Data and Operations

Outline

- Data Types
- Arithmetic Operations
- Strings
- Variables
- Declaration Statements
- Named Constant
- Assignment Statements
- Intrinsic (Built-in) Functions

Data Types

- In computers data such as integers, real numbers, character, strings, and Boolean are represented and stored in memory differently.
- Computer languages provide appropriate different data types according to the size of storage.
- There are four fundamental data types:
  - Integral types
  - Floating-point numbers
  - Characters
  - String
  - Boolean values

Integral Types

- An integer type has a value as a whole number either negative or positive, e.g., 123, +123, –123, 22333
- Visual Basic .NET has different integral types depends on the size of the storage. For examples,

  `Byte` -- 1 byte

  `Short` -- 2 bytes

  `Integer` -- 4 bytes

  `Long` -- 8 bytes

Floating-Point Numbers

- A floating number or a real number is any signed or unsigned number having an integer part and a fractional part, with a decimal point in between e.g., 18.0, +9.2, 35.25, 0.57, –138.2, 4., .8
- Visual Basic supports two different categories of floating point numbers:
  - **`Single`** -- Numbers which are stored using the single data type are called single precision numbers.
  - **`Double`** -- Numbers that are stored using the double data type are called double precision numbers.
- A double precision number uses twice the amount of storage as that used by a single precision number.

Exponential Notation

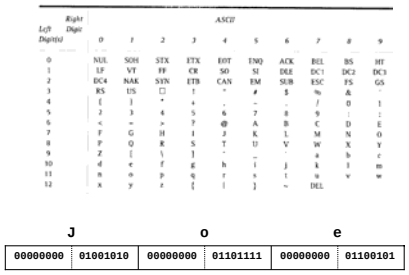
- Exponential notation is commonly used to express both very large and very small floating point numbers in a compact form, e.g., 1.74536E–12, 3.652442E4, 7E20
- It is similar to scientific notation.

Decimal Notation	Exponential Notation	Scientific Notation
1234	1.234E3	1.234×10^{3}
123456	1.23456E5	1.23456×10^{5}
.01234	1.234E –2	1.234×10^{-2}
.0001234	1.234E –4	1.234×10^{-4}

Strings

- A *string* consists of one or more characters that are enclosed within double quotes.
- The number of characters within a string determines the length of the string. For examples, `"apple"`, `"S"`, `"45"`, `"Joe Nobody"` and `"HELLO"`.
- String *concatenation* means the joining of two or more strings into a single string.
- Visual Basic provides two symbols, the ampersand (**`&`**) and the plus sign (**`+`**), for performing string concatenation.
  `"Joe" & " " & "Nobody"` ⇒ `"Joe Nobody"`
- Numeric values and/or strings can be displayed using either a call to the **`MessageBox.Show()`** method or a text box.

ASCII Code



A	01000001	N	01001110
B	01000010	O	01001111
C	01000011	P	01010000
D	01000100	Q	01010001
E	01000101	R	01010010
F	01000110	S	01010011
G	01000111	T	01010100
H	01001000	U	01010101
I	01001001	V	01010110
J	01001010	W	01010111
K	01001011	X	01011000
L	01001100	Y	01011001
M	01001101	Z	01011010

Boolean

- There are only two Boolean data values in Visual Basic.
- These values are the constants: **`True`** and **`False`**.
- The Boolean constants are used in decision-making statements.

Fundamental VB .NET Data Types

Type	Size in Bytes	Description	Range of Values
`Byte`	1	8-bit unsigned integer	0 - 255
`Integer`	4	32-bit signed integer	–2147483648 - 2147483647
`Long`	8	64-bit signed integer	–9223372036854775808 - 9223372036854775807
`Short`	2	16-bit signed integer	–32,768 - 32,767
`Single`	4	32-bit floating point variable	–3.4E38 - 3.4E+38
`Double`	8	64-bit floating point variable	–1.7E308 - 1.7E+308
`Decimal`	16	128-bit floating point variables	
`Char`	2	16-bit Unicode characters	0 - 65,535
`String`	Varies	Non-Numeric Type	
`Boolean`	1	Non-Numeric Type	False or True
`Date`	8	Non-Numeric Type	00:00:00 on January 1, 0001 – 11:59:59 on December 31, 9999
`Object`	4	Non-Numeric Type	Reference to an object

Arithmetic Operations

- Visual Basic provides a number of numeric operators, which can be used to perform operations such as additions, subtractions, multiplications, etc.
- A simple numeric expression is an expression that has a numeric operator connecting two arithmetic operands. This type of expression has the syntax:
  `operand` *`operator`* `operand`
- A binary operator is an operator that requires two operands.
- A unary operator is an operator that requires only one operand.

Arithmetic Operations

+	**Unary Plus**
–	**Unary Negation**
+	**Addition**
–	**Subtraction**
*	**Multiplication**
/	**Division**
\	**Integer Division**
^	**Exponentiation**
Mod	**Modulus -- return remainder**

Precedence of Arithmetic Operations

Operator	Operation
^	**Exponentiation**
–	**Negation**
* /	**Multiplication and Division**
\	**Integer Division**
Mod	**Modulus -- return remainder**
+	**Addition and Subtraction**

Rules of Forming Expressions

The followings are rules when writing expressions in Visual Basic:

- Two binary operators must never be placed adjacent to one another.
  20 * – 40 is invalid.
- Parentheses may be used to form groupings, and all expressions enclosed within parentheses are evaluated first.
- Parentheses cannot be used to indicate multiplication.
  (10 + 20) (30) is invalid.

Expression Types

- *Integer expression* -- is an arithmetic expression containing only integers.
- *Floating point expression* -- is an arithmetic expression containing only floating point numbers.
- *Mixed-mode expression* -- is an arithmetic expression containing both integers and floating point numbers.

Types of Arithmetic Expressions

- combining floating point numbers with either integers or floating point numbers
  ⇒ floating point
- combining integers or floating point numbers with \
  ⇒ integer
- combining integers with either +, –, *,\, `Mod`
  ⇒ integer

Arithmetic Expression

Expression	Result Type
`11 + 4 - 27`	Integer
`11 / 4.6 * 0.879`	Floating Point
`11 / (4.6 * 0.879)`	Floating Point
`11.36 + 4.6 * 0.879`	Floating Point

Integer Division Operator

VB .NET uses back slash (\) as integer division operator.

Expression	Actual Value	Result
`11\4`	`11\4 = 2.75`	`2`
`11\4.6`	`11\5 = 2.20`	`2`
`11.36\4`	`11\4 = 2.75`	`2`
`11.36\4.6`	`11\5 = 2.20`	`2`

`Mod` Operator

The modulus operator `Mod` yields the *remainder* of the result of dividing its first operand by its second.

Expression	Actual Value	Result
`11 Mod 4`	`11 ÷ 4 = 2 r 3`	`3`
`20 Mod 4`	`20 ÷ 4 = 4 r 0`	`0`
`11 Mod 4.6`	`11 ÷ 4.6 = 2 r 1.8`	`1.8`
`11.36 Mod 4.6`	`11.36 ÷ 4.6 = 2 r 2.16`	`2.16`

Variables

- A *variable* is a name given by the programmer to a memory storage location.
- The value stored in a variable can be changed and referenced.
- The rules that the programmer must follow when selecting a variable name are:
  - The name must begin with a letter or an underscore.
  - The name can only consist of letters, numeric digits, or underscores.
  - The name cannot exceed 16,383 characters.
  - The name cannot be a Visual Basic keyword, such as **`Integer`**.
- A variable name should be chosen to indicate what it represents -- `Width`, `Length`, `Area`, `Total_Sum`, `EmployeeIncome`, `Interest_Rate`.

Declaration Statements

- A *declaration statement* is a step used for naming a variable and specifying the data type that can be stored in it.
- Declaration statements placed within a procedure are non-executable statements that have the general syntax:

```
Dim variable-name As data-type
```

where

  - *data-type* designates a Visual Basic data types and
  - *variable-name* is a variable name selected by the programmer.

Examples

```
Dim Number1 As Integer
Dim Wages As Single
Dim Last_Name As String
Dim MiddleIntial As Char
Dim Empty As Boolean
```


Single-line Declarations

Visual Basic allows combining multiple declarations into one statement, using the syntax:

```
Dim var-1 As data-type, var-2 As data-type,...,var-n As data-type
```

Examples:

```
Dim FirstName As String, LastName As data-type,
  MiddleInit As Char, Test1 As Integer, Test2 As
  Integer, Average As Single
Dim Test1, Test2, Test3, FianlExam As Integer,
  Average As Single
```


Initialization

- An *initialization* is the process of storing a value in a variable for the first time.
- In Visual Basic, by default, all numeric variables are initialized to zero and all strings are initialized to zero-length strings (strings containing no characters).
- Declaration statements perform both a software and a hardware function.
  - From a software perspective, declaration statements provide a convenient, up-front list of all variables and their data types.
  - The hardware function that declaration statements perform is that they inform Visual Basic of the physical memory storage that must be reserved for each variable.
- Programmers use variable names to reference their contents.

Named Constants or Literals

- *Constants* are numbers that have a special meaning in the context of a particular application.
- **`Const`** statement is used to assign symbolic names to constants.
- The syntax for this statement within a form's procedure is:
  `Const named-constant As data-type = expression`
- A *named constant* is a symbolic name assigned to a constant.
- A named constant cannot be altered after it is defined.
- The rules for selecting a constant's name are the same as the rules for selecting a variable's name.
- All **`Const`** statements must be placed in the Declarations section of the form or within a procedure, but for clarity they are usually placed immediately after the procedure's declaration

Examples

```
Const HourlyRate As Byte = 15
Const Weekly As Byte = 40
Const PI As Single = 3.141
Const SalesTax As Double = .0825
```

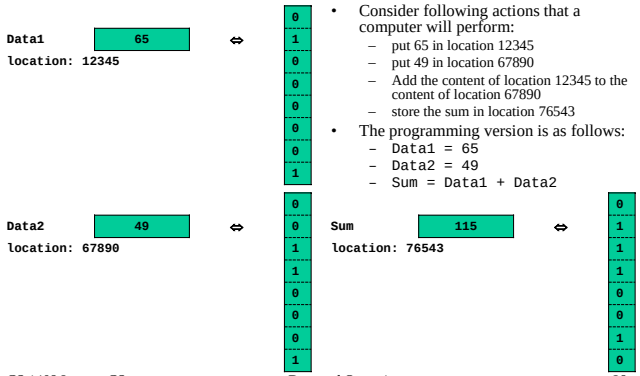

Assignment Statements

- *Assignment statements* are statements that tell the computer to store a value into a variable.
- An assignment statement can be used to both assign a value to a variable and to perform calculations.
- The assignment statement has the following syntax:
  `variable = expression`
- An expression is any combination of constants, variables, and operators that can be evaluated to yield a value.
- Assignment statements always have an equal (=) sign and exactly one variable name immediately to the left of the equal sign.
- The value on the right of the equal sign is assigned to the variable on the left of the equal sign.

Examples

```
Count = 0
Count = Count + 1
Sum = Sum + Data
Area = Height * Width
AverageScore = TotalSum / NemberOfScores
Solution = Math.Sqrt(B^2 – 2*A*C)
```

Assignment Statements



- Consider following actions that a computer will perform:
  - put 65 in location 12345
  - put 49 in location 67890
  - Add the content of location 12345 to the content of location 67890
  - store the sum in location 76543
- The programming version is as follows:
  - `Data1 = 65`
  - `Data2 = 49`
  - `Sum = Data1 + Data2`

Assignment Variations

An *assignment variation statement* is an assignment statement in which the variable on the left of the equal sign is also used on the right of the equal sign. For example :

```
Total = Total + 34.7
Sum = Sum + Number
```


Accumulating

To accumulate subtotals when data is entered one number at a time, assignment statements such as **`Total = Total + 34.7`** are used. These statements have the syntax:

`variable = variable + new value`

Example:

```
SumScore = SumScore + Score
```


Counting

- The counting statement is a variation of the accumulating assignment statement. It has the following form:

  `Variable = Variable + fixed number`
- After a counting statement is executed, the value of the variable is increased by a fixed amount.
- Example:

```
Count = Count + 1
Count += 1
```

Type Conversions

- When an assignment statement is used, the expression on the right side of the assignment operator is converted to the data type of the variable to the left of the assignment operator.
- In addition, Visual Basic also provides for explicit user-specified type conversions.

Examples

Expression	Assigned Value
`Dim Average As Integer` `Average = 98 / 10`	`Average = 10`
`Dim NumberString As Integer` `NumberString = "100"`	`NumberString = 100`
`Dim Number As Single` `Number = 100`	`Number = 100.0`

Function Name and Arguments	Description and Returned Value
Asc(string)	Convert the first character in the string to an ACII value
Chr(val)	Returns string associated with the specified ASCII value.
CBool(expression)	Convert an expression to a Boolean value.
CByte(expression)	Convert an expression to a Byte with rounding if necessary.
CChar(expression)	Convert the first character of a string expression to a Char
CDate(expression)	Convert an expression to a date.
CDbl(expression)	Convert a numeric value or string expression to a Double.
CDec(expression)	Convert a numeric value or string expression to a Decimal.
CInt(expression)	Convert a numeric value or string expression to a Integer.
CLng(expression)	Convert a numeric value or string expression to a Long.
CObj(expression)	Convert an expression to an Object.
CShort(expression)	Convert a numeric value or sting expression to a Short.
CSng(expression)	Convert a numeric value or sting expression to a Single.
CStr(expression)	Convert an expression to a String.
CType(expression, type)	Convert an expression to a specified type.
Val(string)	Return the first numeric value of the string.

Intrinsic (Built-in) Functions

- Intrinsic functions are preprogrammed routines provided by Visual Basic .NET that can be used in a procedure.
- These include mathematical functions, functions to carry out conversion between data types, formatting functions, and string manipulation functions.
- A number of mathematical functions are found in the **System.Math** class of Visual Basic .NET.
- Functions operate in a manner similar to sub procedures, except for the fact that a function always directly returns a *single value*.

Examples

Function Name	Description	Returned Value
`Math.Sqrt(9)`	Square root	`3`
`Math.Abs(-42)`	Absolute value	`42`
`Math.Round(4.779, 2)`	Round	`4.78`
`Math.Log(24.212)`	Log	`3.186...`

Examples

Function Name	Description	Returned Value
`Val("100")`	Convert string to value	`100`
`CStr(100)`	Convert value to string	`"100"`
`Int(99.8)`	Round to nearest integer less than or equal to the number	`99`
`CInt(99.8)`	Round to nearest integer	`100`
`Ceiling(99.8)`	Returns the smallest integer greater than the number	`100`
`Floor(99.8)`	Returns the integer largest less than the number	`99`

Common Programming Errors and Problems

- The most common errors associated with the material presented in this chapter that students should be aware of are:
- Misspelling the name of a method.
- Forgetting to close string messages to be displayed by the **`MessageBox.Show`** method within double quote symbols.
- Incorrectly typing the letter O for the number zero (0), and vice versa.
- Incorrectly typing the letter l, for the number 1, and vice versa.
- Forgetting to declare all the variables used in a program.
- Storing an inappropriate data type value in a declared variable.
- Using a variable in an expression before an explicit value has been assigned to the variable.
- Using an intrinsic function without providing the correct number of arguments of the proper data type.
- Being unwilling to test an event procedure in depth