Introduction

- This chapter covers the use of text boxes to gather input from users.
- It also discusses the use of:
  - variables
  - named constants
  - intrinsic functions
  - mathematical calculations
  - format menu commands
  - the Load procedure of a form

3.1 Gathering Text Input

In This Section, We Use the Textbox Control to Gather Input That the User Has Typed on the Keyboard

TextBox Control

- TextBox
  - A rectangular area on a form that accepts input from a keyboard or mouse
  - Tutorial 3-1 provides an example in the use of a TextBox

Text Property of a TextBox

- User
  - Can change Text property by typing in TextBox

- Programmer
  - Can change Text property via an assignment statement
  - Syntax:
    ```
    txtInput.Text = "Type your name"
    ```
Text Property of a TextBox

- Programmer
  - Can use Text property to retrieve value entered by user
    - Display user entered value in a Label control:
      `lblSet.Text = txtInput.Text`
    - Display user entered value in a pop-up message
      `MessageBox.Show(txtInput.Text)`

String Concatenation

- Combine two or more strings into one string
- Concatenation operator:
  - The `&` (ampersand) character
- Examples:
  ```csharp
  txtName.Text = txtFirst & " " & txtLast
  MessageBox.Show("Hello " & txtFirst & " " & txtLast);
  ```

Concept: Focus

- When a control has Focus
  - It accepts input from user
  - The control may:
    - Ignore user input
    - Pass user input to a container object (e.g., Form)
- In an active Form
  - One and only one control can have the focus at any point in time
  - Which controls can have focus?
    - Only controls capable of receiving user input
    - Control has a TabStop property
Buttons and Keyboard Access Keys

- To “click” a Button using the keyboard
  - Button must have a keyboard access key
  - Example:
    ```vbnet
    btnSave.Text = "&Save"
    ```
  - Ampersand causes next letter to be keyboard access key
  - “Click” the button using Alt-S on the keyboard
  - Be careful not to use same access key for two different Buttons

'S' in Button Text Property

- Displaying an ampersand in Button Text
  ```vbnet
  btnFiveTen.Text = "Five && Dime"
  ```

Accept and Cancel Buttons

- Accept Button
  - Implicitly “clicked” if user hits ENTER Key
  - Specified via Form AcceptButton Property
- Cancel Button
  - Implicitly “clicked” if user hits ESCAPE Key
  - Specified via Form CancelButton Property
- In both cases
  - Name of button is Property value

Variables

An Application Uses Variables to Hold Information So It May Be Manipulated, Used to Manipulate Other Information, or Remembered for Later Use

What is a Variable?

- A name given to a storage location in computer’s memory
  - Used for holding data while program is running
  - Data stored in variable may change, hence “variable”
- What are variables used for?
  - Copy and store values entered by user
  - Perform arithmetic
  - Test its value
  - Temporarily hold and manipulate a value
  - Remember data for later use in program

Steps in Creating and Using Variables

1. Make up a name for variable
2. Declare variable name
   - VB associates name with a location in computer’s RAM
3. Use variable name in various VB statements
   - Uses current value of variable as stored in memory location
Variable Naming Rules

- First character of a variable name
  - Must be a letter or underscore
- Subsequent characters
  - May be a letter, underscore, or digit
- Thus variable names cannot contain
  - spaces or periods
- Other special characters (e.g., period, comma)
- Also
  - VB keywords cannot be used as variable names

Variable Naming Conventions
(Guidelines to improve code readability)

- A variable name should:
  - Describe its use (purpose)
  - Begin with the recommended prefix
  - More on this in a moment
  - Capitalize 1st letter of each subsequent word in variable name
- Examples:
  - intHoursWorked
  - strLastName

Declaring a Variable

Variable Declaration

- A statement that associates a name with a memory location
- Syntax (three variations):
  - Dim VarName As DataType
  - Dim VarName As DataType = InitialValue
  - Dim VarName1, VarName2, ..., VarNameN As DataType
- Dim A VB keyword (short for Dimension)
- VarName Programmer-defined variable name
- As A VB keyword

Visual Basic Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Amount of memory</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>2 bytes</td>
<td>True, False</td>
</tr>
<tr>
<td>Byte</td>
<td>1 byte</td>
<td>0 thru 255</td>
</tr>
<tr>
<td>Char</td>
<td>2 bytes</td>
<td>single character (Unicode-16)</td>
</tr>
<tr>
<td>Date</td>
<td>8 bytes</td>
<td>00:00:00 1/1 thru 11:59:59 12/31/9999</td>
</tr>
<tr>
<td>Decimal</td>
<td>16 bytes</td>
<td>up to 20 significant digits</td>
</tr>
<tr>
<td>Double</td>
<td>8 bytes</td>
<td>up to 15 significant digits</td>
</tr>
<tr>
<td>Integer</td>
<td>4 bytes</td>
<td>-2^31 thru 2^31-1</td>
</tr>
<tr>
<td>Long</td>
<td>8 bytes</td>
<td>-2^63 thru 2^63-1</td>
</tr>
<tr>
<td>Object</td>
<td>4 bytes</td>
<td>memory address</td>
</tr>
<tr>
<td>Short</td>
<td>2 bytes</td>
<td>-2^15 thru 2^15-1</td>
</tr>
<tr>
<td>Single</td>
<td>4 bytes</td>
<td>up to 7 significant digits</td>
</tr>
<tr>
<td>String</td>
<td>2 bytes per character</td>
<td>Unicode-16</td>
</tr>
</tbody>
</table>

VB Data Types and Prefixes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Prefix</th>
<th>Data Type</th>
<th>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>bin</td>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>Byte</td>
<td>byt</td>
<td>Long</td>
<td>ling</td>
</tr>
<tr>
<td>Char</td>
<td>chr</td>
<td>Object</td>
<td>obj</td>
</tr>
<tr>
<td>Date</td>
<td>dat</td>
<td>Short</td>
<td>shr</td>
</tr>
<tr>
<td>Decimal</td>
<td>dec</td>
<td>Single</td>
<td>sng</td>
</tr>
<tr>
<td>Double</td>
<td>dbl</td>
<td>String</td>
<td>str</td>
</tr>
</tbody>
</table>

Use Variable in VB Statements

- Dim intNumber As Integer
- Dim intTwiceNumber As Integer
- Dim strFullName As String

- Assignment Statement
  - Change value of variable
    - intNumber = 100
    - strFullName = "Alexander Graham Bell"
  - Use value of variable
    - intTwiceNumber = intNumber - 2
    - txtFullName.Text = strFullName
Scope of Variable

- **Scope**
  - Part of program where variable is visible
  - May be accessed by VB statements
- **Begins**
  - At variable declaration statement
- **Extends**
  - Local Variable
  - A variable declared inside a procedure
  - End of procedure where declared
  - Rules
    - Cannot be declared a 2nd time in same procedure

Lifetime of Variable

- **Lifetime**
  - Time during which it exists in memory
  - Storage for a variable is
    - Created when it is declared
    - Destroyed when procedure finishes executing

Variable Default Value

- When variable declared, it is assigned a default value
  - Numeric types given value of zero
  - Strings given value of Nothing
    - Nothing – a VB keyword
    - String by default has no value
  - Dates given value of 12:00:00 AM January 1, 1

Data Conversions in VB

- **Implicit Type Conversions**
  - An attempt to convert to receiving variable’s data type
    - `Dim intNumber As Integer = 100`
    - `Dim dblNumber As Double`
    - `dblNumber = intNumber`
    - `intNumber = "200"`
    - `dblNumber = "1.5"`
- **Explicit Type Conversions**
  - Use of a VB intrinsic (built-in) conversion function
    - More on this in a moment

Implicit Type Conversions

- A value of one data type may be assigned to a variable of a different type
- **Widening Conversion**
  - Suffers no loss of data
    - Examples
      - Byte to Short, Integer, Long, Single, or Double
      - Short to Integer, Long, Single, or Double
      - Integer to Long, Single, or Double
- **Narrowing Conversion**
  - May lose data
    - Examples
      - Short to Byte
      - Integer to Short or Byte
      - Long to Integer, Short, or Byte

Option Strict

- A VB configuration setting
  - See Project, Properties…, Compile settings
- **When Option Strict is On**
  - Allows only widening conversions
- **Option Strict On**
  - Recommended to help catch errors
  - Default setting when you create new project
Type Conversion Runtime Errors

- Example of narrowing conversion
  ```vbnet
dim intCount as Integer = "abc123"
  ```
- If Option Strict On
  - Statement will not compile
- If Option Strict Off
  - Statement compiles, but:
    - String "abc123" will not convert to an integer
    - A runtime error (type mismatch) occurs when statement executed

Explicit Type Conversions

- A VB function performs some predetermined conversion operation
- VB
  - Provides set of functions that permit narrowing conversions with Option Strict On
  - Each function:
    - Accepts a constant, variable name, or arithmetic expression
    - Returns the converted value

VB Conversion Functions

General Syntax

- `functionName ( expr )`

Convert to Whole Number

- `CByte`, `CShort`, `CInt`, `CLong`
  - `expr` - can result in number or string value
  - `functionName` - rounds to nearest whole number
  - Examples:
    ```vbnet
dim bytNumber as Byte = CByte("1.6")
dim shrNumber as Short = CShort(10.6)
dim intNumber as Integer = CInt(16.0 / 10.0)
dim lngNumber as Long = CLng(100.6)
  ```

Convert to Real Number

- `CDbl`, `CDec`, `CSng`
  - `expr` - can result in number or string value
  - Examples:
    ```vbnet
dim dblNumber as Double = CDbl("1.06")
dim decNumber as Decimal = CDec("$1,234.99")
dim sngNumber as Single = CSng("1,234.99")
  ```

Convert to Boolean

- `CBool ( expr )`
  - `expr` - a number, string that represents a number, "True", or "False"

Convert to Character

- `CChar ( expr )`
  - `expr` - a string value

Convert to Date

- `CDate ( expr )`
  - `expr` - a valid date literal (more on this later)

Convert to Object

- `CObj ( expr )`
  - `expr` - any valid expression

Convert to String

- `CStr ( expr )`
  - `expr` - a number, Boolean, Date, or string value

Invalid Conversions

- Conversion functions can fail
- Examples
  - String "xyz" can't be converted to a number:
    ```vbnet
dim dblSalary as Double = CDbl("xyz")
  ```
  - No day 35 in May:
    ```vbnet
dim datHired as Date = CDate("05/35/2005")
  ```
- Failed conversions cause an **invalid cast exception runtime error**
Val Function
- A more forgiving way of converting a string to a number
- General Syntax
  - Val (stringExpr)
- Semantics
  - When initial characters form a numeric value
  - Val function returns that value
  - Otherwise, returns zero
- Examples
<table>
<thead>
<tr>
<th>String</th>
<th>Val</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;34.90&quot;</td>
<td></td>
<td>34.9</td>
</tr>
<tr>
<td>&quot;86abc&quot;</td>
<td>86</td>
<td></td>
</tr>
<tr>
<td>&quot;124.95&quot;</td>
<td></td>
<td>124.95</td>
</tr>
<tr>
<td>&quot;3,789&quot;</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

ToString Method
- Returns a string representation of the value in the variable
- Every VB data type has a ToString method
- Uses dot notation
- Example
  ```vbnet
dim number as integer = 123
dblNumber.text = number.ToString()
```

Date Data Type
- Holds date and time value
- Date literal enclosed within # symbols
- Examples
  ```vbnet
dim datStart as date
datStart = #10/20/2005 6:30:00 AM#
datStart = #12/10/2005#
datStart = #21:15:02#
```
- Can use function to convert String to Date
  ```vbnet
datStart = System.Convert.ToDateTime("12/3/2002")
datStart = System.Convert.ToDateTime(txtDate.Text)
```

Retrieving Current Date/Time
- VB keyword
- Purpose
- Example
  ```vbnet
dim datCurrent as date
now = now
timeofday = timeday
today = today
datCurrent = now
datCurrent = timeofday
datCurrent = today
```

Common Arithmetic Operators
- VB Arithmetic Operators:
  - + Addition
  - - Subtraction
  - * Multiplication
  - / Division
  - ^ Exponentiation
  - \ Integer Division
  - Mod Modulo (remainder)
Integer Division Operator \ \n• Result is always an integer
• Any remainder is discarded
Option Strict Off
• Floating-point operands rounded to nearest integer
Option Strict On (default project setting):
• Floating-point operands not allowed
Allowed:
\ CInt(15.0) \ CInt(5.0)  
Not Allowed:
15.0 \ 5.0

Modulo (Mod) Operator
• Produces remainder as the result of a division operation
\ intRemainder = 17 MOD 3 'result is 2
\ dblRemainder = 17.5 MOD 3 'result is 2.5

DivideByZeroException
• Any attempt to divide by zero using / or \ or Mod

Combining Assignment and Arithmetic Operators
Special assignment operators provide easy means to perform common operations:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Usage</th>
<th>Equivalent To</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>x += 2</td>
<td>x = x + 2</td>
<td>Add to</td>
</tr>
<tr>
<td>-=</td>
<td>x -= 5</td>
<td>x = x – 5</td>
<td>Subtract from</td>
</tr>
<tr>
<td>*=</td>
<td>x *= 10</td>
<td>x = x * 10</td>
<td>Multiply by</td>
</tr>
<tr>
<td>/=</td>
<td>x /= y</td>
<td>x = x / y</td>
<td>Divide by</td>
</tr>
<tr>
<td>&amp;=</td>
<td>name &amp;= last</td>
<td>name = name &amp; last</td>
<td>Concatenate</td>
</tr>
</tbody>
</table>

Literal Constants
Programs often use predetermined values
• Add 6% sales tax to an order total
decTotal *= 1.06

Two problems with using literal constants
• Reason for multiplying decTotal by 1.06 not obvious
• When sales tax rate changes, must find and change every occurrence of .06 or 1.06
• Also need to determine if use of literal constant represents the sales tax rate or some other value
Thus, use named constants

Named Constants
• General Syntax
const constName As dataType = initialValue
• constName – same rules as variable names
• dataType – a VB data type
• initialValue – an expression that results in a value of dataType
• Semantics
• Value associated with constName cannot be changed at runtime
• Programming Convention
constName is all uppercase; words separated with underscore
Const sngSALES_TAX_RATE As Single = 1.06
decTotal *= sngSALES_TAX_RATE
A Well-Engineered Program Should Report Errors and Try To Continue Or Explain Why It Can’t Continue and Then Shut Down. Use Exception Handling to Recover Gracefully from Errors

3.4

Exception Handling

We’ve discussed two possible runtime errors
- DivideByZeroException
- InvalidCastException

A runtime error results when:
- Visual Basic throws an exception
- And it is an unhandled exception

Exception handling
- Allows program to catch and respond to runtime errors
- Program can either fail gracefully and recover from error (if possible)

Runtime Errors

VB Exception Handler

A simple form that ignores some options:

Try
  try-block
  Catch [exception-type]
    catch-block
End Try

try-block
- Contains statements that might throw an exception

exception-type
- Identifies type of runtime exception that should be caught
- Optional part of statement
  - When not specified, Catch will capture any type of exception thrown

catch-block
- Contains statements to execute if exception is caught

VB Exception Handling Example

Try
  Dim decSalary as Decimal
  decSalary = CDec(txtSalary.Text)
  MessageBox.Show("Your salary is ", 
                   decSalary & " dollars")
  End Try

Any exception thrown by a statement in try-block will result in catch-block being executed

CDec may throw an InvalidCastException
- This exception is caught, and execution continues with statements in the catch-block

More Exception Handling Features

Can catch specific types of messages
- Can capture and show the exception message issued by Visual Basic

Try
  Dim decAnnualSalary as Decimal
  Dim intPayPeriods as Integer
  Dim decSalary as Decimal
  decAnnualSalary = CDec(txtAnnualSalary.Text)
  intPayPeriods = CInt(txtPayPeriods.Text)
  decSalary = decAnnualSalary / intPayPeriods
  lblSalary.Text = decSalary.ToString()
  End Try

Message Boxes

An easy way to notify user via a pop-up window

Basic Syntax
- MessageBox.Show (String msg )
- MessageBox.Show ( String msg, String caption )
  - More syntax forms exist

Operation:
- Displays a pop-up window with:
  - First syntax form: message and OK button
  - Second syntax form: message, caption, and OK button
- Message appears in body of window
- Caption appears in title bar of window
Message Box Example

- Following code displays message box shown below

```csharp
MessageBox.Show("Please try again, and enter a number", "Entry Error")
```

3.5 Formatting Numbers for Output

Numbers May Be Formatted in Various Ways for Output

VB Number Formats Functions

- `FormatNumber`
  - Include commas and specified number of decimal places
  - `FormatCurrency`
    - Format as currency with dollar sign or other currency symbol
    - `FormatPercent`
      - Display number as a percent
      - `FormatDate`
        - Format number as a date, time, or both
  - The computer’s regional settings determine some format items such as currency symbol

FormatNumber Function

- **Purpose**: Format a number for display
- **General Syntax**
  ```csharp
  FormatNumber(Expression [, DecimalPoints])
  ```
  - `Expression`: Evaluated and output as a number with commas and a decimal point
  - `DecimalPoints`: Optional argument; specifies number of decimal places to display
    - If not specified, decimal positions default to 2
  - **Operation**: Decimal positions not shown are rounded
  - **Examples**
    - `FormatNumber(3921.387)` returns "3,921.39"
    - `FormatNumber(.75)` returns "0.75"

FormatCurrency Function

- **Purpose**: Used to format a number for display as a currency figure such as dollars and cents
- **General Syntax**
  ```csharp
  FormatCurrency(Expression [, DecimalPoints])
  ```
  - `Expression`: Evaluated and returned with commas, decimal point, and currency symbol
  - **Operation**: Decimal positions not shown are rounded
  - **Examples**
    - `FormatCurrency(3921.387)` returns "$3,921.39"
    - `FormatCurrency(.75)` returns "$0.75"

FormatPercent Function

- **Purpose**: Used to format a number for display as a percent
- **General Syntax**
  ```csharp
  FormatPercent(Expression [, DecimalPoints])
  ```
  - `Expression`: Evaluated and returned with % sign
  - **Operation**: Decimal positions not shown are rounded
  - **Examples**
    - `FormatPercent(78.47)` returns "78.47%"
    - `FormatPercent(820)` returns "820%"
3.6 Group Boxes, Form Formatting, and the Load Event Procedure

In this section, we discuss the GroupBox control, which is used to group other controls, and how to align and center controls on a form.

FormatDateTime Function

- **Purpose**
  - Displays a date in various formats
- **General Syntax**
  - `FormatDateTime(Expression [, Format])`
- **Expression**
  - Must evaluate to a Date data type
- **Format**
  - Optional argument
  - Specifies desired format, examples:
    - DateFormat.GeneralDate: "4/7/2006 3:22:18 PM"
    - DateFormat.LongDate: "Friday, April 7, 2006"
    - DateFormat.ShortDate: "4/7/2006"
    - DateFormat.LongTime: "03:22:18 PM"
    - DateFormat.ShortTime: "15:22"

Group Box

- A container control
- Creates a logical and physical grouping of controls
- **Physical**
  - Controls are surrounded by a box and have a title
  - It's apparent to the user that controls within a GroupBox are related
- **Logical**
  - Controls in box have their own tab ordering within GroupBox
  - Moving a GroupBox also moves its controls
  - Removing a GroupBox also removes its controls

Placing Controls in a Group Box

- Can create controls in any order
- If GroupBox created first
  - Select GroupBox control
  - Then, either:
    - Double-click control in ToolBox to place control in GroupBox
    - Click control in ToolBox and draw control inside GroupBox
- If other controls created first
  - Create GroupBox control
  - Then, either:
    - Drag controls into GroupBox
    - Select control, cut from form, select GroupBox, paste into GroupBox

Events associated with a Form

- **Load**
  - Occurs before Form is displayed for the first time
  - Can create a Form_Load event procedure by double-clicking on the form in the designer window
  - Put initialization logic that should be done only once in this event
- **Activated**
  - Occurs when Form has become the active form
- **Deactivate**
  - Occurs when Form loses focus and is no longer the active form