

CHAPTER 6

Sub Procedures and Function Procedures

FORM MODULES

- Project is made up of modules which consist of smaller parts-procedures
- Types of Procedures:
 1. Event procedures respond to events
 - Pressing a button or clicking a check box for example
 2. VB procedure pre-packaged Common Tasks (Format\$, If, or LoadPicture)
 3. Sub procedures allow programmers to create their own procedures
 4. Function procedures resolve problems by returning a value or result

FORM MODULES

- Programming tips
 1. use VB procedures when appropriate
 2. VB procedures usually execute faster
 3. each procedure should efficiently execute task
 4. procedures should be no longer than one page
- Procedure is invoked (performing its task) by a procedure call

FORM MODULES

- Call** specifies procedure name and provides arguments the **callee** needs
- Procedures allow for software reuseability
- Procedures can be invoked from several points in program
- Syntax error- writing executable code outside a procedure

SUB PROCEDURES

- Created with Add Procedure (Tools menu)
- Grayed out unless code window is visible
- Terminate when End Sub reached
- Program execution continues after call
- Parentheses may be empty or contain a parameter list
- Private Sub PrintPay(hours As Single, Wage As Currency)
Print hours * wage
- End Sub

SUB PROCEDURES

- Declares two parameter variables in parameter list
- Parameter variables are declared using As or type-declaration character
 - (not explicitly given a type, defaults to **Variant**)
- Receive values from procedure call and used in procedure body

SUB PROCEDURES

- Syntax errors:
 - 1. Giving variable the same name as procedure name
 - 2. Not passing correct number of arguments to procedure
 - Example call PrintPay 40,10.00
 - Note1: procedure using many arguments maybe too complex
 - Note2: VB rounds real numbers when converting to integer

SUB PROCEDURES

- Run-time error= passing arguments that cannot be implicitly converted
- Example : Call PrintPay(40,10.00)
- Can also pass variables: Call PrintPay(x,y)
- Note3: when calling Sub procedure, use Call and enclose arguments in parentheses
- Syntax error3: not using call and enclosing arguments in parentheses

SUB PROCEDURES

- Syntax error4: when using call, not enclosing arguments in parentheses
- Syntax error5: declaring in procedure body variable with same name as parameter variable
- Code example: figure 6.5, p. 183
- **Auto list members**- automatically displays an object's properties and methods
- (Tools, Options)

SUB PROCEDURES

- When **period** is typed after an object name, window lists **properties** and **methods** for that object
- See figure 6.6, p. 185
- Can quickly find property or method
- **Auto quick info**- displaying procedure information for VB procedures and user-defined procedures
- Tools, Options see figure 6.7, p. 185

FUNCTION PROCEDURES

- Returns a value to caller
- Most VB procedures are Functions
- User-defined procedures created with Add Procedure
- Example: Private Function IsVolunteer88()
 ' mIdNumber is module variable
 IsVolunteer = IIf(mIdNumber,True,False)
- End Function

FUNCTION PROCEDURES

- When function procedure encountered at run-time, body of statements are executed
- Invoke procedure: returnValue = IsVolunteer88()
- Function IsVolunteer88 returns either True or False to returnValue
- Program execution then continues with execution of next statement

FUNCTION PROCEDURES

- All function procedure definitions contain parentheses
- Example: Private Function Area(s1 As Single, s2 As Single)
Area = s1 * s2
End Sub
- Area's return type is variant
- squareFtNeeded = Area(8.5, 7.34)
- s1,s2
- Function with no arguments needs no parentheses in Call

FUNCTION PROCEDURES

- Function return types:
 1. Explicitly stated in procedure header As keyword
 2. Or type declaration character
- Example:
 1. Private Function Area(s1 As Single, s2 As Single) As Single
 2. Private Function Area!(s1 As Single, s2 As Single)

FUNCTION PROCEDURES

- Example: Private Function IsVolunteer88() As Boolean(no type declaration)
- Programming example: Interface, figure 6.9,
- P. 188, specifications, figure 6.10, p. 188-89
- Code , figure 6.11, p. 189
- CommandButton property Style determines whether or not graphics can be drawn on command button

FUNCTION PROCEDURES

- When default standard is employed, graphics cannot be drawn on button
- When graphical setting is used, graphics can be drawn on button
- VB constants vbButtonStandard and vbButtonGraphical used in code to explicitly set property
- Example: cmdButton.Picture = LoadPicture (name)>draws image on button

FUNCTION PROCEDURES

- Picture property specifies which image cmdButton displays
- Style property set to Graphical
- Vb function LoadPicture loads specified image into memory
- value returned from LoadPicture is assigned to button's picture property
- result>image is drawn on cmdButton

INVOKING PROCEDURE CALLS

- Call-by-Value vs. Call-by-Reference:
- Arguments passed in programs to this point use Call-by-reference (default)
- Call-by-reference: called procedure has ability to directly access caller's data
- Can also modify that data if called procedure chooses

INVOKING PROCEDURE CALLS

- Call-by-value: copy of argument's value is passed
- Called procedure can manipulate that copy
- Cannot change Caller's data
- Can weaken security because called procedure can change caller's data at any time
- Called-by-value: prevents accidental side effects and is more reliable

INVOKING PROCEDURE CALLS

- Implementing Call-by-values:
- (1). Using keyword ByVal
- (2). Placing arguments in parentheses
- Precede corresponding parameter variables in procedure definition with ByVal
- Note1: if not used, call-by-reference assumed)
- Note2: when passing arguments via Call-by-Value, use optional parentheses around arguments passed

INVOKING PROCEDURE CALLS

- Note3: use ByVal in procedure header
- Logic error: assuming ByVal applies to more than one parameter
- Syntax error: Using ByVal or ByRef outside a procedure header
- Example: Function Foo(ByVal x As Long, y As Boolean) As Double –ByRef is default
- Call1: doubleValue=Foo(passACopy, passOriginal)

INVOKING PROCEDURE CALLS

- Call2: doubleValue = Foo((passACopy), passOriginal)
- Code example: Figure 6.12, p. 191-193
- Interface specifications: Figure 6.12, p. 193
- Explanation: p. 193-194
- Exit Sub and Exit Function can be used to alter flow
- Exit Sub causes immediate exit from a procedure (control returned to caller execute next statement in sequence)
- Exit Function immediate exit from function, control returned to caller, next statement executed

STORAGE CLASSES

- Used identifiers for variable names.
- Attributes of variables: name, type, size, and value.
- Identifiers were used for programmer defined procedure's other attributes-storage class and scope.
- Storage class- determines period during which identifier exists in memory (lifetime varies).
- Scope- region in a program in which identifier can be referenced: (1) throughout program; (2) limited to portions of program.

STORAGE CLASSES

- Storage Class Specifiers:
1. **Automatic variables**- created when the procedure becomes active and exist until procedure is exited (contents are not preserved)
 2. **Static variables**- storage is allocated and initialized once when form is created
 3. **Static procedures**- all variables in procedures are Static

STORAGE CLASSES

Example:

```
Private Static Sub AllVariablesAreStatic()
```

```
    Dim q, a, s;
```

```
    ...
```

```
End Sub
```

Note1: place Static before Sub

Note2: Tools>Add Procedure>check All Local Variables as Static

STORAGE CLASSES

Types Of Identifiers:

1. **External**- module variables and procedure names
2. **Local Variables**- declare with storage class specifier **Static**
 1. Only known in procedure defined
 2. Retain their value when procedure is exited
 3. Next procedure call, static variable contains value it had when procedure last exited

STORAGE CLASSES

All numeric variables of Static storage class are initialized to zero

Example: Static count As Integer

```
    Static Dim count As Integer (ok, but  
    Dim statement is redundant)
```

Syntax error: declaring Static variable outside procedure

STORAGE CLASSES

Scope Rules:

Region of project in which identifier can be represented

Three Scopes for identifier:

1. Local applies to variables declared in procedure's body (End Sub or End Function)
2. Module applies variables declared in general declarations
3. Public variables are accessible to all modules

Note1: identifier in outer scope (module) has same name as identifier in inner scope

STORAGE CLASSES

Note2: outer scope identifier is hidden until inner scope terminates

Note3: inner scope sees value of own identifiers, not value of identically named identifier in outer scope

Note4: storage duration does not affect scope of identifier

Code example: Figure 6.15, p. 199-200

Interface: p. 200

RANDOM NUMBER GENERATOR

Function Rnd returns single random number in range $0 \leq \text{Rnd} < 1$

Multiple calls to Rnd should not produce same set of numbers

Example: dieFace = Int(6 * Rnd()) – produce integers in range 0 to 5

Called scaling, where 6 is scaling factor

Shift range: dieFace = 1 + Int(6 * Rnd())

Function Int returns Integer part of argument passed

Example: passing 8.567 to Int returns 8

RANDOM NUMBER GENERATOR

Interface for 6-sided dice: Figure 6.16, p. 202

Code: Figure 6.17, p. 203

Note1: controls, as with variables, can be passed to procedures

Modified interface counting occurrences of dice:

Interface: Figure 6.18, p. 204

Code: Figure 6.19, p. 205-206

Note2: Rnd generates pseudo-random numbers

RANDOM NUMBER GENERATOR

Note3: Repeated calls to Rnd produces sequence that appears random

(repeated sequence on each execution of program)

Randomizing- conditioning program to produce random numbers

Randomize statement seeds Rnd to produce a different sequence of random numbers on each execution

RANDOM NUMBER GENERATOR

Seed- initial number random numbers generator uses to produce series of random numbers

Code to randomize die program:

Figure 6.21, p. 207

Call Randomize- obtains value for seed from VB function Timer

Seed for random numbers generator uses current time (seconds elapsed since midnight)

RANDOM NUMBER GENERATOR

$n = a + \text{Int}(b * \text{Rnd}())$

Where: a = shift adjustment (1st numbers in range)

b = scaling factor (equal to width of desired range)

Example: A Game of Chance (craps)

Rules of game- p. 208

Interface: Figure 6.22, p. 211

A GAME OF CHANCE

Specifications: Figure 6.23, p. 211

Code: Figure 6.24, p. 212-213

Frame control- groups other controls placed inside frame

Controls cannot be separated from Frame

Enumeration: user-defined type (Enum)

Values of enumeration constants (name constants) start a 0 and incremented by 1

Note: can specify another start value

A GAME OF CHANCE

Example code: Enum DiceNames Assign value

snakeEyes = 2 (2)

trey (3)

[yo leven] = 11 (11)

boxcars (12)

End Enum

Enumeration constants (ex. yo leven) containing one or more spaces enclosed in braces

A GAME OF CHANCE

Syntax errors:

1. Attempting to define an enumeration in scope other than module scope
2. Attempting to modify an enumeration constant's value

Form_Load

```
Icon = LoadPicture ("d:\images\ch06\die.ico")
```

End Sub

Form_Load: event procedure called by VB when form is created at run time

A GAME OF CHANCE

Code changes image the form title displays to a die

Set Icon property to icon image(graphics file>.ico)

Game Sequence:

1. Play begins a new game of craps
2. Event procedure cmdPlay_Click calls RollDice
3. Sum of dice returned from RollDice and assigned to sum

A GAME OF CHANCE

4. Select Case determines outcome of first roll
5. 7 or 11 disables Roll button, displays message player has lost
6. 2, 3, or 12 disables Roll button displays message player has lost
7. Point occurred, Case Else is executed, Play disabled, Roll enabled, and pressed when attempting to make point

A GAME OF CHANCE

8. RollDice - roll dice, compute, and return sum
9. DisplayDice - utility procedure to handle drawing dice images
10. Game won lblStatus.Caption displays game has been won or lost, otherwise Roll pressed 2 to roll dice

RECURSION AND THE FACTORIAL FUNCTION

Recursive procedure - calls itself either directly or indirectly through another call

Used to solve a problem

Solve only simplest case(s) or base case(s)

If called with base case, recursion is stopped and returns to caller

More complex problem, divides into two parts:

1. Known solution
2. Unknown solution

RECURSION AND THE FACTORIAL FUNCTION

For recursion to work:

Part2 must be smaller or simpler than original problem

Based on similarity, procedure launches (call) fresh copy of itself to work on smaller problem (recursive call and recursion step)

Recursion step either Exit Sub or Exit Function executes while original call is still open (executing)

RECURSION AND THE FACTORIAL FUNCTION

Recursion continues to simplify problem until base case is reached

Result is passed up line to original procedure call which returns final result

Example: factorial of nonnegative Integer n, n! (n factorial)

$$n * (n-1) * (n-2) * \dots * 1$$

$$1! = 0$$

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

RECURSION EXAMPLE: 5 FACTORIAL

■ Push on Stack

1!
2!
3!
4!
5!

Popping the Stack

Returns 1 (1!=1)
Returns 2 (2 * 1)
Returns 6 (3 * 2)
Returns 24 (6 * 4)
Returns 120 (24 * 5)

ANOTHER RECURSIVE EXAMPLE

```

Option Explicit
Private Sub cmdCalculate_Click()
    Dim b as Integer, exp as Integer
    b = txtBase.Text
    exp = txtExp.Text
    lblDisplay.Caption = "Result: " & IntegerPower(b, exp)
End Sub
Private Function IntegerPower(base As Integer, ___
    exponent As Integer) As Integer
    If exponent = 1 Then
        IntegerPower = base
        Exit Function
    End If
    IntegerPower = base * IntegerPower(base, exponent - 1)
End Function
    
```