



CSE 1530
**Introduction to Computer Use II:
Programming**
Winter 2006 (Section M)
Topic E: Subprograms - Functions and Procedures
Monday, March 6 2006
Bill Kapralos

CSE 1530, Winter 2006, Bill Kapralos

Overview (1):

- **Before We Begin**
 - Some administrative details
 - Some questions to consider
- **Topic Overview**
 - Introduction
- **Subprograms**
 - Introduction
 - Function subprograms
 - Function example

Before We Begin

Administrative Details (1):

- **Lab Exercise**
 - You should be working on Ex 6-3 this week
 - Due Monday, March 13
- **Test 2 Reminder**
 - Wednesday, March 15 2006
- **Course Drop-Deadline**
 - Last day to withdraw from course is Friday, March 10 2006

Some Questions to Consider (1):

- Describe the Replace function
- Describe the InStr function
- Describe the Len function
- Describe the "Mid" function

Topic Overview

Introduction (1):

- **So Far, Two Alternatives to Sequential Programming → If Statements and Loops**
 - These are however not the only alternatives!
 - Another departure from sequential programming is a **sub-program** (function, method or procedure)
 - While executing a set of statements, and call to a subprogram is encountered, execution of those statements is interrupted
 - Execution of subprogram statements occurs and when subprogram statements have been executed, return back to original set of statements and continue at point after call to subprogram

Introduction (2):

- **We Have Already Encountered Subprograms**
 - We have made use of many subprograms up until this point, including the following
 - All the string-related functions → "Mid", "Len", "InStr", "Replace" etc...
 - AddItem from the ListBox
 - Date-related functions
 - Format function
 - But up until this point, the subprograms (functions) have been given to us
 - We simply use them without worrying about them!

Introduction (3):

- **Overview of Topic E**
 - We will examine subprograms (functions) in detail
 - We will learn how to write our own subprograms
 - Main concepts of the this topic
 - Abstraction and modularization
 - Function subprograms
 - Procedure (or Sub) subprograms
 - Transferring values via an argument list
 - The scope of variables

Subprograms

Introduction (1):

- **What is a Subprogram ?**
 - A convenient way to encapsulate some computation that can be then used many times over without worrying about its implementation
 - Allows us to ignore *how* a job is done
 - All we need to know is *what* is done (e.g., the outcome)
 - Can be used by many other programs as well

Introduction (2):

- **Why Use Subprograms ?**
 - Separate the performance of some task from the rest of the program
 - In designing a large program, its usually best to "**divide and conquer**" → break the task down into a number of pieces, each of which can be programmed separately
 - Imagine having to compute some computation many times → you can replicate the code many times or you can write the code once within a function and simply call the function

Introduction (3):

- **Why Use Subprograms ? (cont.)**
 - Break large sections of code into smaller units that perform a specific task
 - By breaking your calculations into smaller tasks
 - Simplify maintenance that needs to be done to the program in the future
 - Make the code easier to read/follow and troubleshoot

Introduction (4):

- **Subprograms are "Connected" to the Program That Calls Them**
 - They must usually use data from the calling program
 - Two ways that data from the calling program can be made available in the subprogram
 - Transferred to the subprogram via an argument list (arguments)
 - Global variables are also accessible within subprogram

Introduction (5):

- **Specific Types of Subprograms**
 - We have already encountered various subprograms
 - **Event handlers** → called in response to a user interaction via the GUI (e.g. `command1_Click()`)
 - **Functions** → Called whenever it is encountered during program execution (e.g. `Mid(inputTxt, position, 1)`)
 - **Methods** → a subprogram that is associated with a particular class/object and in fact the method can only be called via the object (e.g., `listBox.AddItem(myString)`)

Introduction (6):

- **We Will Divide Subprograms Into Two Categories**
 - **Function Subprograms**
 - Restricted to computing and returning a single result only
 - Restricted
 - **Procedure subprogram**
 - More "freedom" to perform "greater" operations
 - For the remainder of the lecture, we will focus on function subprograms

Function Subprograms (1):

- **Purpose**
 - Calculates a some **specific single result**
 - Separate that calculation from the rest of the program code
 - Can perform this specific calculation many times by simply calling function within program
 - Depending on how the function is defined, it may also be called within different programs → the built in functions of VB are an example
 - Function should do nothing else except calculate a single result → shouldn't change object properties or modify global variables for example

Function Subprograms (2):

- **Promote Modularization**
 - Functions allow you to separate a well defined piece of some calculation
 - That piece of calculation becomes represented by the name of the function
 - Think of the larger problem independently of the piece represented by the function
 - This is known as **modularization**
 - **Divide and conquer** → dividing the task into smaller, well defined pieces or **modules** such that you can focus your thinking on smaller, more manageable tasks

Function Subprograms (3):

- **The Result of a Function**
 - A function (subprogram) can only calculate a single value
 - The value may be an integer, real number, string, boolean etc.
 - A function is essentially an expression and can therefore be used in the same places that a variable or expression might be used
 - For example, a function may be used on the right hand side of an assignment statement → myValue= Round()

Function Subprograms (4):

- **Defining a Function**
 - Syntax
Private Function functionName(argument list) As resultDataType
function body (statements)
End Function
 - **Private, Function, As and End Function**
 - Key words
 - **functionName**
 - The name of the function that you provide
 - The name should be meaningful and represent the calculation performed by the function

Function Subprograms (5):

- **Defining a Function (cont.)**
 - **(Argument list)**
 - The argument list is optional however the parenthesis are not → they must be used even if there are no arguments
 - **resultDataType**
 - Specifies the data type of the result returned by the function (e.g., Integer, Single, Double...)

Function Subprograms (6):

- **Defining a Function (cont.)**
 - **Function body**
 - Statements that ultimately calculate the result
 - Must assign the result to the function name → therefore, within the function body itself, the following statement must appear

functionName = ...
 - The function name is treated as if it were a normal variable name
 - Function body may contain local variable declarations and may use any global variables

Function Example (1):

- **Compute a Sum**
 - Consider a function that will compute (and return) the sum of the numbers in the range 1-100
 - Function name → computeSum
 - Arguments → none
 - Return data type → Integer
 - Function definition
Private Function computeSum() As Integer
...
End Function

Function Example (2):

- **Compute a Sum (cont.)**
 - Here is the Visual Basic code for the function

Private Function computeSum() As Integer
Dim loopIndex As Integer
Dim sum As Integer
sum = 0

For loopIndex = 1 To 100
sum = sum + loopIndex
Next
computeSum = sum
End Function

Function Example (3):

- **Compute a Sum (cont.)**
 - Here is another (equivalent) version of the function
 - What is the difference ?

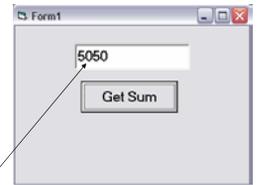
```
Private Function computeSum() As Integer
    Dim loopIndex As Integer
    computeSum = 0

    For loopIndex = 1 To 100
        computeSum = computeSum + loopIndex
    Next
End Function
```

Function Example (4):

- **Compute a Sum (cont.)**
 - Lets use the function now
 - Call it in the button Click event handler

```
Private Sub btnSum_Click()
    Dim sum As Integer
    sum = computeSum()
    txtSum.Text = sum
End Sub
```



Output after pressing button