

Chapter 3 – 4.2, Data Types, Arithmetic, Strings, Input

Data Types

Visual Basic distinguishes between a number of fundamental data types. Of these, the ones we will use most commonly are:

- Integer
- Long
- Double
- String
- Boolean

The table below summarizes the different types:

Type	Bytes of Storage	Range of Values
Byte	1	0 to 255
Integer	4	–2,147,483,648 to 2,147,483,647
Long	8	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Short	2	–32,768 to 32,767
Single	4	–3.402823E38 to +3.402823E38
Double	8	–1.79769313486231E308 to +1.79769313486231E308
Char	2	Any Unicode character
String	2 per char	0 to approximately 2 billion Unicode characters
Boolean	1	True or False
Decimal	16	–79,228,162,514,264,337,593,543,950,335 to 79,228,162,514,264,337,593,543,950,335 with 29 Significant digits
Date	8	00:00:00 on January 1, 0001 to 11:59:59 on December 31, 9999
Object	4	(Reference to an object)

An Integer is a positive or negative number with no value past the decimal point. Note the limitation on the range of values it can hold. If we allocate more storage space (e.g., more bytes) then we can represent larger numbers.

The Long data type uses 8 bytes of storage instead of 4 like the Integer, so it can represent much larger values.

Similarly, VB has two commonly used floating point values: Single and Double. These data types are used to represent real numbers. The Single uses 4 bytes and the Double uses 8 bytes, so the Double can store larger values than the single.

If double has a larger data range than integer, and can store floating point numbers, you might wonder why we don't just always use 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.

Booleans are used to represent True or False. These are the only two values that are allowed. Booleans are useful in programming due to their ability to select a course of action based upon some outcome that is either true or false, so we will use Booleans extensively in decision-making.

Strings consist of textual data and are enclosed in double-quotes. Strings are represented as a sequence of bit patterns that match to alphanumeric values. For example, consider the following mapping for the letters A, B, and C:

A	01000001
B	01000010
C	01000011

To store the string “CAB” we would simply concatenate all of these codes:

01000011 01000001 01000010

Note that there is a difference between a string of numbers, and a number such as an Integer. Consider the string “0” and the number 0. The String “0” is represented by the bit pattern 00110000 while the integer 0 is represented by 00000000. Similarly, the string “10” would be represented as 00110001 00110000 while the integer 10 is represented as 00001010.

Strings are simply a sequence of encoded bit patterns, while integers use the binary number format to represent values. We will often convert back and forth between String and Number data types.

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:

<u>Arithmetic Operator</u>	<u>VB.NET Symbol</u>
Addition	+
Subtraction	-
Multiplication	*
Division	/ (floating point)
Division	\ (integer, truncation)
Exponent	^
Modulus	mod

Here are some examples of arithmetic operations and outputting the result to the console:

```
Console.WriteLine(3 + 2)
Console.WriteLine (3 - 2)
Console.WriteLine (5 * 2 * 10)
Console.WriteLine (14 mod 5)
Console.WriteLine (9 mod 4)
Console.WriteLine (10 / 2)
Console.WriteLine (11 / 2)
Console.WriteLine(11 \ 2)
Console.WriteLine (1 / 2)
Console.WriteLine (2 ^ 3)
Console.WriteLine ((2^3)*3.1)
```

The results are:

```
5
1
100
4
1
5
5.5
5
0.5
8
24.8
```

Extremely large numbers will be displayed in scientific notation, where the letter E refers to an exponent of 10^E :

```
Console.WriteLine(2^50)
```

outputs: 1.1259E+15

Variables

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

$$\text{Force} = \text{Mass} \times \text{Acceleration}$$

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, Single, Double, String, etc.

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.

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

```
myVar = newValue
```

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

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

Here are some examples using numeric variables:

```
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)
```

Output:

100
100
5.5
6
0.5
0

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:

```
intVal = intVal + 1
```

This is common enough that there are shortcuts:

$x = x + y$	\leftrightarrow	$x += y$
$x = x * y$	\leftrightarrow	$x *= y$
$x = x - y$	\leftrightarrow	$x -= y$
$x = x / y$	\leftrightarrow	$x /= y$

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:

Dim x As Integer	Value stored in X
$x = 1 + 2 + 6 / 6$	4
$x = (1 + 2 + 3) / 6$	1
$x = 2 * 3 + 4 * 5$	26
$x = 2 / 4 * 4$	2
$x = 2 / (4 * 4)$	0
$x = 10 \text{ Mod } 2 + 1$	1

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`.

More on 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"
Console.WriteLine(day1)
Console.WriteLine (day2)
Console.WriteLine ("Wednesday")
```

This outputs “Monday”, “Tuesday”, and “Wednesday”.

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
str.ToUpper()	; returns the string with all letters in uppercase does not change the original string, returns a copy
str.ToLower()	; 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?)
```

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:

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

What is the problem?

Right:

```
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:

```
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.

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.

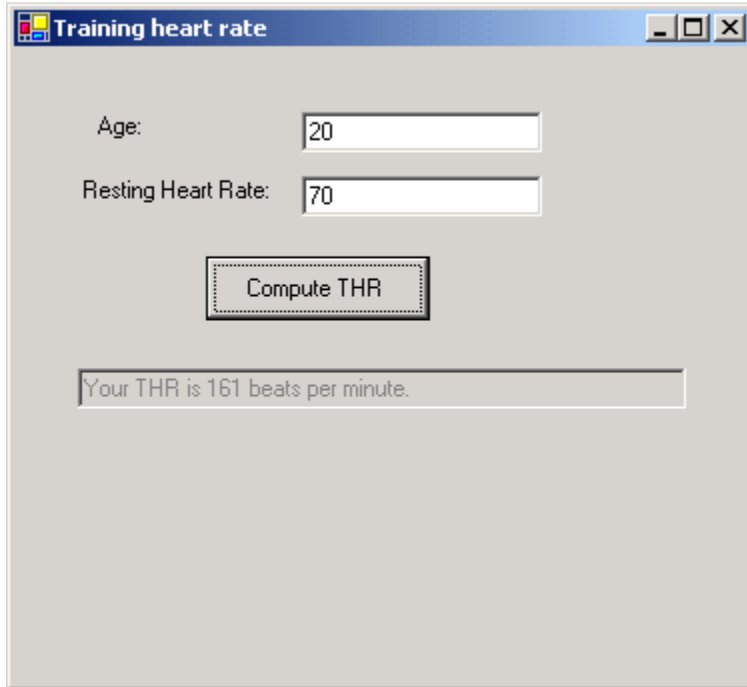
Highlight the text to uncomment and click the icon shown above on the right.

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:



The screenshot shows a window titled "Training heart rate" with a standard Windows-style title bar. Inside the window, there are two labels with corresponding text input fields: "Age:" followed by a field containing "20", and "Resting Heart Rate:" followed by a field containing "70". Below these fields is a button with the text "Compute THR". At the bottom of the window, there is a text area containing the output: "Your THR is 161 beats per minute."

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 mile. 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. $(\text{TotalSeconds mod } 3600) / 60$
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. $(\text{TotalSeconds mod } 3600) \% 60$
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)
```

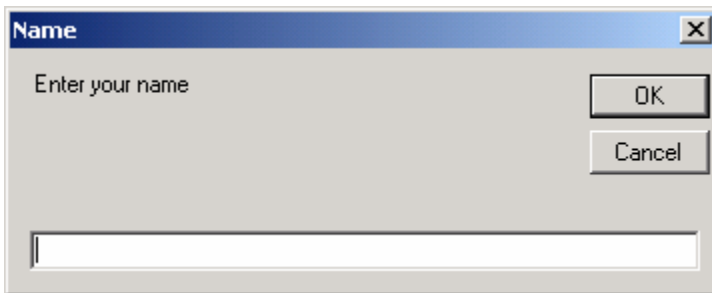
To output data via a popup use a statement of the form:

```
MessageBox.Show(string)
```

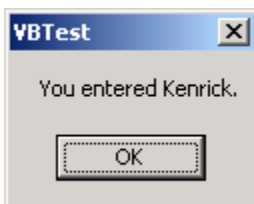
for example:

```
Dim s as String  
s = InputBox("Enter your name", "Name")  
MessageBox.Show("You entered " & s & ".")
```

Generates a window like the following:



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:



If the user presses cancel, then the string returned is empty.

There are additional options on the `InputBox` and `MessageBox` to set the title, icon, and buttons that appear on the pop-up window. See the VB.NET reference for more information.