Chapter 3 Part 2

Numbers

We have already seen a little bit of working with numbers – for example, setting the size or position of a window. When we put a numeric value directly into the program, these are called numeric literals.

VB.NET allows us to perform standard arithmetic operations:

<table>
<thead>
<tr>
<th>Arithmetic Operator</th>
<th>VB.NET Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
</tr>
<tr>
<td>Division</td>
<td>/ (note / not )</td>
</tr>
<tr>
<td>Exponent</td>
<td>^</td>
</tr>
<tr>
<td>Modulus</td>
<td>mod</td>
</tr>
</tbody>
</table>

One way to show a number on the screen is to display it in a list box. If there is a control named **ListBox** added to your project, and \( n \) is a number, then the instruction:

```vbnet
ListBox.Items.Add(n)
```

will display the number \( n \) as the last item in the list box. Add is called a **method**, sometimes referred to as a subprogram, procedure, or function. The code behind this method has been written by the developers of VB.NET for you to use.

To erase all items in the list box, use the method **Clear**:

```vbnet
ListBox.Items.Clear()
```

Here are some examples of arithmetic operations and storing the result in a list box:

```vbnet
ListBox.Items.Add(3 + 2)
ListBox.Items.Add(3 - 2)
ListBox.Items.Add(5 * 2 * 10)
ListBox.Items.Add(14 mod 5)
ListBox.Items.Add(10 / 2)
ListBox.Items.Add(11 / 2)
ListBox.Items.Add(1 / 2)
ListBox.Items.Add(2 ^ 3)
ListBox.Items.Add((2^3)*3.1)
```
The results are:

5
1
100
4
5
5.5
0.5
8
24.8

Note that VB.NET will convert numbers with a fractional result to a new format that stores the fractional part. Extremely large numbers will be displayed in scientific notation, where the letter E refers to an exponent of $10^E$:

lstBox.Items.Add($2^{50}$)

outputs: $1.1259E+15$

Another often more convenient way to output variables is to print them to the Console screen. Use either of the following methods to output to the console window:

    Console.WriteLine(n)  - outputs n with a carriage return and newline
    Console.Write(n)  - outputs n with no carriage return and newline

The results will show up in the “Debug” window of VB.NET if you are running in Debug mode, or in a DOS window if the program was invoked from DOS directly.

**Variables**

In math problems quantities are referred to by names. For example, in physics, there is the well known equation:

    Force = Mass $\times$ Acceleration

By substituting two known quantities we can solve for the third. When we refer to quantities or values with a name, these are called **variables**. Variables must begin with a letter and may contain numbers or underscores but not other characters.

To use variables we must tell VB.NET what **data type** our variables should be. We do this using the **Dim** statement, which “Dimensions” a storage location for us using the format:

    Dim varName as DataType
The Dim statement causes the computer to set aside a location in memory with the name varName.  DataType can take on many different types, such as:

- **Integer** - Format for integral data
  Range: -2,147,483,648 through 2,147,483,647
- **Double** - Format for floating point data
  Range: \(-1.8 \times 10^{308}\) to \(1.8 \times 10^{308}\)
- **String** - Textual data

If varName is a numeric variable, the Dim statement also places the number zero in that memory location. (We say that zero is the initial value or default value of the variable.) Strings are set to blank text.

If double has a larger data range than integer, and can store floating point numbers, you might wonder why we don’t just declare everything as a double. We could do this, but it would be wasteful – the double format takes up more space than an integer. Also it is slower to perform arithmetic operations on a number stored as double than it is on a number stored as integer. The integer data type is better to use if that is all your application needs.

To assign or copy a value into a variable, use the = or assignment operator:

```plaintext
myVar = newValue
```

We can also assign an initial value when we declare a variable:

```plaintext
Dim myVar as Integer = 10
```

Here are some examples using numeric variables:

```plaintext
Dim dblVal as Double
Dim intVal as Integer

dblVal = 5 * 2 * 10
intVal = 5 * 2 * 10
Console.WriteLine(dblVal)
Console.WriteLine(intVal)
dblVal = 11 / 2
intVal = 11 / 2
Console.WriteLine(dblVal)
Console.WriteLine(intVal)
dblVal = 1 / 2
intVal = 1 / 2
Console.WriteLine(dblVal)
Console.WriteLine(intVal)
```
VB.NET will round floating point values up or down when converted to an integer (although 0.5 seems to be an exception).

A common operation is to increment the value of a variable. One way to do this is via:

```vbnet
intVal = intVal + 1
```

This is common enough that there are shortcuts:

```vbnet
x = x + y    <->    x += y
x = x * y    <->    x *= y
x = x - y    <->    x -= y
x = x / y    <->    x /= y
```

**Precedence Rules**

The precedence rules of arithmetic apply to arithmetic expressions in a program. That is, the order of execution of an expression that contains more than one operation is determined by the precedence rules of arithmetic. These rules state that:

1. parentheses have the highest precedence
2. multiplication, division, and modulus have the next highest precedence
3. addition and subtraction have the lowest precedence.

Because parentheses have the highest precedence, they can be used to change the order in which operations are executed. When operators have the same precedence, order is left to right.

Examples:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value stored in X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dim x As Integer</td>
<td></td>
</tr>
<tr>
<td>x = 1 + 2 + 6 / 6</td>
<td>4</td>
</tr>
<tr>
<td>x = (1 + 2 + 3) / 6</td>
<td>1</td>
</tr>
<tr>
<td>x = 2 * 3 + 4 * 5</td>
<td>26</td>
</tr>
<tr>
<td>x = 2 / 4 * 4</td>
<td>2</td>
</tr>
<tr>
<td>x = 2 / (4 * 4)</td>
<td>0</td>
</tr>
<tr>
<td>x = 10 Mod 2 + 1</td>
<td>1</td>
</tr>
</tbody>
</table>

In general it is a good idea to use parenthesis if there is any possibility of confusion. There are a number of built-in math functions that are useful with numbers. Here are just a few:
Math.Sqrt(number) returns the square root of number

Ex:

    Console.WriteLine(Math.Sqrt(9)) ' Displays 3

    Dim d as Double
    d = Math.Sqrt(25)
    Console.WriteLine(d) ' Displays 5
    Console.WriteLine(Math.Sqrt(-1)) ' Displays NaN

Math.Round(number) returns the number rounded up/down

Ex: Math.Round(2.7) returns 3

Math.Abs(number) returns the absolute value of number

Ex: Math.Abs(-4) returns 4

There are many more, for sin, cos, tan, atan, exp, log, etc.

When we have many variables of the same type it can sometimes be tedious to declare each one individually. VB.NET allows us to declare multiple variables of the same type at once, for example:

    Dim a, b as Double
    Dim a as Double, b as Integer
    Dim c as Double = 2, b as integer = 10

Variable Scope

When we DIM a variable inside an event, the variable only “exists” within the scope of the event. This means we are free to define other variables of the same name in different events (which is often quite useful to keep variables from stomping on each other’s values!) For example

    Private Sub MyClick1(...) Handles MyButton.Click
        Dim i As Integer
        i = 10 / 3
    End Sub
Private Sub MyClick2(...) Handles MyButton2.Click
    Dim i As Integer
    i = 30
End Sub

The variable i in the two subroutines is a different i; the first exists only within the scope of MyClick1 and the second only exists within the scope of MyClick2.

**Strings**

A string variable is a variable that refers to a sequence of textual characters. A string variable is declared by using the data type of String:

    Dim s as String

To assign a literal value to a string, the value must be in double quotes. The following shows how to add three strings to a listbox:

    Dim day1 As String
    Dim day2 As String
    day1 = "Monday"
    day2 = "Tuesday"
    ListBox.Items.Add(day1)
    ListBox.Items.Add(day2)
    ListBox.Items.Add("Wednesday")

This fills the listbox with “Monday”, “Tuesday”, and “Wednesday”.

If we are adding a lot of items to the listbox then it can be tedious to write “ListBox.Items” every single time. A shorter way is to use the With command:

    With ListBox.Items
        .Add(day1)
        .Add(day2)
        .Add("Wednesday")
    End With

The with command takes a collection (ListBox.Items) which denotes the beginning of a block of statements that all pertain to ListBox.Items. Since VB.NET knows the statements are in reference to ListBox.Items, we can leave it off and only use the dot followed by the method name.

Two strings can be combined to form a new string consisting of the strings joined together. The joining operation is called **concatenation** and is represented by an ampersand (&).
For example, the following outputs “hello world”:

    Dim s1 as String = “hello”
    Dim s2 as String = “world”
    Console.WriteLine(s1 & “ “ & s2)

This outputs: hello world

Note that if we output: Console.WriteLine(s1 & s2)

Then we would get: helloworld

Sometimes with strings we can end up with very long lines of code. The line will scroll off toward the right. You can keep on typing to make a long line, but an alternate method is to continue the line on the next line. To do that, use the line continuation character. A long line of code can be continued on another line by using underscore ( _ ) preceded by a space:

    msg = ”640K ought to be enough “ & _
         ”for anybody. (Bill Gates, 1981)”

is the same as:

    msg = “640K ought to be enough “ & “for anybody. (Bill Gates, 1981)”

### String Methods and Properties

There are a number of useful string methods and properties. Just like control objects, like list boxes, that have methods and properties, strings are also objects and thus have their own properties and methods. They are accessed just like the properties and methods: use the name of the string variable followed by a dot, then the method name.

    str.Length() ; returns number of characters in the string
    strToUpper() ; returns the string with all letters in uppercase
                 ; does not change the original string, returns a copy
    strToLower() ; returns the string with all letters in lowercase
                 ; does not change the original string, returns a copy
    str.Trim() ; returns the string with leading and trailing whitespace
               ; removed. Whitespace is blanks, tabs, cr’s, etc.
    str.Substring(m,n) ; returns the substring of str starting at character m
                        ; and fetching the next n characters. M starts at 0
                        ; for the first character! If n is left off, then the remainder
                        ; of the string is returned starting at position m.
Here are some examples:

Dim s As String = "eat big macs   
Console.WriteLine(s.Length())
Console.WriteLine(s.ToUpper())
Console.WriteLine(s & "!")
s = s.Trim()
Console.WriteLine(s & "!")
Console.WriteLine(s.Substring(0, 3))
Console.WriteLine(s.Substring(4))
Console.WriteLine(s.Substring(20))

Output:

15
EAT BIG MACS
eat big macs!
eat big macs!
eat
big macs
CRASH!  Error message (do you know why?)

Sometimes it is useful to convert individual characters back and forth from their ANSI representation to the number that represents that character. Recall that a string is really a sequence of codes, where each code represents a different character. A small subset of the ASCII code is shown below:

<table>
<thead>
<tr>
<th>32 (space)</th>
<th>48 0</th>
<th>66 B</th>
<th>122 z</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 !</td>
<td>49 1</td>
<td>90 Z</td>
<td>123 {</td>
</tr>
<tr>
<td>34 “</td>
<td>57 9</td>
<td>97 a</td>
<td>125 }</td>
</tr>
<tr>
<td>35 #</td>
<td>65 A</td>
<td>98 b</td>
<td>126 -</td>
</tr>
</tbody>
</table>

The string “Ab” is really represented by storing the values 65 and 98 consecutively. To access the ASCII value of a character use ASC:

Console.WriteLine(Asc("A"))  ' Outputs 65

To go in the reverse direction, from the code to the character, use Chr:

Console.WriteLine(Chr(65))  ' Outputs A
On occasion you may be interested in generating the **empty string**, or a string with nothing in it. This is a string of length 0. It is referenced by simply “” or two double quotes with nothing in between.

Finally, if you would like to create a string that contains the “ character itself, use two “”s:

Wrong:

```vbnet
s = “Dan Quayle said, “I love California; I practically grew up in Phoenix.””
```

What is the problem?

Right:

```vbnet
s = “Dan Quayle said, “”I love California; I practically grew up in Phoenix.”””
```

**Using Text Boxes for Input and Output**

It turns out that any text property of a control is also a string, so what we just learned about strings also applies to the controls! A particularly useful example is to manipulate the content of text boxes.

For example, say that we create a text box control named txtBox. Whatever the user enters into the textbox is accessible as a string via txtBox.Text. For example:

```vbnet
Dim s as String
s = txtBox.Text.ToUpper()
txtBox.Text = s
```

This changes the txtBox.Text value to be all upper case letters.

Text Boxes provide a nice way to provide textual input and output to your program. However, recall that other items also have a text property, such as Me.Text, which will change the caption on the title bar of your application. Similarly you can change items in a list box.

Because the contents of a text box is always a string, sometimes you must convert the input or output if you are working with numeric data. You have the following functions available for *type-casting*:

- **CDbl(string)** ; Returns the string converted to a double
- **CInt(string)** ; Returns the string converted to an integer
- **CStr(number)** ; Returns the number converted a string

For example, the following increments the value in a text box by 1:
Dim i as Integer
    i = CInt(txtBox.Text)
    i = i + 1
    txtBox.Text = CStr(i)

Options

It turns out that VB.NET actually allows you to perform these operations without the conversion functions:

    i = txtBox.Text  ' implicitly converts the string to a number

However, this practice is not recommended because it can often lead to errors when the programmer really didn’t intend to convert the variables. For this reason, VB.NET includes a way to require type-casting. At the top of the code, add the line:

    Option strict on

This forces type-casting or a program will not compile.

Another recommended option is option explicit:

    Option explicit on

This forces the programmer to define any variables that are going to be used, another potentially common error (consider a typo in a variable name; without option explicit on VB.NET will just create a new variable instead of using the one you may intend).

You can put both options next to each other at the top of your code and it is highly recommended you always use both:

    Option strict on
    Option explicit on

Comments

As your code gets more complex, it is a good idea to add comments. You can add a comment to your code by using the ‘ character. Anything from the ‘ character to the end of the line will be ignored. If you neglect to add comments, it is very common to forget how your code works when you go back and look at it later!

Another common use of comments is to “comment out” blocks of code. For example, if you insert code for testing purposes or if code is not working properly, you can comment it out and have the compiler ignore it. However, the code will still be there if you want to use it again later without having to type it in again – just uncomment the code.
VB.NET has a button to comment and uncomment blocks of code:

Highlight the text to comment and click the icon shown above on the left. Highlight the text to uncomment and click the icon shown above on the right.

In-class Exercise:

It is recommended that you maintain your training heart rate during an aerobic workout. Your training heart rate is computed as:

\[ 0.7(220-a)+(0.3*r) \]

where \( a \) is your age in years and \( r \) is your resting heart rate. Write a program to compute the training heart rate as shown below:

Example:

You are running a marathon (26.2 miles) and would like to know what your finishing time will be if you run a particular pace. Most runners calculate pace in terms of minutes per mile. So for example, let’s say you can run at 7 minutes and 30 seconds per
Write a program that calculates the finishing time and outputs the answer in hours, minutes, and seconds.

Input:
Distance : 26.2
PaceMinutes: 7
PaceSeconds: 30

Output:
3 hours, 16 minutes, 30 seconds

Here is one algorithm to solve this problem:

1. Express pace in terms of seconds per mile, call this SecsPerMile
2. Multiply SecsPerMile * 26.2 to get the total number of seconds to finish. Call this result TotalSeconds.
3. There are 60 seconds per minute and 60 minutes per hour, for a total of 60*60 = 3600 seconds per hour. If we divide TotalSeconds by 3600 and throw away the remainder, this is how many hours it takes to finish.
4. TotalSeconds mod 3600 gives us the number of seconds leftover after the hours have been accounted for. If we divide this value by 60, it gives us the number of minutes, i.e. (TotalSeconds mod 3600) / 60
5. TotalSeconds mod 3600 gives us the number of seconds leftover after the hours have been accounted for. If we mod this value by 60, it gives us the number of seconds leftover. (We could also divide by 60, but that doesn’t change the result), i.e. (TotalSeconds mod 3600) % 60
6. Output the values we calculated!

In-Class Exercise: Write the code to implement the algorithm given above.

In-Class Exercise: Write a program that takes as input an amount between 1 and 99 which is the number of cents we would like to give change. The program should output the minimum number of quarters, dimes, nickels, and pennies to give as change assuming you have an adequate number of each coin.

For example, for 48 cents the program should output;
1 quarter
2 dimes
0 nickels
3 pennies

First write pseudocode for the algorithm to solve the problem. Here is high-level pseudocode:

- Dispense max number of quarters and re-calculate new amount of change
- Dispense max number of dimes and re-calculate new amount of change
- Dispense max number of nickels and re-calculate new amount of change
- Dispense remaining number of pennies
**Input and Output**

We have already seen how to get input via textboxes and we can also output data via textboxes, listboxes, or the console window.

We will not cover this in class but to format numbers, currency, or percents, there is a format function (for example, FormatNumber(1.23456,1) turns the number into only a single decimal point, 1.2). We can also format to pad numbers with spaces.

Another way to input and output data is through “pop-up” windows. To input data through an input dialog box, use a statement of the form:

```
stringVar = InputBox(prompt, title)
```

To output data via a popup use a statement of the form:

```
MsgBox(string)
```

for example:

```
Dim s as String
s = InputBox(“Enter your name”, “Name”)
MsgBox(“You entered “ & s & “.”)
```

Generates a window like the following:

![Input dialog box](image1)

Whatever the user types into the text area is stored into variable s when the user presses OK. The output is then shown in a message box:

![Message box](image2)

If the user presses cancel, then the string returned is empty.
There are additional options on the InputBox and MsgBox to set the title, icon, and buttons that appear on the pop-up window. See the VB.NET reference for more information.

**Getting Help**

How do you get help from the VB.NET reference? There are many ways. One is to use the “Dynamic Help” pane. Another is to select “Help” from the menu and navigate through. A method that I like is to click on the item you want help on, and press F1. If you have MSDN installed, this will bring up a help screen that has information about the selected item, and often includes useful code samples.

**Reading Data from a Text File**

If you have stored data in a text file, using a program such as Notepad, you can also read it with VB.NET. Later we will see how to write programs that can save data to text files. Reading from a text file is useful to load the program with large amounts of data that would otherwise be tedious to type.

Data can be stored in files and accessed with a StreamReader object.

The steps to use the StreamReader object are as follows:

1. Execute a statement of the form

   Dim readerVar As IO.StreamReader

   A StreamReader is an object from the Input/Output class that can read a stream of characters coming from a disk or coming over the Internet. The Dim statement declares the variable readerVar to be of type StreamReader.

2. Execute a statement of the form

   readerVar = IO.File.OpenText(filespec)

   where filespec identifies the file to be read. This statement establishes a communications link between the computer and the disk drive for reading data from the disk. Data then can be input from the specified file and assigned to variables in the program. This assignment statement is said to “open the file for input.”

   Just as with other variables, the declaration and assignment statements in Steps 1 and 2 can be combined into the single statement:

   Dim readerVar As IO.StreamReader = IO.File.OpenText(filespec)

3. Read items of data in order, one at a time, from the file with the ReadLine method.
Each datum is retrieved as a string. A statement of the form

\[
\text{strVar} = \text{readerVar.ReadLine}
\]

ciauses the program to look in the file for the next unread line of data and assign it to the variable \text{strVar}. The data can be assigned to a numeric variable if it is first converted to a numeric type with a statement such as

\[
\text{numVar} = \text{CDbl(readerVar.ReadLine)}
\]

Note: If all the data in a file have been read by ReadLine statements and another item is requested by a ReadLine statement, the item retrieved will have the value Nothing.

4. After the desired items have been read from the file, terminate the communications link set in Step 2 with the statement

\[
\text{readerVar.Close()}
\]

As an example, an estimate of the cost it takes to raise a child to age 18 is stored in the file C:\COSTS.TXT and it contains the following:

- Housing
- 54000
- Food
- 33000
- Transportation
- 23000
- Clothing
- 13000
- Health
- 18000
- Education
- 14000
- Other
- 16000

This says that Housing costs $54,000 while Food costs $33,000, etc.

Here is a program that reads in all the data and outputs the total cost:
Dim costFile As IO.StreamReader = IO.File.OpenText("C:\COSTS.TXT")
Dim strAll, strTemp As String
Dim intTotal As Integer

strAll = costFile.ReadLine() ' Reads "Housing"
strTemp = costFile.ReadLine() ' Reads 54000
intTotal += CInt(strTemp) ' Conver to Integer
strAll = strAll & "," & costFile.ReadLine() ' Reads "Food"
intTotal += CInt(costFile.ReadLine()) ' Reads and converts 33000
strAll = strAll & "," & costFile.ReadLine() ' Reads "Transportation"
intTotal += CInt(costFile.ReadLine())
strAll = strAll & "," & costFile.ReadLine() ' Reads "Clothing"
intTotal += CInt(costFile.ReadLine())
strAll = strAll & "," & costFile.ReadLine() ' Reads "Health"
intTotal += CInt(costFile.ReadLine())
strAll = strAll & "," & costFile.ReadLine() ' Reads "Education"
intTotal += CInt(costFile.ReadLine())
strAll = strAll & "," & costFile.ReadLine() ' Reads "Other"
intTotal += CInt(costFile.ReadLine())
costFile.Close() ' Close the file

MsgBox("For:" & strAll & " The total cost is : " & intTotal)

The output is:

For:Housing,Food,Transportation,Clothing,Health,Education,Other The total cost is : 171000

Note that there is a lot of repeat code. We repeat the same two lines of code to read in the name and cost of an item. Later we will see how to make a loop that will repeatedly execute the same lines of code over and over again so we don’t need to tediously make duplicate copies of the code.