Sub Procedure and Function Procedures

Making code shorter, manageable, robust, and reusable

Divide and Conquer

- programs for real world problems are usually large and complex
- large and complex programs can be constructed from smaller pieces
- each piece can be composed of smaller pieces
- a piece of code is implemented using a procedure

Organization of code in VB

- Modules
  - Procedures
    - sub procedures (*)
    - function procedures (*)
    - event procedures (*)
    - VB procedures

(*) programmers can write their own. User-defined event procedures are part of user-defined classes.
Examples of Procedures

• sub procedures
  Private Sub number_key(number as double)
  End Sub
  call number_key(x)

• function procedures
  Private Function pictorial(number as double)
    pictorial = pictorial(number - 1)
  End Function
  y = pictorial(x)

Examples of Procedures (cont.)

• event procedures
  Private Sub cmdOK_Click()
  End Sub

Examples of Procedures (cont.)

• VB procedures
  – Format
  – Iif
  – Load
  – MsgBox
Analogy of procedure and function calls

- Boss calls workers 1, 2, 3
- Workers 1, 2, 3 return the call (reporting)
- Workers 1, 2, 3 call Workers 4, 5, 6
- Workers 4, 5, 6 return calls (reporting)

Tips and General Rules

- Do not reinvent the wheels. Become familiar with VB procedures
- VB procedures normally more efficient than the ones you write
- Each procedure should be limited to one single task - well defined and effectively expressed
- A procedure should be shorter than one page

Adding Procedures
Specify a procedure

Tips on procedure names

- If you cannot choose a concise name to express what the procedure does, it is possible that it's doing too many things
- Choosing meaningful procedure and variable names help readability and avoid excessive comments
- Too many arguments may indicate too many tasks performed by a procedure

Calling a sub procedure

Private sub printPay(hours as Single, wage as Currency)
    Print hours * wage
End Sub

printPay 40 10.00
or
Call printPay(40, 10.00)
    – preferred form
    – makes sub call stand out (improve readability)
Option Explicit

- used at module level to force explicit declaration of all variables in that module
- must appear in a module before any procedures
- when used, all variables must be declared using the Dim, Private, Public, ReDim, or Static statements.
- attempting to use an undeclared variable name will lead to an error at compile time.

Option Explicit

- if not used, all undeclared variables are of Variant type unless the default type is otherwise specified with a DefType statement.
- use to avoid incorrectly typing the name of an existing variable
- use to avoid confusion in code where scope of the variable is not clear.

Call-by-Value vs. Call-by-Reference

- default is call-by-reference
- call-by-reference allows the procedure being called to modify the values of the arguments
- call-by-value passes a copy of the arguments’ values to the procedure
- choice is based on trade-off of security and performance
Specifying Call-by-Value or Call-by-Reference

Function Foo(ByVal x as Long, y as Boolean) As Double
Function Foo(ByVal x as Long, ByRef y as Boolean) As Double
  ByVal applies only to one argument
doubleValue = Foo(a, b)  '* copy of a passed
  '* original of b passed
doubleValue = Foo((a), b)  '* parentheses optional
  - use the optional parentheses to make call-by-value clear

Exit Sub and Exit Function

- Exit statement may be used anywhere in a sub or function to immediately terminate the execution of the sub or function
- Some programmers believe exit statements violate the spirit of structured programming by allowing multiple exit points
- properly used, exit offers better performance

Storage class (life time)

- an identifier can
  - exist briefly
  - repeatedly created and destroyed
  - exist throughout a program execution
- storage class determines the life time of an identifier
Two kinds of storage classes

- **Automatic**
  - default
  - created when the procedure in which they are defined become active
  - exist until procedure is exited
  - memory for automatic variables reclaimed
- **Static**
  - specify with keyword Static
  - when Static applied to procedure, all variables declared in procedure are static
  - created, initialized and unchanged once the form/module is created

Static variables

- Only use when a variable’s value must be maintained between procedure calls

```vbnet
Private Function accidents (intNumb as Integer)
    Static totalAccidents As Integer
    totalAccidents = totalAccidents + intNumb
    accidents = totalAccidents
End Function
```

Public vs. Private - Scope

- **local scope**
  - applies to variables declared in a procedure
  - local variables can be used in that procedure
- **module scope**
  - applies to variables and procedures in a module
  - can be accessed within the module
- **public scope**
  - accessible to all modules
Public Variables

- Public variables in different modules sharing the same name can be distinguished by prefixing the module names

  Module1        Module1. intX
  Module2        Module2. intX
  Form1          Form1. intX

Optional Variables

Private Sub K(optional a As Integer, _
  optional b As Integer, _
  optional c As Integer)

  Call K
  Call K(0, 0, 20)
  Call K(1)
  Call K(20)