Visual Logic can be installed from the CD that accompanies our textbook. It is a nifty tool for creating program flowcharts, but that is only half of the story. Until this software came along, flowcharts were static diagrams that outlined the flow of execution of a computer program. When a Visual Logic flowchart is done, you can run the program “behind” the flowchart. This is a great learning resource for budding programmers.

You can even create interactive programs that allow the user to input values for variables at runtime. Visual Logic is a terrific tool for grasping basic programming concepts such as variables, operators, expressions, branching, and looping. It is excellent preparation for more sophisticated programming with Visual Basic.NET. We hope you like it!

Install Visual Logic on your home PC by accepting all defaults in the installation wizard. Then, put the CD away in a safe place.

Once installed, Visual Logic can be started in the usual way from the Windows Start menu. On the “splash screen” shown here, click on “Continue” to produce the starting flowchart. The starting flowchart displays a “Begin” and “End” symbol as shown. To create a flowchart, you add symbols between these starting symbols.

Lesson One : Sample Visual Logic Program
(perform the actions that are in boldface)

Hover your mouse over the red arrow between the Begin and End symbols. It will turn green, indicating that you can click on it to get a popup window. This popup window has the other symbols we will need to create programs.

Click on the green arrow and select the Output symbol. This inserts the Output symbol into the flowchart as shown. The Output symbol produces output on the display screen. Next, we will specify what the program should output.

Double click on the Output symbol to get the Output dialog box.
The button in the lower left corner of the Output dialog toggles between More >> and <<Less. Clicking on More >> reveals that there are 3 Output types. For now, leave the type as “Dialog Box” as shown.

Enter your own name and be sure to enclose it within quotes. Later, we will use variables that don’t use quotes. Click on the “OK” button when done, and the Output symbol will change to show your intended output.

Run the program by clicking on the button on the toolbar, or by pressing the F5 function key.

Your name should appear in a popup dialog box as shown above on the right. If you got red error windows like the one on the left, you likely forgot one, or both, quotes. We all do this now and then but, no problem, it’s easy to debug. Just close the error windows, double click on the Output symbol, and add the quotes around your name. Run it again and it should work fine.

Click on the green arrow below the first Output symbol to add another Output symbol. Make this second Output generate the output “Visual Logic is fun”.

Run the program again. Did you remember the quotes? You should get two popup dialogs, one after the other.

Edit the first Output by clicking the More >> button. Select “console”. An odd looking “line feed” symbol, \n, will be appended to your name. This is normal for console output.
Click on OK.
Edit the second Output box in the same manner to select console output.

Note: the small white rectangle that now appears in each Output symbol indicates console output.

Run the program again.

It may seem that nothing happened, but if you look on the taskbar at the bottom of your screen, you will see that a new window has been created.

Click to open this console window. You should see a window as below (although much larger). This is console output, output to one window.

But the output needs more work. We need a space, or better yet, a line feed after the name.

Double click the first Output symbol. Position the cursor just to the left of the $ symbol and press Enter. The $ symbol will be moved down to the second line as shown below. This will cause a line feed in the output.

Run the program once more. You should now get two lines of output in your console window.

Save this program as example1 on your PC. Visual Logic will add a .vls filename extension to save the file as example1.vls on your system.

Summary

- Text output must be enclosed within quotes
- The More>> and <<Less toggle buttons will reveal or hide symbol options
- Console output will be to one window and is indicated by a small rectangle
- The $ symbol causes a line feed in a console window
Visual Logic Assignment One
Create a Visual Logic program named VLogic1.vls that uses four Outputs. The first three Outputs should create a console window similar to the one shown here. Specify your name, place of birth, and hobby or pastime. The fourth Output should create the popup dialog box exactly as shown.

This is the console window.

This is the popup dialog window.

Save the program as VLogic1.vls

Lesson Two (perform actions in boldface)

Start a new Visual Logic program with File | New on the menu.
Insert an Input symbol. The Input symbol is a parallelogram like the Output symbol, but unlike Output, it “leans” to the right.

Double click on the Input symbol and click More>> to expand the window.

Inputs are special because they enable a user to enter data when the program runs. Inputs require you to specify a variable name and a prompt.

The prompt will inform the user of the expected input, and the variable will accept and hold what the user enters. Complete the Input as shown.

Insert another Input below the first Input. Configure this Input to ask a user for his/her age as shown. Pay attention to the case of your variable names. These names follow the variable naming convention used in Visual Basic.
NOTE: If you forget to specify a prompt in an Input, Visual Logic will supply a generic prompt that is not very user friendly. Avoid this by expanding the Input with More >> and specifying a helpful prompt.

Insert an Assignment symbol (a rectangle ) below the second Input.

Assignment symbols interpret an expression on the right side of an = sign and assign the result to a variable named on the left side of the = sign.

You can start receiving Social Security payments at age 62. The Assignment shown above computes the number of years before a user could start receiving payments and stores it in a variable named SocialStart.

Insert an Output below the Assignment. Configure the Output as shown here.

Explanation of this Output

This Output introduces the & (ampersand) symbol in Outputs. The & is used to link strings of characters enclosed in quotes with variables that do not use quotes. Computer programmers called this linking concatenation. In this case, we have the variable YourName concatenated with the short string " is ", the variable Age, and the string " years of age". The spaces inside the strings are necessary to avoid jamming the variables up against characters in the strings.

Insert another Output above the End symbol. Configure it as shown below.

This Output concatenates the SocialStart variable between the two strings. Again, note the spaces after "in" and before "years" are essential for good quality output.

Run the program by clicking on the button on the toolbar, or by pressing the F5 function key.

NOTE: When prompted to input your name, enclose it in quotes. Always use quotes around text Inputs. Do not enclose the number entered for your age. Never use quotes with numeric input.
If you got a red error screen, check your work for quotes. It’s easy to forget one.

Edit all Inputs and Outputs by clicking on More >> and selecting the console option. The small white rectangle will indicate the console. In the first Output, press Enter with the cursor in front of the line feed symbol §. The program should appear as below on the left.

Run the program again to see the console output. It should appear as below:

Save the program as example2.vls on your system.

Whether you choose console output or dialog output is often a matter of personal preference.

Console output is better when you need to review your previous Inputs later in a program.
Summary

- An Input symbol is used to enter data into a program during a program run.
- Inputs store the entered data in variables, and are best with helpful prompts.
- Text input must be within quotes but numeric input is never inside quotes.
- An Assignment symbol interprets an expression and stores the result in a variable.
- The Assignment symbol in a flowchart is a rectangle.
- Always use meaningful variable names. For example, use Age for age...not x.
- The & character is used to concatenate variables and strings in Outputs.
- Spaces at the beginning or end of strings avoids “jams” when concatenating.

Visual Logic Assignment Two

Create a Visual Logic program that prompts the user to enter two numbers and then displays the sum, product, and average of the numbers. The entire program should run in one console window. Here are two sample runs:

Save the program as VLogic2.vls.

Hints

- Use the * operator for multiplication as shown.
- You will need sharp concatenation skills to get the outputs. Always place an & between a variable and a quoted string.
- You will need 2 Inputs, 3 Assignments, and at least 1 Output. It can be done with one Output (by pressing Enter for line feeds) but if you use 3 Outputs that’s OK.

Lesson Three  ( perform actions in boldface )
This assignment simulates a checkout in a retail store and covers these Visual Logic skills:

- Inputting non-numeric values for variables
- The built-in CurrencyFormat function
- A complete list of arithmetic operators

Start a new Visual Logic program. Insert an Input symbol, and use it to prompt the user to enter his/her name as shown. Remember to enclose the prompt in quotes, and select console. Add a few spaces before the closing quote, too.

Add two more Input symbols, one for the unit price of a merchandise item, and the other for the quantity purchased. Use console, appropriate variable names, and suitable prompts for both Inputs.

Now use an Assignment symbol to calculate the extended price (unit price x quantity) and store it in another variable.

At this point, the sample program should look something much like the graphic on the right.

Finally, use an Output symbol (in console) to summarize all of the Inputs.

The final output might be as shown here.

The entire program consists of just five symbols, namely three Inputs, one Assignment, and one Output (in that order).

**NOTE:** When entering the name, use quotes. Always use quotes for non-numeric inputs.

**NOTE:** This Output generated the final three lines of output.
Visual Logic has a number of built-in functions. The FormatCurrency function can produce better output for the UnitPrice and ExtPrice in our sample program. To use FormatCurrency, identify the variable to be formatted inside the parentheses ( ) as shown next.

```
Adding FormatCurrency function (in two locations)
```

```
New output with currency format where suitable
Save this example as retail.vls

Arithmetic Operators
The following table summarizes the Visual Logic arithmetic operators, and their usage in arithmetic expressions. You will use most of these in future assignments.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>Sum = 5 + 3</td>
<td>Sum = 8</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>Diff = 5 - 3</td>
<td>Diff = 2</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>Product = 5 * 3</td>
<td>Product = 15</td>
</tr>
<tr>
<td>/</td>
<td>real division</td>
<td>Quotient = 5 / 3</td>
<td>Quotient = 1.6666666</td>
</tr>
<tr>
<td>\</td>
<td>integer division</td>
<td>Answer = 5 \ 3</td>
<td>Answer = 1 ( no decimals )</td>
</tr>
<tr>
<td>^</td>
<td>exponentiation</td>
<td>Power = 5 ^ 3</td>
<td>Power = 125 ( 5<em>5</em>5 )</td>
</tr>
<tr>
<td>Mod</td>
<td>modulus</td>
<td>Remainder = 5 Mod 3</td>
<td>Remainder = 2</td>
</tr>
</tbody>
</table>

NOTE: modulus is another term for the remainder after division. Some more examples:
22 Mod 5 evaluates to 2  ( 5 divides into 22 four times with 2 remainder )
36 Mod 7 evaluates to 1  ( 7 divides into 36 five times with 1 remainder )
Summary
- Always enclose non-numeric input within quotes
- Press enter in an Output symbol to get a line feed (for a new line)
- Visual Logic has seven arithmetic operators
- The FormatCurrency function can be used to display dollars and cents

Visual Logic Assignment Three
Create a Visual Logic program that calculates a user's weekly paycheck. The program should prompt the user for his/her name, rate of pay, and hours worked. Then, the program should summarize the input. Strive to duplicate this output (all in the console). Note the currency formats in the output. Save as payroll.vls.

Lesson Four
Virtually all computer programs of any significance can make decisions and execute different branches of code based on the decision outcome. Often this means that one block of code will execute if a specified condition is true, but another block of code will execute if that condition was false.

In most programming languages, branching is achieved with IF. Visual Logic is no exception. To illustrate branching, we will create a Visual Logic program that determines if a user is old enough to vote in the USA.

Start a new Visual Logic program and set up an Input symbol as shown here. The Dialog Box option is selected.

Insert an IF condition symbol immediately after the Input. Note that the IF symbol is a diamond. Note too that the IF creates two branches, True and False. The False branch is also known as the ELSE branch.

Double click on the IF diamond to get the dialog box. This allows us to enter the condition for the IF symbol. In our case, the condition must test the user's age.
Enter Age >= 18 as the condition. Then click OK. The symbol >= means “greater than or equal to”, and as you know the voting age is 18 in the USA.

**NOTE:** All IF conditions are expressions that evaluate to either True or False.

**NOTE:** All IF conditions use one or more **conditional operators**. See a list on the next page.

Next, we add symbols to each branch of the IF.

**Insert an Output in the True branch. Configure it to output “You are of voting age”.
Insert an Output in the False branch that will output “You are too young to vote”.

Your completed program should look like this:

![Flowchart Diagram]

Run the program a few times. Try ages older and younger than 18. Try 18, too.
Save the program as agevote.vls.

The program should respond properly with respect to the Age variable. This simple program used only one symbol in both branches of the IF, but multiple operations in each branch are common.

**Operators Used for IF conditions**

<table>
<thead>
<tr>
<th>Conditional Operator</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>
Nested IFs
The branches of an IF diamond can themselves contain IFs that have their own branches. This is called nesting. For example, a program that qualifies a voter in the USA should ensure that the user is a US citizen, in addition to the age test. Let's try this.

Delete the Output in the True branch of voteage.vls and in its place add the Input symbol for Citizen shown here.
Insert an IF symbol under the Citizen Input in the True branch. Specify the condition as shown here.

Add Outputs to the branches of this second IF as shown

Save it again. Run the program with different ages and try both possible responses to the Citizen Input. Make sure that you use quotes around your response to the Citizen Input. The output should be what would be expected. If not, contact your instructor. The entire program should now appear as shown next.
If your program gets too expansive for the display, reduce the % on the toolbar.
**Compound Conditions**

By using logical operators, programmers can create compound conditions for IF symbols.

### Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>Less than</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>Greater than</td>
</tr>
<tr>
<td><strong>NOT</strong></td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

As an example of a compound condition, we will modify the previous program to make it detect a teenager.

Load the voteage.vls program if necessary. Save it as teenager.vls (to make a copy). Delete everything under the Age Input. Add an IF diamond and edit the condition as shown here. Click OK.

![IF diamond with compound condition](image)

This compound condition will be True for a teenager, since teenagers are aged between 13 and 19, inclusive. Note that there are two distinct conditions separated by the logical operator AND, and note that both conditions must be True for a teenager.

**NOTE**: Each condition in a compound condition must be enclosed within parentheses.

Edit the Output symbols as shown below:

- **Input**: Age
- **False**: (Age >= 13) AND (Age <= 19)
  - **Output**: "You are not a teenager"
- **True**: (Age >= 13) AND (Age <= 19)
  - **Output**: "You are a teenager"
- **End**

Run the program at least three times. Try ages for teenagers and ages that are too young or too old for a teenager. Note that the False branch executes for input that is too young or too old. Save the program again as teenager.vls.
Visual Logic Assignment Four

This program enhances payroll.vls from Assignment Three. You can save effort by loading it and saving another console version as overtime.vls. In the new version:

- Delete the Input for the user's name (to simplify things)
- Add Input that asks if the employer pays overtime at "time and a half" for hours over 40 in one week (42 hours = 2 hours of OT, which earns 3 hours of pay)
- Add an IF branch to determine if OT pay applies based on this Input
- Add an IF branch to determine the OT pay (if any is deserved)
- Add Output(s) showing the total pay (regular pay and OT, if any and applicable)

Here are some possible runs for the program:

In the above run, note that OT is not an issue since the weekly hours are not over 40.

In the above run, since hours exceed 40, the prompt about the employer's OT policy appears but the employer does not pay "Time and a half", so no OT is paid.

In the above run, since hours exceed 40, the prompt about the employer's OT policy appears but the employer does not pay "Time and a half", so no OT is paid.
In this final run, hours exceed 40 and the employer pays “time and a half”. The 4 hours of OT earn $60.00 of OT pay.

**NOTE: Your program should run in a like manner for these inputs.**

Save as `overtime.vls`.

---

**Lesson Five  (perform actions in boldface)**

This lesson is about loops. Often, computer programs need to repeat a block of code. The most efficient way to do this is by placing the code inside a loop. Like most programming languages, Visual Logic provides several methods for creating loops, namely For Loops and While Loops. In this lesson you will learn about **For Loops**.

**Start a new Visual Logic program.**

For loops use a “loop control variable” to either continue or halt the loop. As well as a name for the loop control variable (in the example below this is Count), there are three settings the programmer must specify in a for loop dialog box (as shown):

1. **an Initial Value**
   - this is the starting value for the Count variable
2. **a Final Value**
   - this is the stop value for the Count variable
3. **a Step**
   - this is the amount by which the Count variable will change in each loop cycle

**Insert a For Loop symbol and configure it as shown here. Click OK when done.**

The dialog box shown here is setting up a loop that will cycle 10 times. Count will start at 1 and increase by 1 in each loop cycle, stopping when Count reaches a value of 10. This means that Count will have the values 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10. We can prove this.

**Inside the loop, insert an Output symbol and configure it as shown. Run the program.**
NOTE: The “line feed” symbol, $\$\$, causes the Count values to be displayed one per line

Edit the For Loop symbol. Enter new values as shown here. Run the program again. Save as forloop1.vls.

You should get output from 10 – 100 in increments of 10. A negative value for Step can be useful when the Initial Value is greater than the Final Value.

CHALLENGE: Can you generate output that counts backwards from 10 to zero in decrements of 2?

Here’s another example that outputs a “times table” like the ones that can be found on the back cover of a grade school notebook.

Start a new Visual Logic program. Insert an Input and configure it as shown.

Insert a For Loop symbol and configure it to cycle from 1 to 12 as shown above. Inside the loop, insert an Output using concatenation to display the “times table”.

Run the program. Try some different inputs for Table. Your output should be something like the output shown above. Save as timestab.vls.

For Loops are best suited for programs in which the number of loop cycles, or iterations, are predictable or known at the outset.
Visual Logic Assignment Five
Create a Visual Logic program named namesay.vls. The program should prompt the user to enter his/her name and the number of times it should be displayed. Then, it should use the variable storing the number of times the name is displayed to generate output as shown in the two sample runs below. A For Loop is required.

```
What is your name? "Sarah"
How many times should your name be displayed? 3
Your name is Sarah
Your name is Sarah
Your name is Sarah
```

```
What is your name? "Tony"
How many times should your name be displayed? 4
Your name is Tony
Your name is Tony
Your name is Tony
Your name is Tony
```

Lesson Six (perform actions in boldface)
This lesson is about While Loops. Like For Loops, While Loops also can also execute instructions repeatedly. A While Loop will cycle if a specified condition is True. The condition is typically an expression using conditional operators such as < or >.

Start a new Visual Logic program.
Insert an Assignment symbol and configure it as shown.

Summary
- Loops use a “loop control variable” to run one or more enclosed instructions repeatedly.
- Each run through the loop is called a cycle or iteration.
- The number of cycles depends on the Initial Value, Step, and Final Value of the loop control variable.
- Any or all of the three For Loop settings for the loop control variable can be numeric variables.
The Assignment declares a value named Num and initializes it to a value of zero. Num will be the loop control variable.

Insert a While Loop symbol and configure it as shown here. The While Loop is being configured to execute repeatedly when Num is less than 10. The “Pre-Test” option specifies that the condition should be checked before the loop starts, and at the start of each iteration.

Insert an Output symbol within the While Loop and configure it as shown. Be sure to select Console. This symbol as configured should output the value of Num and generate a line feed. At this point, your program should look like the graphic below on the right.

It isn’t correct yet, but run the program.

You will get a console window that looks like the one on the left. Something is wrong. The program is locked in an endless loop. It is still running and it will continue to run until you stop it by clicking on the “Terminate” button on the toolbar. We have made a programming logic error to make a point. A While Loop must contain a “modifying statement” that will modify the loop control variable, or an endless loop results.

Stop the program by clicking on the “Terminate” button on the toolbar. Insert this Assignment symbol and configure it as shown. This Assignment will modify Num by incrementing it, so that the loop will end gracefully.

Run the program again. You should get the values from 0 to 9 in a console window as shown here.
When Num got to 10, the While Loop condition became False, and the program ended. **Save this example as while1.vls.**

Now let’s see what that “Pre-Test” option was all about.

**Edit the first Assignment. Give Num an initial value of 11. Run the program again.** Nothing happens because, with Num being 11, the condition of the While Loop evaluates to false before the loop starts (11 is not less than 10). The program follows the False route, and ends without ever running the While Loop at all.

**Edit the While Loop symbol. Select “Post-Test” instead of “Pre-Test”.** The Visual Logic program reacts on your screen to reflect this change. The program now looks like the graphic here. The “Post-Test” selection means that the condition will be tested at the end of the loop. By force, the While Loop will run at least once when “Post-Test” is selected. In this case it does run only once, because the condition evaluates to False (11 is not less than 10). The program should output 11 and nothing else.

**Exit Loop and Intentional Endless Loops**
Visual Logic has an Exit Loop symbol that will immediately end a loop, and allow the program to continue with any instructions that might follow the loop. Usually the Exit Loop symbol is used inside an IF, to stop a loop when a specified IF condition is True.

Exit Loop is also useful inside intentional endless loops. An endless loop isn’t always a programming error. A Visual Logic program might use an intentional endless loop to repeatedly run one or more instructions until a certain IF condition is met. When the IF condition is True, Exit Loop is used to end the endless loop. This is especially useful for validating user input to a program. An endless loop can run over and over again until the user enters the input as desired. **While Loops can be made intentionally endless by entering True as the while condition.** Here is an intentional endless While Loop that validates the entry of a test score. The loop won’t allow a TestScore higher than 100.
Visual Logic Assignment Six

Write a program called VLogic6.vls that asks the user to enter an odd integer and runs repeatedly until he/she complies. Thank the user when the input is odd and let the user know if the input is incorrect. Use an endless While Loop and the Exit Loop symbol. The entire program should run in a console window.

HINT: You will need the modulus operator (Mod) to test for an odd number in the IF. This task may stump you at first, but if you really think about how you identify odd numbers, you will succeed. Here is a sample run:

Lesson Seven (perform actions in boldface)

This lesson is about accumulators. Many computer programs that feature loops make use of accumulators. An accumulator is a variable that increases by the value of another variable. An accumulator is like a bucket into which a program can toss numbers. Both counters and accumulators should be initialized with a value of zero. When you approach the checkout at the supermarket, your initial total bill is an accumulator with a value of zero. As each item you are purchasing is rung up, the bill “accumulates” by the price of each item. As another example, we will create a program that calculates the average of three inputted numbers.

Start a new Visual Logic program and begin it with an Assignment symbol for an accumulator named Total. Initialize Total to zero.

Add a For Loop that will iterate 3 times so the user can input 3 numbers. Inside the For Loop, insert an Input symbol for a variable named Number. Use the Console option and a suitable prompt.

At this point your program should appear as shown here.
Beneath the Input, add an Assignment symbol to accumulate Total by the value of Number as shown.

After the loop (just above End), insert an Assignment to calculate the Average of the numbers that were entered.

Finally, insert an Output to display the Average.

Here is the final program and a sample run:

```
Enter a number, please 12
Enter a number, please 13
Enter a number, please 14
The average of those numbers is 13.
```

**IMPORTANT:** note the syntax used for the accumulator Assignment symbol. The name of the accumulator variable is used on both sides of the equals sign.

---

**Summary**

- An accumulator is like a “bucket” that increases by the value of another variable
- Accumulators should be initialized to a starting value of 0
- The syntax for adding on to an accumulator has this form:
  
  \[
  \text{Accumulator} = \text{Accumulator} + \text{SomeVariable}
  \]

**Visual Logic Assignment Seven**

Write a program called `VLogic7.vls` that simulates checking out at the supermarket, but extend the averaging example above by asking the user to input how many items are being purchased. **NOTE:** You will need to use the variable for this input as one of the settings in your `For Loop`. Use an accumulator to total the bill. The program should run in a console. Try to duplicate the output in the sample run shown. **Note the currency format.**
Lesson Eight  ( perform actions in boldface )

This final lesson is about counters and sentinel values. A counter is a variable that increments by one, usually when a certain condition is True. Counters are often used within loops and inside IF symbols.

A sentinel "watches out for" a particular value in a variable. Typically, a sentinel is used in an IF symbol to detect a specific value input by the user. When the special sentinel value is detected, the program reacts in some desired way.

In the following example, a counter and a sentinel are used to write a different version of the averaging program outlined in Lesson 7. If you have trouble grasping the following step-by-step instructions, the entire program and two sample runs can be seen on the next page.

Start a new Visual Logic program with an Assignment symbol for an accumulator named Total. Initialize Total to zero (the same as Lesson 7).

Insert another Assignment that initializes a counter named Count to 0.

Insert a While Loop and make it endless by setting the while condition as True.

Inside the loop, insert an Input symbol for a variable named Number. In the prompt, ask the user to enter a number, but this time, notify the user that entering a value of 0 will end the process. This program will use a sentinel value of 0 for the Number variable to determine that the user has no more numbers to average.

Directly below the Input for Number, insert an IF symbol. Set the IF condition to be Number = 0 (the sentinel value). Insert an Exit Loop symbol in the True branch of the IF.

Insert two Assignment symbols beneath the IF. One should accumulate the Total variable by the value of Number, and the other should increment the Count variable by 1.

Just above the "End" symbol (in the False branch of the While Loop), insert an Assignment that determines the Average using the Total variable and the Count variable.

Finally, insert an Output symbol to display the average. Run the program a few times.

The entire program, and two sample runs, can be viewed on the following page.
This program is impressive since the user does not have to indicate in advance how many items are being purchased.

Note that the counter Count increases by just 1, but the accumulator Total increases by the value of another variable, Number.
Visual Logic Assignment Eight
Write a Visual Logic program named VLogic8.vls that implements another version of the supermarket checkout program from assignment 7. This time, use a counter and a sentinel so any number of items may be purchased and totaled up at the checkout. Here are two sample runs:

All DONE!
This completes our official study of Visual Logic. We hope you had fun. You might want to create some graphical programs that we didn’t have time for with the graphics symbols available in Visual Logic. You can create some very cool graphical displays, especially with loops.

B. Yourth
St. Petersburg College, FL