CSE 1530

Introduction to Computer Use II: Programming

Winter 2006 (Section M)

Topic E: Subprograms – Functions and Procedures

Monday, March 13 2006

Bill Kapralos

Overview (1):

- **Before We Begin**
  - Some administrative details
  - Some questions to consider
- **Input Validation**
  - Introduction
  - Example
- **Miscellaneous Notes**
  - Control Arrays
  - "White-space"

Before We Begin

Administrative Details (1):

- **No Lab Exercise to Submit This Week**
  - Nothing to submit Monday, March 20
- **Correction For Next Week's Exercise**
  - Submit Exercise 6-8 and not Exercise 6-9 as stated on the website
- **Test 2 Reminder**
  - Wednesday, March 15 2006
  - Be on time, the test will start at 1:30pm

Some Questions to Consider (1):

- What is the purpose of an argument list ?
- How do we specify a function's argument list ?
- How do we specify a procedure's argument list ?
- For a function's argument list, do we need to specify whether the arguments are passed by reference or passed by value ?

Input Validation

Introduction (1):

- **Erroneous User Input**
  - Many programs require user input of some form or another
    - We cannot always guarantee the validity of user entered data → cannot assume it is always valid!
  - Typically, a program will rely on and require the user entered data to proceed and complete its task
  - It is up to the programmer to handle erroneous data entered by the user
    - Need to take the appropriate course of action when invalid data is entered

Introduction (2):

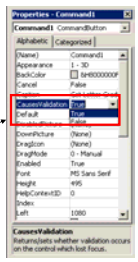
- **Erroneous User Input (cont.)**
  - Many times we do not wish to proceed with the execution of the program until we are "certain" that all entered data is valid
    - How about performing some form of check on any entered data and only proceeding once we can determine the data is valid ?
    - If we find the data is not valid, then we can keep prompting the user to re-enter the data until it is valid

Introduction (3):

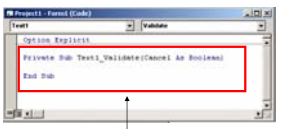
- **Validating User Entered Input**
  - Visual Basic contains a "built-in" construct to allow you to easily validate user input data
    - Most controls have a property called CauseValidation
    - In addition, controls also have an event called Validate
  - CauseValidation and Validate work together and allow for a simple manner of checking (validating) user entered data

Introduction (4):

- **Validating User Entered Input (cont.)**



CausesValidation property of an object → default is set to True



Validate event handler for some TextBox called "Text1"1 → initially it is empty hence, even though default Causes Validation is True, indicating function will be called, no actions will be performed!

Introduction (5):

- **Validating User Entered Input (cont.)**
  - But how can we keep from making such a construct specific to a particular type of data ? For example, numeric data → Consider the following
    - What about indicating to Visual Basic that the user input entered in a particular control (e.g., TextBox) needs to be validated and
    - When the user enters the data via the control (e..g, TextBox), an event is generated and a particular event handler is called/executed → of course, as with all event handlers, we as programmers write the code for the event handler specific to our needs

Introduction (6):

- **Validating User Entered Input (cont.)**
  - This is essentially the scheme that we will use in Visual Basic → lets assume all data will be entered via a TextBox
    - As soon as user enters the data in the TextBox and then focus is shifted from the TextBox to some other control, provided the CausesValidation of the other control is set to True, the Validate event of the TextBox will be called/executed
    - When the CausesValidation property is False, of course the Validate event handler is not called

Introduction (7):

- **Validating User Entered Input (cont.)**
  - The event handler is of course application specific
    - Can be written to handle any input needs we require
  - It is also up to the programmer to determine which controls will have their CausesValidation property set so that the Validate event handler is called
  - Up to the programmer to write the code of the Validate event handler

Introduction (8):

- **Validating User Entered Input (cont.)**
  - Lets take a closer look at the Validate event handler

```
Private Sub Text1_Validate(Cancel As Boolean)
  ...
End Sub
```

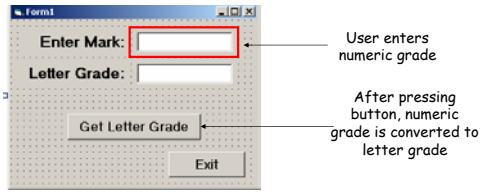
  - Contains an argument called "Cancel" of type Boolean
    - If the data entered by the user is invalid, then set "Cancel" to True (this tells VB that focus should remain on the TextBox since user needs to re-enter data) otherwise set it to False

Some Notes (1):

- **Multiple Validate Event Handlers**
  - What happens if we have multiple controls for user to enter data (e.g., multiple TextBoxes) – which Validate event handler is called ?
    - The validate event handler for the control that last had focus

Data Validation Example (1):

- **Lets Look at an Example**
  - Simple program that converts number grades (percent) into corresponding letter grades
  - User enters numeric grade, presses button and letter grade is given



Data Validation Example (2):

- **Lets Look at an Example (cont.)**
  - So we want to basically ensure that the user entered data is a number between 0 and 100
    - We will set the CausesValidation property of the "Get Letter Grade" button control to True so that once it obtains focus (e.g., the user presses it), we call the Validate event handler of the TextBox (the one user enters the data in)
    - Of course we must provide the implementation (e.g., relevant statements) for the Validate event handler!

Data Validation Example (3):

- **Lets Look at an Example (cont.)**
  - Set CausesValidation property of the button to True
    - But there is also the "Exit" button and the default for its CausesValidation property is also True indicating that we will validate user input upon pressing it → is this really necessary ? NO!
    - Set the CausesValidation property of the "Exit" button to False
  - Complete the code for the Validate event handler

Data Validation Example (4):

- **Validate Event Handler**

```
Private Sub Text1_Validate(Cancel As Boolean)
  Cacel = False
  If (IsNumeric(Text1.Text)) Then
    Dim value As Integer
    value = CInt(Text1.Text)
    If ((value < 0) Or (value > 100)) Then
      Cancel = True
    End If
  Else
    Cancel = True
  End If
End Sub
```

Miscellaneous Notes

A Look Back at Control Arrays (1):

- **Additional Property of Control Arrays**
  - Recall that a control array contains a specific number of elements (e.g., Option control objects)
  - Every control array also includes a property called Count
    - Set up by Visual Basic when you declare the array
    - Specifies the number of elements the control array contains
    - Not visible in the properties window since this property is not a property of the elements within the array but rather specific to the array itself

A Look Back at Control Arrays (2):

- **Additional Property of Control Arrays (cont.)**
  - We can use this property as any other property
    - It is nothing more than a value (number)
  - Consider the following example
    - Assume a control array called myOptionArray
    - The following will allow us to set the "Value" property of each option control to True

```
Dim loopIndex As Integer
For loopIndex = 0 To myOptionArray.Count-1
  myOptionArray(loopIndex).Value = True
Next
```

"White-Space" (1):

- **What is "White-Space" ?**
  - Extra space that we add within our program to illustrate the program's structure
    - Allows to easily locate "blocks" of code (e.g., If statements, loops, subprograms) within program

```
For loopIndex = 1 to 100
    optionArray(loopIndex) = loopIndex
Next
```

    - Makes the code easy to follow/debug/update etc. especially when you consider very long programs written by multiple programmers

"White-Space" (2):

- **How Do We Add "White-Space" ?**
  - No general rule describing how much white-space is necessary The main point is to be consistent throughout!
    - For example, if you decide you will use three spaces, then use three spaces always to offset all "blocks" of code

```
Function myFunction() As Integer
  Dim loopIndex As Integer
  For loopIndex = 1 To 100
    text1.Text = CStr(oopIndex)
  Next
End Function
```