Chapter 3
VB .NET Programming Fundamentals

Objectives
In this chapter, you will:
• Learn about the VB .NET programming language
• Write a VB .NET module definition
• Use VB .NET variables and data types
• Compute with VB .NET
• Write decision-making statements
• Write loops
• Declare and access arrays

Introducing VB .NET
• VB .NET:
  – Has achieved popularity and widespread acceptance
  – Is a powerful, full-featured, object-oriented development language
  – Is easy to learn and use
  – Supports development of applications for networked environments

Writing a VB .NET Module Definition
• VB .NET code can be structured as a module definition, form definition, or class definition
  – Module definition begins with "Module" and ends with "End Module"
  – Form definition is used to create a GUI
  – Class definition is written to represent an object

• The VB .NET statements consist of keywords and identifiers
  – Keywords have special meanings to VB .NET
  – VB .NET identifiers are the names assigned by the programmer to modules, procedures, and variables, etc.
• VB .NET identifiers:
  – Can be of any length
  – Can include any letter or number, but no spaces
  – Must begin with a letter of the alphabet
Writing a VB .NET Module Definition

• VB .NET code is not case sensitive
• VB .NET compiler does not require indentation of code, but good programming practice encourages indentation
• Comment lines:
  – Add explanations to code
  – Are ignored by compiler
  – Begin with a single quote (’)

Writing a VB .NET Module Definition

• Procedures begin with a procedure header, which identifies the procedure and describes some of its characteristics
• VB .NET has two types of procedures: Function and Sub
  – A Function procedure can return a value
  – A Sub procedure cannot return a value

Writing a VB .NET Module Definition

• Many statements invoke a method to do some work
• Information sent to a method is called an argument
• A literal is a value defined within a statement

Using VB .NET Variables and Data Types

• A variable: named memory location that can contain data
• All variables have:
  – Data type – kind of data the variable can contain
  – Name – An identifier the programmer creates to refer to the variable
  – Value – Every variable refers to a memory location that contains data. This value can be specified by the programmer

Declaring and Initializing Variables

• Before declaring a variable, the programmer must specify its data type
• VB .NET has nine primitive data types:
  – Data types for numeric data without decimals
    • Byte, Short, Integer, Long
  – Data types for numeric data with decimals
    • Single, Double, Decimal
  – Other data types
    • Boolean, Char

Declaring and Initializing Variables

• To declare a VB .NET variable, write:
  – Keyword “Dim”
  – Name to be used for identifier
  – Keyword “As”
  – Data type
• Example:
  ‘ declare a variable
  Dim i As Integer
Declaring and Initializing Variables

• A value can be assigned by the programmer to a variable
• Assignment operator (=) assigns the value on the right to the variable named on the left side
• Example:
  - populate the variable
  i = 1

Declaring and Initializing Variables

• The code to both declare and initialize a variable can be written in one statement:
  - declare a variable
  Dim i As Integer = 1
• Several variables of the same data type can be declared in one statement:
  Dim x, y, z As Integer

Changing Data Types

• Option Strict helps prevent unintentional loss of precision when assigning values to variables
• If Option Strict is set to On, whenever an assignment statement that may result in a loss of precision is written, VB .NET compiler displays an error message

Changing Data Types

• With Option Explicit On, the programmer must define a variable before using it in a statement
• Option Explicit is generally set to On

Using Constants

• Constant: variable with a value that does not change
• Code to declare a constant is identical to the code to declare a variable, except:
  - Keyword “Const” is used instead of “Dim”
• Constants must be initialized in the statement that declares them
• By convention, constant names are capitalized

Using Reference Variables

• There are two kinds of variables
  - Primitive variable
    • Declared with a primitive data type
    • Contains the data the programmer puts there
  - Reference variable
    • Uses a class name as a data type
    • Refers to or points to an instance of that class
    • Does not contain the data; instead, it refers to an instance of a class that contains the data
Using Reference Variables

![Diagram](image)

Figure 3-4: Contrasting primitive and reference variables

Computing with VB .NET

- VB .NET uses:
  - Arithmetic operators (+, -, *, /) for addition, subtraction, multiplication, and division
  - Parentheses to group parts of an expression and establish precedence
  - Remainder operator (Mod) produces a remainder resulting from the division of two integers
  - Integer division operator (\) to produce an integer result
  - Caret (^) for exponentiation

Math class contains methods to accomplish exponentiation, rounding, and other tasks

To invoke a Math class method, write:
  - Name of the class (Math)
  - A period
  - Name of the method
  - Any required arguments

Writing Decision-Making Statements

- Decision-making statements evaluate conditions and execute statements based on that evaluation
- VB .NET includes: If and Select Case statements
- If statement:
  - Evaluates an expression
  - Executes one or more statements if expression is true
  - Can execute another statement or group of statements if expression is false

- Select Case statement:
  - Evaluates a variable for multiple values
  - Executes a statement or group of statements, depending on contents of the variable being evaluated

VB .NET If statement interrogates a logical expression that evaluates to true or false
- An expression often compares two values using logical operators
Writing If Statements

**Simple If**
- Evaluates an expression
- Executes one or more statements if expression is true

**If-Else**
- Evaluates an expression
- Executes a second statement or set of statements if expression is false

Table 3.6 VB .NET Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
</tbody>
</table>

Writing Select Case Statements

**Select Case statement**
- Acts like a multiple-way If statement
- Transfers control to one of several statements, depending on the value of an expression
Writing Loops

- Loops: repeated execution of one or more statements until a terminating condition occurs
- Three types of loops:
  - Do While
  - Do Until
  - For Next

Writing Do While Loops

- Use a Do While to display the numbers 1-3:
  'do while loop
  declare and initialize loop counter variable
  Dim i As Integer = 1
  Do While i <= 3
    Console.WriteLine("do while loop: i = " & i)
    i += 1
  Loop

Writing Do Until Loops

- Do While loop continues executing the statement as long as the expression evaluates to true
- An infinite loop: loop that does not terminate without outside intervention
- While loop
  - A variation of the Do While loop
  - Functions like the Do While loop

Writing Do Until Loops

- A Do Until loop:
  'do until loop
  i = 1 ' re-initialize loop counter variable
  Do Until i > 3
    Console.WriteLine("do until loop: i = " & i)
    i += 1
  Loop

Writing Post-Test Loops

- Difference between a Do While and Do Until loop:
  - Do While loop executes while the expression is true
  - Do Until loop executes until the expression is false

- Programming languages provide two kinds of loops:
  - Pre-test loop tests the terminating condition at the beginning of the loop
  - Post-test loop tests the terminating condition at the end of the loop
Writing Post-Test Loops

• Do While and Do Until loops can be written as either pre-test or post-test loops
• For Next and While loops are always pre-test

Writing For Next Loops

• VB .NET For Next loop:
  – Includes loop counter initialization and incrementing code as a part of the For statement
  – Uses pre-test logic – it evaluates the terminating expression at the beginning of the loop
• Example:
  ```csharp
  ' for next loop
  For i = 1 To 3 Step 1
    Console.WriteLine("for next loop: i = " & i)
  Next
  ```

Writing Nested Loops

• Nested loop:
  – A loop within a loop
  – Can be constructed using any combination of Do While, Do Until, or For Next loops

Declaring and Accessing Arrays

• Arrays: create a group of variables with the same data type
• In an array
  – Each element behaves like a variable
  – All elements must have the same data type
  – Elements either can contain primitive data or can be reference variables

• Arrays can be either one-dimensional or multi-dimensional
  – One-dimensional array consists of elements arranged in a row
  – Two-dimensional array has both rows and columns
  – Three-dimensional array has rows, columns, and pages
• VB .NET implements multi-dimensional arrays as arrays of arrays
Using One-Dimensional Arrays

- Declare a 5-element array with of integers:
  ' declare an integer array with 5 elements
  Dim testScores(4) As Integer
- Individual array elements are accessed by writing
  the array reference variable, followed by the
  index value of the element enclosed in
  parentheses

Using One-Dimensional Arrays

- Code to initialize the array elements:
  testScores(0) = 75
  testScores(1) = 80
  testScores(2) = 70
  testScores(3) = 85
  testScores(4) = 90
- An array can be declared and populated using a
  single statement:
  Dim testScores() As Integer = {75, 80, 70, 85, 90}

Using One-Dimensional Arrays

- Conceptually
  - A two-dimensional array is like a table with rows
    and columns
  - A three-dimensional array is like a cube, with rows,
    columns, and pages
- Each dimension has its own index
- Declare an Integer array with five rows and two
  columns
  Dim testScoreTable(4, 1) As Integer

Using Multidimensional Arrays

- Code to populate the array:
  ' populate the elements in column 1
  testScoreTable(0, 0) = 75
  testScoreTable(1, 0) = 80
  testScoreTable(2, 0) = 70
  testScoreTable(3, 0) = 85
  testScoreTable(4, 0) = 90
  ' populate the elements in column 2
  testScoreTable(0, 1) = 80
  testScoreTable(1, 1) = 90
  testScoreTable(2, 1) = 60
  testScoreTable(3, 1) = 95
  testScoreTable(4, 1) = 100

Using Multidimensional Arrays

- Code to initialize the array elements:
  testScores(0) = 75
  testScores(1) = 80
  testScores(2) = 70
  testScores(3) = 85
  testScores(4) = 90
- An array can be declared and populated using a
  single statement:
  Dim testScores() As Integer = {75, 80, 70, 85, 90}
### Summary

- An identifier is the name of a class, method, or variable
- All variables have a data type, name, and value
- VB .NET has nine primitive data types
- VB .NET has two kinds of variables: primitive variables and reference variables
- Math class has methods to accomplish exponentiation, rounding, etc.
- VB .NET provides two types of decision-making statements: If statement and Select Case statement

### Summary

- You write VB .NET loops using one of three keywords: Do While, Do Until, or For Next
- There are two kinds of loops: pre-test loop and post-test loop
- A nested loop is a loop within a loop
- A one-dimensional array consists of elements arranged in a single row
- A two-dimensional array has both rows and columns