

## CHAPTER 6 PART II

### RECURSION AND THE FACTORIAL FUNCTION

## RECURSION AND THE FACTORIAL FUNCTION

Nonrecursive: factorial = 1

For counter = number To 1 Step -1

factorial = factorial \* counter

Next counter

Recursive definition:  $m! = n * (n-1)!$

Recursive function is Factorial:

Factorial = number \* Factorial (number-1)

Code example: Figure 6.26, p. 216-217

## RECURSION AND THE FACTORIAL FUNCTION

Factorial receives parameter type Double and returns type Double

Double can hold largest when values assigned to a variable-type are exceeded

Note: omitting base case or writing recursion step incorrectly will cause infinite recursion using up all available memory

Another recursion example: Fibonacci series is unending sequence where term is defined as sum of its two predecessors; 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Series occurs in nature and describes form of spiral

## RECURSION AND THE FACTORIAL FUNCTION

The ratio of successive Fibonacci numbers converges on a constant value of 1.618 (Golden ratio or Golden mean)

Data type Double is necessary for the Fibonacci numbers

Code: Figure 6.27, p.218

Interface: Figure 6.28, p.219

screen.MousePointer = vbHourglass  
changes mouse pointer to hourglass icon

## RECURSION AND THE FACTORIAL FUNCTION

Screen object represents environment display  
mousepointer- set shape of mouse pointer

Note: each level of recursion in Fibonacci has a doubling effect on number of calls

Example>calculating only 20<sup>th</sup> Fibonacci number would require on order of  $2^{20}$  or million calls (exponential complexity)

avoid recursive programs that result in exponential explosion of calls

## RECURSION AND THE FACTORIAL FUNCTION

Recursion vs. Iteration:

Control>iteration uses repetition structure  
recursion uses selection structure

Repetition>iteration explicitly uses a repetition structure

recursion does repetition through repeated procedure calls

## RECURSION AND THE FACTORIAL FUNCTION

Termination test>iteration terminates when loop-condition fails

recursion terminates when base case is recognized

Iteration keeps modifying a counter until counter assumes value makes loop-continuation condition fail

Recursion keeps producing simpler versions of original problem until base case reached

## RECURSION AND THE FACTORIAL FUNCTION

Both can occur infinitely

Recursion negatives:

1. Repeatedly invokes mechanism of procedure calls
2. Expensive in both processor time and memory
3. Each call causes another copy of procedure (data) to be created (consumes a lot of memory)

## RECURSION AND THE FACTORIAL FUNCTION

Iteration occurs within a procedure, overhead of repeated calls and extra memory is omitted

Choose recursion when it mirrors the problem and results in clearer program which is easier to understand and debug

Note1: avoid using recursion in performance situations (calls take time and consume memory)

## RECURSION AND THE FACTORIAL FUNCTION

Note2: good software engineering is important and high performance is important (many times are at odds)

Good software engineering is key to making more manageable task of developing larger and more complex software systems

High performance is key to realizing systems of future place ever-greater computing demands of hardware

## OPTIONAL ARGUMENTS

Procedures take one or more optional arguments

Caller has option of passing that particular argument

Example: Private Sub FooBar(y As Boolean, Optional z as Long)

Any procedure calling FooBar must pas at least one argument to FooBar to avoid syntax error

## OPTIONAL ARGUMENTS

Caller can pass a second argument to FooBar

Call FooBar (syntax error)

Call FooBar (True) (valid)

Call FooBar (False, 10) (valid)

Syntax error:

1. Not passing a required argument to a procedure
2. Attempting to use default argument without Optional keyword

## OPTIONAL ARGUMENTS

Code example: Figure 6.29, p.222

Interface: p. 223

Declaring Optional variables in a procedure header, all variable declarations to right of Optional variable must be declared optional

Example: Private Sub Bad(Optional x As Integer = 1, y As Integer, z As Integer = 1)

syntax error: declare non-optional

## NAMED ARGUMENTS

Simplify procedure calls with many Optional arguments

Programmer specifies in Call name of argument and value passed into argument

Example: Private Sub K(Optional a As Integer, Optional b As Integer, Optional c as Integer)

K's caller only wants to pass value of 20 to c:

Call K(0,0,20) or Call K(, ,20)

## NAMED ARGUMENTS

Simplified: Call K(c:=20)

A named argument is specified in a procedure Call by writing parameter variable name followed by := and value

Auto quick info allows programmer to see parameter variable name

Syntax error:

1. Using = instead of :=
2. Using variable name for named argument not declared in procedure header

## NAMED ARGUMENTS

Logic error: local variable has same name as named argument specified in Call

Can be used with procedures that do not specify Optional arguments

Private Sub Display(flag As Boolean, number As Long, message As String)

Call Display(message:="Named Arguments", Flag:=True, number := 1000000)

Any order for 3 arguments is possible since named arguments are used

## NAMED ARGUMENTS

Example code: Figure 6.30, p. 224-225

Interface: p.225

Visual Basic Math Functions

Figure 6.31, p. 225-226

Code Modules (standard)

Has no GUI only code

Created by selecting Add Module from Project menu (Figure 6.32, p. 226)

Promote software reusability

Package procedures for reuse in multiple modules

Will have a .bas extension

## Code Modules

Code modules are maintained in separate folders in **Project Explorer Window**

Figure 6.33, p. 227

Activate: (1) doubling-clicking; (2) highlight and press View Code button

Code module has a general declaration

Code file names have .bas extension

Project can have multiple code modules

Can be added or removed from project

## Code Modules

By default, module variables and event procedures are Private to module declared

Private variables or procedures can only be used within module declared

Public variables or procedures can be used in other modules in project

Utility procedures are generally Private

Syntax errors:

1. Using dim with Public or with Private

## Code Modules

2. Attempting to access Private variables or call Private procedures in another module

3. Attempting to declare local variables as Private or Public

4. Declaring a Public procedure in one code module that has same name as another procedure in another code module in same project

Code example calling Public procedure: Figure 6.34, p. 228