Unit 4: Loops | **Application: Bank Notices**

This application, "Bank 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:

- Use 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 application 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 program 'knows' which **End** goes with which statement.

An Example of Nested Structures

- Nesting is the programming technique of placing one control structure inside another. The term is derived from the idea of placing one cardboard box inside another in order to save space.
- A programmer places loops within loops, such as **If**s inside loops and loops inside **If**s, to accomplish more complex tasks as the program requires.
- It's important to put one *complete* structure *completely* inside a block of another structure in order to avoid errors.
- The program listing at the right shows a **While** loop and an **If** structure inside another **While** loop.
- Notice the multiple uses of the **EndWhile** statement; the program 'knows' which **EndWhile** belongs with which **While**.
- The indenting is for visual confirmation and helps to clarify the logic.

```
Define perfsquare()=
Prgm
a:=1
While a>0
  Request "Enter a number (0 to end)",a
  While a<0
    Text "Must be non-negative!"
    Request "Enter a positive number",a
  EndWhile
  s:=√(a·1.)
  If s=int(s) Then
    Disp "It is a perfect square"
  Else
    Disp "It is not a perfect square"
  EndIf
  Disp "Its square root is",s
EndWhile
EndPrgm
```

The program first sets up a loop to continue until the entered value is zero. Then it tests to see if **a<0**. If it is, the program displays "Must be non-negative!" and asks for another value for **a**. But when **a** is not negative then the square root calculation and an **If** statement is processed and then the **Disp** statement is executed. When 0 is entered the program ends.

Summary of the Three TI-Nspire™ CX Loops:

For(*var*, *start*, *end***)**	**While** <*condition*>	**Loop**
		If <*condition*> **: Exit**
EndFor	**EndWhile**	**EndLoop**

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.

Loop is used when you are certain that you want the loop body to run at least once and it must contain an **Exit** statement, usually as part of an **If** statement.

Teacher Tip: The only 'necessary' loop is the **While…EndWhile** loop. It can perform the same tasks as the other two loops. Do not use a **GoTo** statement to exit a Loop structure or anywhere else in any program. That is bad programming.
There are other control structures in the Control menu that are not covered in this tutorial series. See the TI-Nspire Reference Guide for more information.

In the square root step, the variable *a* is multiplied by 1. to ensure that the output is the same on both a TI-Nspire and TI-Nspire CAS platform. Using a decimal point in an expression forces the result to be approximate.

Unit 4 Application: Program "Bank Notices"

A bank customer can have several bank accounts at a certain bank. The bank requires a minimum average balance of $1000 in all of the accounts to avoid paying service charges.

If the average is between $1000 and $1250, then the bank sends the customer a notice with a warning of the potential of a service charge.

When the average moves above $1250, the bank sends the customer a thank you message for maintaining a good average balance.

Let's write a program that gives the user information about their accounts. The user will enter account balances. The program will determine the average balance amounts, and display the number of accounts, the average of the balances, and an appropriate message to the user. The messages can be: "SERVICE FEE CHARGED," "IN DANGER OF A SERVICE FEE," and "THANK YOU FOR YOUR BUSINESS!"

We can use two methods for entering an unknown number of values:

- Method 1: Ask for the total number of accounts first and use a **For…EndFor** loop to enter the values.
- Method 2: Ask for amounts, but use a 'flag' value such as -999 to indicate that there are no more accounts. This method will use a **While…EndWhile** loop.

In both methods, we will have to keep a running *total* of the values entered.

In Method 2, we also have to *count* the number of account balances so that we can divide the *total* by that *count*.

Your program should display 1) the number of account balances entered, 2) the average of the account balances, and 3) the appropriate message based on the balance average.

If the average is below 1000: "SERVICE FEE CHARGED"

If the average is 1000 to 1250: "IN DANGER…"

If the average is above 1250: "THANK YOU…"

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 assignment 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

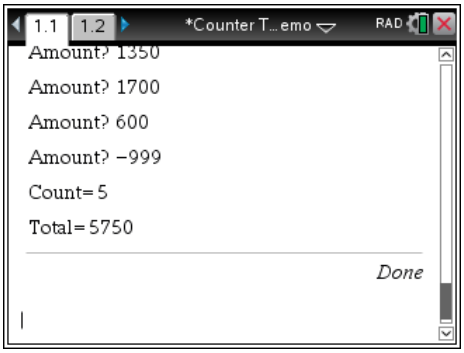
A statement such as **c:=c+1** is called a 'counter' because it adds 1 to the variable **c** each time it is executed.

A statement such as **t:=t+n** 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 encountered.

Here's an example that uses a counter, an accumulator (total), and a 'flag' value (-999) to keep track of the number of amounts entered in a program:

```
Prgm                                  Notes
Local counter,amount,total            initialize variables
total:=0
counter:=0
Request "Amount?",amount              get first amount
While amount≠−999                     as long as it is not -999
    counter:=counter+1                add 1 to the counter
    total:=total+amount               add the amount to the total
    Request "Amount?",amount          ask for another amount
EndWhile
Disp "Count=",counter
Disp "Total=",total
endPrgm
```

Sample Output



The **While…EndWhile** loop above continues counting and accumulating the amounts 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 **Request** statements in the code above. The first one gets the first amount and the last one gets the rest of the amounts. The last one is at the bottom of the loop to prepare to test the value of *amount* in the **While** statement again.

Extension

As part of your input routine, check to make sure that the value entered is a legitimate amount (greater than 0), and take appropriate action when the entered value is not legitimate.

> **Teacher Tip:** The extension requires another loop around each of 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. Encourage students to be eative.

Here is one possible solution:

```
Prgm
local amount, counter, total, avg
0→ amount
0→counter
0→total
Request "ENTER A balance amount: ", amount
While amount ≠ -999
  If amount <0 Then
    Disp "OUT OF RANGE!"
    Stop
  Else
    counter+1→counter
    total+ amount →total
  EndIf
  Request "ANOTHER amount (OR -999): ", amount
EndWhile
total/counter→avg
Disp "NUMBER OF accounts:", counter
Disp "TOTAL OF balances:", total
Disp "AVERAGE balance:",avg
If avg<1000 Then
  Disp "SERVICE FEE CHARGED"
ElseIf avg<1250 Then
  Disp "IN DANGER OF A SERVICE FEE"
Else
  Disp "THANK YOU FOR YOUR BUSINESS!"
EndIf
EndPrgm
```

Teacher Tip: Try extreme cases such as entering -999 as the first score. The program 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 should be wrapped in a test to make sure at least one amount has been entered. A possible fix is this:

```
If counter>0
Then
  Avg:=total/counter
ElseIf
  Disp “NO AMOUNTS 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 ‘Done’ message appears.

Encourage students to try changing the **Disp** statements to **DispAt** statements.)