of one or two characters. Processing will be branched to the corresponding **Lbl** statement with the specified one or two character label. Again, Goto's allow for unstructured programs which are difficult to follow and sometimes prone to logical errors. These statements are accessed from the [PROG], CTL menu. The [PROG] menu can only be accessed when editing a program.

To create a menu which enables the user to read and display a number, read and display a string, or quit:

```

PROGRAM:MENU                                [PRGM], NEW, 1:Create
                                                New, [M], [E], [N], [U],
1 → X                                         [ENTER]
                                                [1], [STO], [ALPHA], [X],
While (X = 1)                                [ENTER]
                                                [PRGM], CTL, 5:While, [(],
                                                [ALPHA], [K], [TEST]
                                                ([2ND])[MATH]), 1:=, [1],
                                                [)], [ENTER]

Menu("MY MENU", "NUMBER", NU, "STRING", ST, "QUIT",
QU)
                                                [PRGM], CTL, C:Menu,
                                                [2ND],[ALPHA], ["], [M],
                                                [Y], [ ], [M], [E], [N], [U],
                                                ["], [ALPHA], [,],
                                                [2ND],[ALPHA], ["], [N],
                                                [U], [M], [B], [E], [R], ["],
                                                [ALPHA], [,], [2ND],
                                                [ALPHA], [N], [U], [,],
                                                [2ND],[ALPHA], ["], [S],
                                                [T], [R], [I], [N], [G], ["],
                                                [ALPHA], [,], [2ND],
                                                [ALPHA], [S], [T], [,],
                                                [2ND],[ALPHA], ["], [Q],
                                                [U], [I], [T], ["], [ALPHA],
                                                [,], [2ND], [ALPHA], [Q],
                                                [U], [)], [ENTER]
Lbl NU                                       [PRGM], CTL, 9:Lbl, [2ND],
                                                [ALPHA], [N], [U],
                                                [ENTER]

```

Input "ENTER NUM ", N	[PRGM], I/O, 1:Input, [2ND],[ALPHA], ["],[E],[N],[T],[E],[R],[],[N],[U],[M],[],["], [ALPHA],[,],[ALPHA],[N], [ENTER]
Disp "NUM ", N	[PRGM], I/O, 3:Disp, [2ND], [ALPHA],["],[N],[U],[M], [],["],[ALPHA],[,], [ALPHA],[N],[ENTER]
Goto E	[PRGM], CTL, 0:Goto, [ALPHA],[E],[END]
Lbl ST	[PRGM], CTL, 9:Lbl, [2ND], [ALPHA],[S],[T],[ENTER]
Input "ENTER STR ", Str1	[PRGM], I/O, 1:Input, [2ND],[ALPHA], ["],[E],[N],[T],[E],[R],[], [S],[T],[R],[],["], [ALPHA],[,],[VAR], 7:String..., 1:Str1, [ENTER]
Disp "STR ", Str1	[PRGM], I/O, 3:Disp, [2ND], [ALPHA],["],[N],[U],[M], [],["],[ALPHA],[,], [VAR], 7:String..., 1:Str1, [ENTER]
Goto E	[PRGM], CTL, 0:Goto, [ALPHA],[E],[END]
Lbl QU	[PRGM], CTL, 9:Lbl, [2ND], [ALPHA],[Q],[U], [ENTER]
0 → X	[0],[STO],[ALPHA],[X], [ENTER]
Lbl E	[PRGM], CTL, 9:Lbl, [2ND], [ALPHA],[E],[ENTER]
Pause "PRESS ENTER"	
End	[PRGM], CTL, 7:End, [ENTER]
Stop	[PRGM], CTL, F:Stop, [ENTER]

Variables: Numeric, Strings, and Lists

Numeric variables can only have single character names A through Z. The variables are specified with [ALPHA] [character].

Strings can only be stored in ten system variables names Str0 through Str9. The system string variables are accessed through the [VAR] 7:String... menu.

Lists can be stored in the six system lists, L1 through L6, or through a user specified name of up to five characters. When referencing the system lists L1 through L6, they can not be entered as the character L followed by the digit. Instead, the system lists must be referenced by the L1 through L6 located on the keypad. To specify L1 the [2ND][1] keys would be pressed. When a user specified name is used to store a list, only the name is required. When a user specified list is used to access a list, the name must be preceded with the \lfloor symbol found in [LIST], OPS, B: \lfloor .

System Variables

System variables are available to the programmer. The system variables can be read or changed. The system variables are located under the [VARS] menu. Some system variables that are commonly used when graphing are: Xmin, Xmax, Ymin, Ymax, Xscl, Xres, and Yscl.

To set the graph window to for a plot of the data in L_1 and L_2 :

min(L_1) \rightarrow Xmin	[MATH], NUM, 6:min, [(], [2ND], [1], [)], [STO], [VARS], VARS, 1:Window, 1:Xmin, [ENTER]
max(L_1) \rightarrow Xmax	[MATH], NUM, 7:max, [(], [2ND], [1], [)], [STO], [VARS], VARS, 1:Window, 2:Xmax, [ENTER]
1 \rightarrow Xscl	[1], [STO], [VARS], VARS, 3:Xscl, [ENTER]
1 \rightarrow Xres	[1], [STO], [VARS], VARS, 7:Xres, [ENTER]
min(L_2) \rightarrow Ymin	[MATH], NUM, 6:min, [(], [2ND], [2], [)], [STO], [VARS], VARS, 1:Window, 4:Ymin, [ENTER]
max(L_2) \rightarrow Ymax	[MATH], NUM, 7:max, [(], [2ND], [2], [)], [STO], [VARS], VARS, 1:Window, 5:Ymax, [ENTER]
1 \rightarrow Yscl	[1], [STO], [VARS], VARS, 6:Yscl, [ENTER]

Clearing the Draw Screen

Sometimes it is desirable to clear the calculator's draw screen before making a new graph or drawing. The **ClrDraw** statement will clear the draw screen. The **ClrDraw** statement is located by pressing the [DRAW] key and select **ClrHome** from the DRAW sub menu.

Displaying the Draw Screen

The statement **DispGraph** switches to the draw screen which displays the graph, plot, or drawing. The **DispGraph** statement is located in the [PROG], I/O, 4:DispGraph menu. The [PROG] menu can only be accessed when editing a program.

PlotsOn PlotsOff Statements

The three Stat Plots can be turned on or off and configured from a program. The **PlotsOn** and **PlotsOff** statements are used to turn the Stat Plots on or off either as a group or individually. The [STAT PLOT] menu can only be accessed when editing a program.

To turn on Stat Plot 1:

PlotsOn 1

[STAT PLOT], PLOTS,
5:PlotOn, [1]

To turn off Stat Plot 2:

PlotsOff 2

[STAT PLOT], PLOTS,
4:PlotOff, [2]

To turn off all Stat Plots:

PlotsOff

[STAT PLOT], PLOTS,
4:PlotOff

FnOn FnOff Statements

The ten Functions, Y=, can be turned on or off and configured from a program. The **FnOn** and **FnOff** statements are used to turn the Functions on or off either as a group or individually.

To turn on Y1= :

FnOn 1

[VARS], Y-VARS, 4:On/Off,
1:FnOn, [1]

To turn off Y2= :

FnOff 2

[VARS], Y-VARS, 4:On/Off,
2:FnOff, [2]

To turn off all Y= Functions:

FnOff

[VARS], Y-VARS, 4:On/Off,
1:FnOn,

Configuring a Stat Plot

The three Stat Plots can be configured from a program. A Stat Plot statement has the form: Plot#(type, xlist, ylist, mark). The # can be 1, 2, or 3. From a program edit, the Plot information is accessed from the [STAT PLOT] menu. The [STAT PLOT] menu can only be accessed when editing a program. The submenus of [STAT PLOT] include PLOTS, TYPE, and MARK. From the submenus, the different plots, types of plots, and marks can be selected. The [STATS PLOT] menu can only be accessed when editing a program.

To configure Plot 1 to be a Scatter Plot of Lists 1 and 2 using a * for the mark:

Plot 1(Scatter,L₁,L₂,')

[STAT PLOT], PLOTS,
1:Plot1(, [STAT PLOT],
TYPE, 1:Scatter, [,], [2ND],
[1], [,], [2ND], [2], [,],
[STAT PLOT], MARK, 3:*,
[]]

Drawing Statements

The draw screen contains 95 pixels horizontally and 63 pixels vertically. This is approximately a 1.5:1 ratio. The pixels are numbered from 0, 0 to 94, 62 where 0, 0 is the upper left corner and 94, 62 is the lower right corner. Most of the drawing statements, except for pixel statements, use X and Y locations. The X and Y locations are with respect to the WINDOW parameters and not a physical pixel location. If you do not want the drawings to be distorted, the distance between Xmax and Xmin needs to be 1.5 times the distance between Ymax and Ymin. The values used in the draw statements can be real numbers and are not restricted to integer values. The calculator will scale the value to the closest pixel given the current WINDOW settings.

There are several statements used to draw on the draw screen. A few typical ones are **Line**, **Circle**, **Pt-On**, **Pt-Off**, **Text**, **Pxl-On**, and **Pxl-Off**. All of these statements are accessed from the [DRAW] menu.

The **Line** statement has the form of **Line (X1, Y1, X2, Y2)**. The X1 and Y1 identify the starting point of the line and X2, Y2 identify the ending point of the line. These values are with respect to the current WINDOW parameters.

The **Line** statement has a second form of **Line (X1, Y1, X2, Y2, 0)**. This functions the same except the line is erased instead of drawn.

The **Circle** statement has the form of **Circle(X, Y, radius)**. The X, Y point identifies the center of the circle. The circle will be drawn with the specified radius. These values are with respect to the current WINDOW parameters.

The **Pt-On** statement has the form of **Pt-On(X, Y)**. The point located at X, Y will be turned on. These values are with respect to the current WINDOW parameters.

The **Pt-Off** statement has the form of **Pt-Off(X, Y)**. The point located at X, Y will be turned off. These values are with respect to the current WINDOW parameters.

The **Text** statement is used to draw text on the draw screen. It has the form **Text(row, column, text1, text2, ...,textn)**. The upper left of the first character of text1 begins at the specified row, column pixel location. When several text are included, they continue immediately after the end of the previous text. These values are with respect to the 95 by 63 pixel grid starting in the upper left corner. Notice this statement uses row, column and not X, Y. The row, column and X, Y references are reversed, i.e. row is Y and column is X.

The **Pxl-On** statement has the form of **Pxl-On(row, column)**. The pixel located at row, column will be turned on. These values are with respect to the 95 by 63 pixel grid starting in the upper left corner. Notice this statement uses row, column and not X, Y. The row, column and X, Y references are reversed, i.e. row is Y and column is X.

The **Pxl-Off** statement has the form of **Pxl-Off(row, column)**. The pixel located at row, column will be turned on. These values are with respect to the 95 by 63 pixel grid starting in the upper left corner. Notice this statement uses row, column and not X, Y. The row, column and X, Y references are reversed, i.e. row is Y and column is X.

```
PROGRAM:DRW
PlotsOff
FnOff
AxesOn
-15*R → Xmin
15*R → Xmax
1→Xscl
-10*R → Ymin
10*R → Ymax
1→Yscl
1→Xres
ClrDraw
Circle(0,0,2)
Line(-5, -5, 5, 5)
Text (2, 2, "TEST")
For (X, 10, 50)
Pxl-On(X, X)
End
DispGraph
Stop
```

Sample Program Draws Coordinates of a Point Moving Around the Circumference of a Circle

This program draws a circle of radius R and draws the triangle formed by a point moving around the circumference of the circle in D degree steps. The coordinates of the point are also displayed.

CIRCLES2

```
ClrHome
Input "RADIUS OF CIRCLE: ",R
Input "STEP ANGLE: ",S
Input "DELAY ",T
PlotsOff
FnOff
-1.5*R →Xmin
1.5*R →Xmax
1 →Xscl
-1.0*R →Ymin
```

```

1.0*R →Ymax
1 →Yscl
1 →Xres
AxesOff
ClrDraw
Degree
0 →A
0 →Y
R →X
Circle(0,0,M)
While getKey=0
Line(0,0,X,Y,0)
Line(X,Y,X,0,0)
Line(0,0,X,0,0)
R*sin(A)→Y
R*cos(A)→X
Line(0,0,X,Y)
Line(X,Y,X,0)
Line(0,0,X,0)
Text(57,0,"X=",X,"Y=",Y)
S+A →A
For(I,0,T)
End
End
Disp
Stop

```

