Problem Solving Using Computers

Variables, Constants, Operators & I/O

Dr. David R. Surma
Department of Computer and Information Sciences

Adapted from Drs. Adaikkalavan, Hakimzadeh & Zhang
Variables:

- Remember this figure.
- **Variables** are the method by which programmers can access and manipulate memory (RAM).
- A variable is the name given to a memory location. (the memory location may hold data items such as numbers, characters, etc.)
Variables

- Programs store and manipulate their data by using variables.
- In most modern programming languages variables must be declared before they are used.
- VB Syntax:
  
  ```vbnet
  Dim variableName As data type
  ```

- Declaring/Creating Variable(s):
  ```vbnet
  Dim studentId As Integer
  Dim middleInit As Char
  Dim address As String
  Dim labScore, examScore, totalScore As Double
  ```
Using Variables, Assignment Operation

- Syntax:

  ```
  variable = expression
  ``

  “=“ is the **assignment operator**

- An expression is a combination of variables, constants, numbers and operators.

- Examples:

  ```
  labScore = 95
  examScore = 85
  totalScore = labScore + examScore + 15
  studentId = 123
  middleInit = ‘A’
  firstName = “Cathy”
  ```
In VB, each variable must have a **data type**. VB provides a number of built-in data types. These include:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>4 bytes</td>
<td>integer</td>
<td>2^{32} values can be represented by 4 bytes (4*8 bits)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>→ 4294967296 (unsigned value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>→ -2147483647 to 2147483648 (signed value)</td>
</tr>
<tr>
<td>Double</td>
<td>8 bytes</td>
<td>floating point or real number</td>
<td></td>
</tr>
<tr>
<td>Decimal</td>
<td>12 bytes</td>
<td>Similar to double but more exact</td>
<td></td>
</tr>
<tr>
<td>Long</td>
<td>8 bytes</td>
<td>Long integer (64 bit integer)</td>
<td></td>
</tr>
<tr>
<td>Char</td>
<td>2 bytes</td>
<td>Unicode character</td>
<td>1 Character</td>
</tr>
<tr>
<td>Byte</td>
<td>1 byte</td>
<td>0 to 255</td>
<td></td>
</tr>
<tr>
<td>Boolean</td>
<td>1 byte</td>
<td>True or False</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>8 bytes</td>
<td>holds time and date</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>variable</td>
<td>holds variable sized strings</td>
<td></td>
</tr>
</tbody>
</table>

Declaring a variable will accomplish the following:

1. Associate the **variable name with some memory location** in which the data will be stored.

2. Tell the compiler **how much memory is needed** for each variable:

3. Allow the compiler to **perform type compatibility checking**. For example the compiler can easily identify problems like:

   ```
   middleInit = 5.8;
   `compiler error` (is char not a floating point number)
   
   middleInit = “hello”
   `compiler error` (is char not a string)
   ```
**Constant Vs. Variable**

Dim varInt as Integer = 10
Dim varStr as String
varInt = 25  'Changes value to 25
varStr = "Hello"  'Changes value to Hello

Const MAX_STU As Integer = 25
Const HELP_STR As String = "HELP"
Const MIN_STU As Integer
MIN_STU = 10  'Error cannot change value

Constant must be initialized and cannot be changed @ runtime
Type Checking

- Traditionally the BASIC language was not a strongly typed language. However, in VB .Net the programmer can explicitly tell the compiler to enforce strong (data) type checking.

- **Strong type checking** can be enforced with the help of the following two statements.
  - **Option Strict On**
    - Because **Option Strict On** provides *strong typing*, prevents unintended type conversions with data loss, disallows late binding, and improves performance, its use is strongly recommended.
  - **Option Explicit On**
    - When **Option Explicit** appears in a file, you must explicitly declare all variables using the **Dim** or **ReDim** statements. If you attempt to use an undeclared variable name, an error occurs at compile time.
Type Compatibility

- The numbers 10 and 10.0 are not the same and therefore, have different types:
  - 10 is an integer number (no fraction part).
  - 10.0 is a floating point number (the decimal point moves around!).

- The compiler will automatically convert an integer number to its equivalent float number however, the reverse is not true.

Example:

labScore = 10  ' Will automatically convert to a float

studentId = 10.5  ' Error if “Option Strict” is turned on.
Arithmetic Operators:

- Remember this figure.
- **Arithmetic operators** are implemented in the Arithmetic Logic Unit of the computer.
Arithmetic Operators

- +    Addition
- -    Subtraction
- *    Multiplication
- /    Real division
- \    Integer division
- Mod  Remainder of an integer division (modulus).
- ^    Exponentiation

Example:

- 5.0 / 2.0 ==> 
- 5.0 / 2  ==> 
- 5 / 2    ==> 
- 5 \ 2    ==> 
- 5 Mod 2  ==> 
- 2 Mod 5  ==> 
I/O CONSOLE APPLICATIONS
Output (Console Applications)

- **Output** can be sent to the computer display by using the `Write()` or `WriteLine()` procedures.

- Syntax:
  
  ```
  Console.Write(argument)
  Console.WriteLine(argument)
  ```

- **Arguments can be:**
  
  - **constant** (string or numeric)
    
    or
  
  - **variable**
    
    or
  
  - **text using “ ”**
Output (Console Applications)

Example 1:

Option Strict On
Option Explicit On

Module Module1
    Sub Main()

        Console.Write(3)
        Console.WriteLine(" {0}", 3 * 5)
        Console.WriteLine("HELLO")

    End Sub
End Module
Output (Console Applications)

Example 2:

```vbnet
Option Strict On
Option Explicit On
Module Module1
    Sub Main()
        Dim temp As Integer = 75
        Console.WriteLine("Temperature is {0} degrees", temp)

        Dim A, B As Integer
        A = 10
        B = 20
        Console.WriteLine("A is = {0} and B is = {1} ", A, B)
    End Sub
End Module

Temperature is 75 degrees
A is = 10 and B is = 20
```

### Example 2:

```vbnet
Option Strict On
Option Explicit On
Module Module1
    Sub Main()

        Dim temp As Integer = 75
        Console.WriteLine("The temperature is {0} degrees", temp)

        Dim A, B As Integer
        A = 10
        B = 20
        Console.WriteLine("A is = {1} and B is = {0} and C is {2}" ,
                                A, B, A+B)

    End Sub
End Module
```

The temperature is 75 degrees
A is = ?? and B is = ?? and C is ??
Input (Console Applications)

- **Input** can be read from the keyboard by using the `ReadLine()` function.

- Syntax:
  
  ```csharp
  Variable = Console.ReadLine()
  ```

  **It is not** `Console.ReadLine(Variable)!!!`

- **Note** that `ReadLine()` will return a **STRING** as the result. So, if a value other than a string is desired the **programmer must perform the conversion.**
Module Module1
    Sub Main()
        Dim Name As String
        Console.Write("What is your name? ")
        Name = Console.ReadLine()
        Console.WriteLine("Good day {0}.", Name)
        Dim Age As Double
        Console.Write("How old are you? ")
        Age = CDbl(Console.ReadLine())
        Console.WriteLine("{0} you are approximately {1} days old.", Name, Age * 365)
        Console.ReadLine()    ' Pause
    End Sub
End Module
I/O GUI APPLICATIONS
Output (Windows Applications)

- Displays a message box that can contain text, buttons, and symbols that inform and instruct the user.

```csharp
MessageBox.Show("Hello")
```

```csharp
MessageBox.Show("Hello",
"Dialog Box Caption",
MessageBoxButtons.OK,
MessageBoxIcon.Exclamation)
```
Input (Windows Applications)

- Displays a prompt in a dialog box, waits for the user to input text or click a button, and then returns a string containing the contents of the text box.

```vbnet
Dim strName As String
strName = Console.ReadLine()
strName = InputBox("What is your name?", "Name")

MessageBox.Show(strName)
```

![Input dialog box example]
VB

- Language Reference

- VB Program Structure
Programming style

- Bottom-line: Make programs easy to read and modify

- Comments:
  - Begin with a single-quote character (‘)
  - Comment your code!

- Indentation
  - Indent blocks of code to convey the program's structure
  - Not a requirement of VB (and most other programming language)
  - To improve program readability
Programming style

- **Identifiers:**
  - names for variables, constants, functions, classes, or other items your might define in a program

- **Identifier naming**
  - A series of characters consisting of letters, digits, and ‘_’
  - Cannot begin with a digit, and cannot contain spaces
  - ALL_CAPS for constants
  - lowerToUpperCase for variables
  - Most important: MEANINGFUL NAMES!
Keywords (reserved words):
- A special type of identifiers
- Such as: `Dim, As, Integer, Char`
- Have predefined meaning, cannot be used as names for variables or anything else.
- The complete list of keywords for VB 2012 can be found at:

Visual Basic is **not case-sensitive**!
- `rate, Rate, RATE, RAtE` are interpreted as the same identifier!!!
Precedence rules

- The precedence of operators decide the order of operations

  - ^
  - *, /,
  - \,
  - Mod
  - +, -
  - =  

  Highest precedence (done first)

  Lowest precedence (done last)

- Operator Precedence in VB:
  - [Link to MSDN](http://msdn.microsoft.com/en-us/library/fw84t893(v=VS.100).aspx)

- Use () to divide an expression into “blocks”. Do the operations inside a block first.