Unit 4: Loops **Application: Warning Notices**

This Application (WARNING NOTICES) makes use of loops to enter as many values as needed and, as an extension, checks for valid values entered and uses **If** statements to display an appropriate message.

Objectives:

- Learn about Counter and Accumulator statements.
- Use a loop in a program to get an undetermined amount of data.
- Use a 'flag' value to terminate a loop.

Teacher Tip: This project illustrates the tendency toward complexity when developing a piece of software. *Nesting* relates to the idea of putting one control structure inside another. As explained below, the OS knows which End goes with which statement.

Nested Structures

- *Nesting* is the programming technique of placing one control statement inside another. The term is derived from the idea of placing one cardboard box inside another in order to save space.
- It's important to put one *complete* structure *completely* inside a block of another in order to avoid errors.
- The program listing at the right shows an **If** structure inside the **Else** block of another **If** structure. Notice the multiple uses of the **End** statement; the computer knows which **End** belongs with which **If**.
- The indenting is for instructional purposes only. You cannot indent lines in TI-Basic.
- The program first tests to see if **A<0**. If it is, the program displays "A is negative!" and nothing else. But when **A** is not negative then the **underlined** square root calculation, another **If** statement, and the **Disp** statement are all executed.
- The programmer places **If**s inside loops and loops inside **If**s to accomplish more complex tasks as the program requires.

```
prgmSQUARE
Prompt A
If A<0
Then
    Disp "A is negative!"
Else
    √(A)→S
    If S=int(S)
    Then
      Disp "A is a perfect square."
    Else
      Disp "A is not a perfect
square."
    End
    Disp "Its square root is",S
End
```

Summary of the three loops:

For(*var*, *start*, *end***)**	**While** *condition is true*	**Repeat** *until condition is true*
End	**End**	**End**

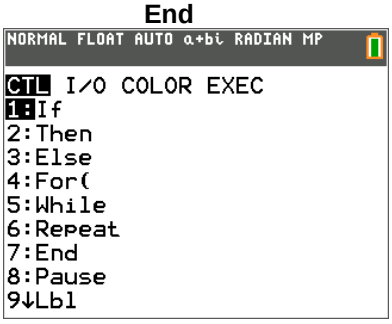
For(is used when 'counting' or processing an arithmetic sequence of values (iteration).
While is used when you might be able to skip the loop body completely.
Repeat is used when you are certain that you want the loop body to run at least once.

About End

The keyword End is used for all the multi-line control structures:

If …	**If …**	**For(…)**	**While …**	**Repeat …**
Then	**Then**			
	Else			
End	**End**	**End**	**End**	**End**

And this is why the $\frac{1}{4}$ CTL menu is arranged the way it is:



7: End comes after the first six CTL menu items because it 'ends' each of those programming control structures.

End statements can appear many times in a program, and the computer knows how the **End**s are connected to their control structures.

Unit 4 Application: Program "Warning Notices"

Most schools send out regular progress reports based on a student's current average. An average of 60% or higher is considered passing, but if the average is below 70% then the student is considered 'in danger' or 'marginal'.

Let's write a program that lets the user enter some test grades, determines the average, then Outputs the number of grades entered, the average of the grades, and an appropriate warning message to the user. The warning messages can be: "Passing", "Marginal", and "Failing".

We can use two methods to enter an unknown number of grades:

- Method 1: ask for the total number of grades first and use a **For** loop to enter the scores.
- Method 2: ask for scores but use a 'flag' value such as -999 to indicate that there are no more grades. This method will use a **While** loop or a **Repeat** loop.

In both methods we will have to keep a running total of the grades. In Method 2 we also have to count the grades so that we can divide the total by that count.,

Your program should display the number of grades entered, the average of the grades and the appropriate warning message:

If the average is below 60: "Failing"

… 60 to 70: "Marginal"

… above 70: "Passing"

Teacher Tip: The section below discusses two related programming ideas: a counter and an accumulator variable. Emphasize that the variable on the left side of the store operator is the same variable as the one on the right but that they represent different values due to the processing involved.

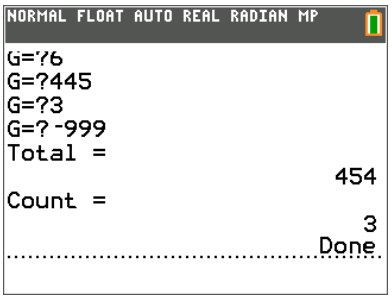
Counters and Accumulators

A statement such as **C+1→C** is called a *counter* because it adds 1 to the variable **C** each time it is executed.

A statement such as **T+N→T** is called an *accumulator* because it keeps a running total of the values of the variable **N**. The value of **N** is added to the variable **T** and then that sum is stored back into the variable **T**. At the end of a loop **T** will contain the total of the **N** values.

Here's an example that uses a **C**ounter and an **A**ccumulator and a 'flag' value to keep track of the **G'**s in a program:

	Notes
0→C	initialize variables;
0→G	G is for Grade
0→T	C is for Count
Prompt G	T is for Total
While G≠-999	get first **G**rade
C+1→C	as long as it is not -999
T+G→T	add 1 to the Count
Prompt G	add the Grade to the **T**otal
End	ask for another Grade
Disp "Total =",T	
Disp "Count =",C	



The **While** loop above continues counting and accumulating the **G'**s as long as -999 is not entered. When -999 is entered the loop stops and the results are displayed.

> **Teacher Tip:** Point out the two **Prompt** statements in the code above. The first one gets the first grade and the last one gets the rest of the grades. The last one is at the *bottom* of the loop to prepare for going back to the **While** statement to test the value of G again.

Extension

As part of your input routine check to make sure that the value entered is a legitimate grade (between 0 and 100) and take appropriate action when the entered value is not legitimate.

> **Teacher Tip:** The extension requires another loop around the input statements to make sure the value entered is legitimate. The sample code below simply stops the program when an invalid value is entered. Students can do better!

Sample Answer (indenting is for instructional purposes only):

```
prgmWARNINGZ
ClrHome
0→A
0→C
0→T
Input "ENTER A GRADE:",A

While A≠999
  If A<0 or A>100
  Then
    Disp "OUT OF RANGE!"
    Stop
  Else
    C+1→C
    T+A→T
  End
  Input "ANOTHER? (OR -999):",A
End
T/C→M

ClrHome
Output(1,1,"NUMBER OF GRADES:")
Output(2,1," TOTAL OF GRADES:")
Output(4,1,"AVERAGE:")
Output(1,19,C)
Output(2,19,T)
Output(4,10,M)
If M<60
Then
  Output(5,9,"FAILING")
Else
  If M<70
  Then
    Output(5,9,"MARGINAL")
  Else
    Output(5,9,"PASSING")
  End
End
Pause
ClrHome
```

When testing a program try extreme cases such as entering -999 as the first score and entering negative values anywhere. If the program generates an error message then the programmer has not taken all scenarios into account. The program listing above will fail when -999 is entered as the first value because it will attempt to divide 0 by 0 which is undefined. The calculation of the average, M, should be wrapped in a test to make sure at least one grade has been entered. Below is a possible fix:

```
If C>0
Then
 T/C→M
Else
 Disp “NO GRADES ENTERED!”
 Stop
End
```

The **Stop** statement is used in this program to terminate a program *immediately*. This can be added anywhere in a program so that execution is interrupted and the home screen ‘Done’ message appears.