TODAY'S QUOTE

- "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are–by definition–not smart enough to debug it."

(Brian Kernighan)

Admin Stuff

- Assignment #1 due
- Assignment #2
  - Posted to website
  - Due Oct 11
- Midterm
  - Review session Oct 16
  - Midterm in class Oct 18 [2 hours long]

Logical Operators

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
=	equal to
<>	not equal to

ASCII values are used to decide order for strings

CONDITION

- Decisions are a result of evaluating a condition
- A *condition* is an expression involving relational and/or logical operators
- Result of the condition is Boolean

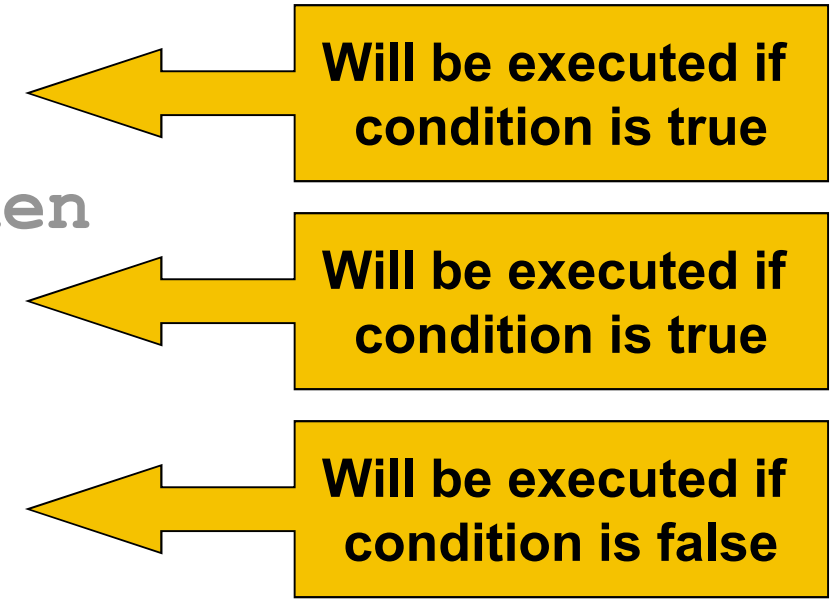
Logical Operators

- Used with Boolean expressions
- *Not* – makes a False expression True and vice versa
- *And* – will yield a True if and only if both expressions are True
- *Or* – will yield a True if at least one of both expressions are True

If Block

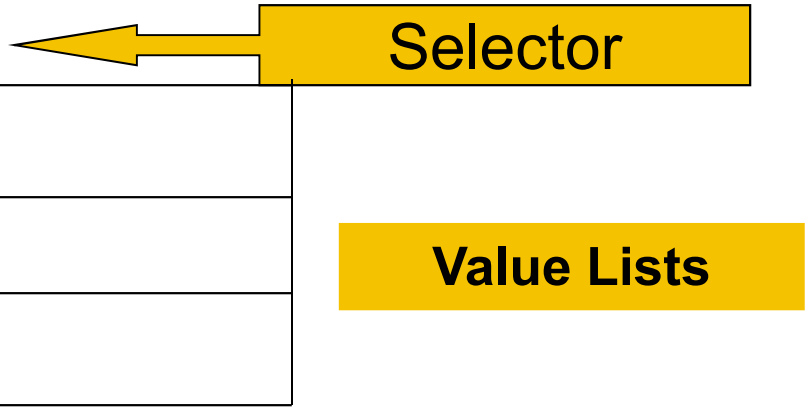
The program will take a course of action based on whether a condition is true.

```
If condition Then
   action1
ElseIf condition Then
   action2
Else
   action3
End If
```



Example 1: Code

```
Private Sub btnEvaluate_Click(...) _
                                    Handles btnEvaluate.Click
  Dim position As Integer = CInt(txtPosition.Text)
  Select Case position
    Case 1
      txtOutcome.Text = "Win"
    Case 2
      txtOutcome.Text = "Place"
    Case 3
      txtOutcome.Text = "Show"
    Case 4, 5
      txtOutcome.Text = "You almost placed in the money."
    Case Else
      txtBox.Text = "Out of the money."
  End Select
End Sub
```



CHAPTER 5

9

Chapter 5 - General Procedures

- 5.1 Sub Procedures, Part I
- 5.2 Sub Procedures, Part II
- 5.3 Function Procedures
- 5.4 Modular Design
- 5.5 A Case Study: Weekly Payroll

Devices for Modularity

- Visual Basic has two devices for breaking problems into smaller pieces:
  - Sub procedures
  - Function procedures

Sub Procedures

- Perform one or more related tasks
- General syntax

```
Sub ProcedureName()
   statements
End Sub
```

Calling a Sub Procedure

- The statement that invokes a Sub procedure is also referred to as a **Call statement**

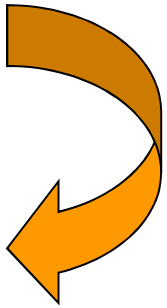
- A Call statement looks like this:

  ProcedureName()

- The rules for naming Sub procedures are the same as the rules for naming variables
  - Make them self-explanatory

EXAMPLE

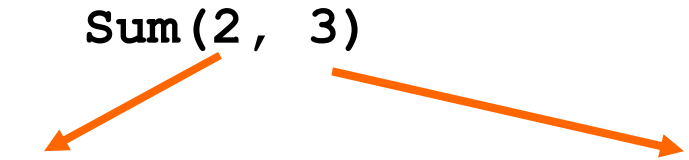
```
lstBox.Items.Clear()
ExplainPurpose()
lstBox.Items.Add("")
```



```
Sub ExplainPurpose()
  lstBox.Items.Add("Program displays a sentence")
  lstBox.Items.Add("identifying a sum.")
End Sub
```

PASSING VALUES

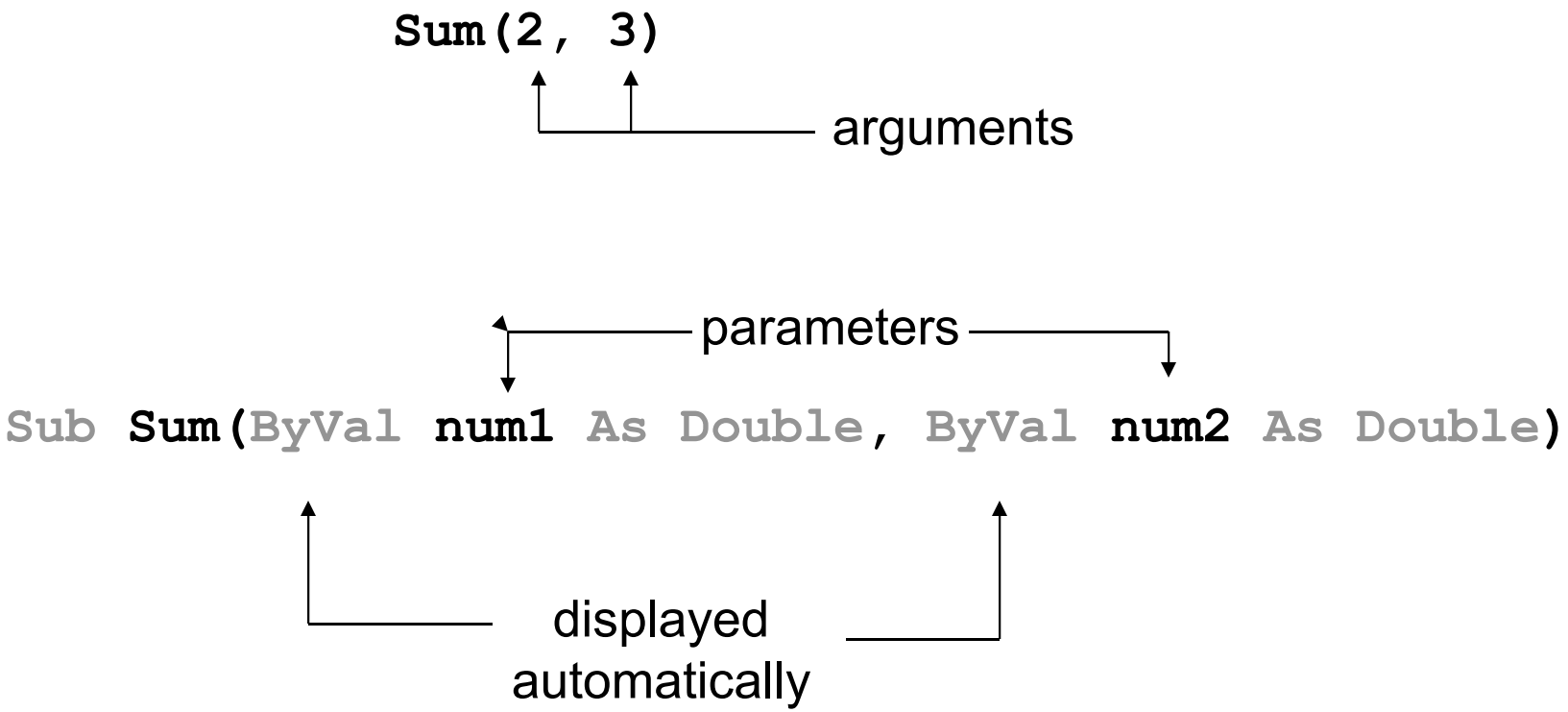
- You can send values to a Sub procedure




```
Sub Sum(ByVal num1 As Double, ByVal num2 As Double)
  lstBox.Items.Add("The sum of " & num1 & " and " _
            & num2 & " is " & (num1 + num2) & ".")
End Sub
```

- In the Sum Sub procedure, 2 will be stored in *num1* and 3 will be stored in *num2*

Arguments and Parameters



SFU

Several Calling Statements

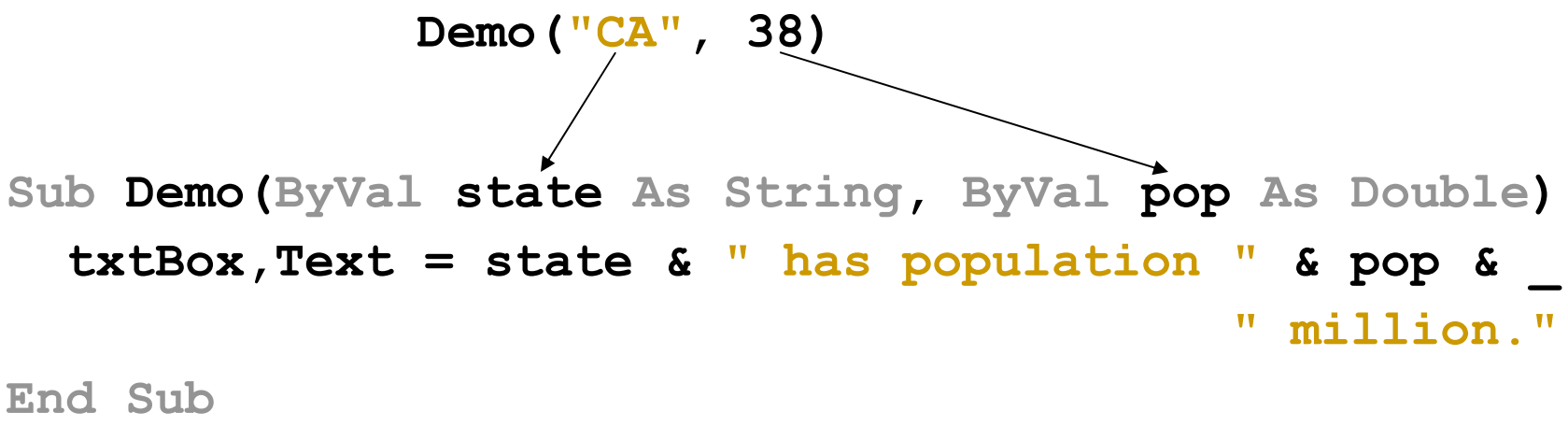
```
ExplainPurpose()
Sum(2, 3)
Sum(4, 6)
Sum(7, 8)
```

```
Sub Sum(ByVal num1 As Double, ByVal num2 As Double)
  lstBox.Items.Add("The sum of " & num1 & " and " _
            & num2 & " is " & (num1 + num2) & ".")
End Sub
```

Output:

```
Program displays a sentence identifying a sum.
```

Passing Strings and Numbers



- **Note**: The statement `Demo(38, "CA")` would not be valid. The types of the arguments must be in the same order as the types of the parameters.

Variables and Expressions

```
Dim s As String = "CA"
Dim p As Double = 19
Demo(s, 2 * p)

Sub Demo(ByVal state As String, ByVal pop As Double)
  txtBox.Text = state & " has population " & pop & _
                                        " million."
End Sub
```

- **Note**: The variable names in the arguments need not match the parameter names. For instance, *s* versus *state*..

Calling

A Sub procedure can call another Sub procedure.

```
Private Sub btnAdd_Click(...) Handles btnAdd.Click
  Sum(2, 3)
End Sub

Sub Sum(ByVal num1 As Double, ByVal num2 As Double)
  DisplayPurpose()
  lstBox.Items.Add("The sum of " & num1 & " and " _
            & num2 & " is " & (num1 + num2) & ".")
End Sub
```

"Any code of your own that you haven't looked at for six or more months might as well have been written by someone else."
(Eagleson's Law)

5.2 Sub Procedures, Part II

- Passing by Value
- Passing by Reference
- Lifetime and Scope of a Variable
- Debugging

ByVal and ByRef

- Parameters in Sub procedure headers are proceeded by ByVal or ByRef
- ByVal stands for *By Value*
- ByRef stands for *By Reference*

Passing by Value

- When a variable argument is passed to a ByVal parameter, just the value of the argument is passed.
- After the Sub procedure terminates, the variable has its original value.

EXAMPLE

```
Public Sub btnOne_Click (...) Handles _
                                    btnOne.Click
  Dim n As Double = 4
  Triple(n)
  txtBox.Text = CStr(n)
End Sub

Sub Triple(ByVal num As Double)
  num = 3 * num
End Sub
```

SAME EXAMPLE: N→NUM

```
Public Sub btnOne_Click (...) Handles _
                                    btnOne.Click
  Dim num As Double = 4
  Triple(num)
  txtBox.Text = CStr(num)
End Sub

Sub Triple(ByVal num As Double)
  num = 3 * num
End Sub
```

Passing by Reference

- When a variable argument is passed to a ByRef parameter, the parameter is given the same memory location as the argument

- After the Sub procedure terminates, the variable has the value of the parameter

EXAMPLE

```
Public Sub btnOne_Click (...) Handles _
                                    btnOne.Click
  Dim num As Double = 4
  Triple(num)
  txtBox.Text = CStr(num)
End Sub

Sub Triple(ByRef num As Double)
  num = 3 * num
End Sub
```

EXAMPLE: NUM ⟶ N

```
Private Sub btnOne_Click(...) Handles _
                                    btnOne_Click
  Dim n As Double = 4
  Triple(n)
  txtBox.Text = CStr(n)
End Sub

Sub Triple(ByRef num As Double)
  num = 3 * num
End Sub
```

Lifetime and Scope of a Variable

- Lifetime: Period during which it remains in memory
- Scope: In Sub procedures, defined same as in event procedures
- Suppose a variable is declared in procedure A that calls procedure B. While procedure B executes, the variable is alive, but out of scope

DEBUGGING

- Programs with Sub procedures are easier to debug
- Why?

"The first 90% of the code accounts for the first 90% of the development time. The remaining 10% of the code accounts for the other 90% of the development time."
(Tom Cargill)

5.3 Function Procedures

- User-Defined Functions Having Several Parameters
- User-Defined Functions Having No Parameters
- User-Defined Boolean-valued Functions
- Comparing Function Procedures with Sub Procedures
- Named Constants

Some Built-In Functions

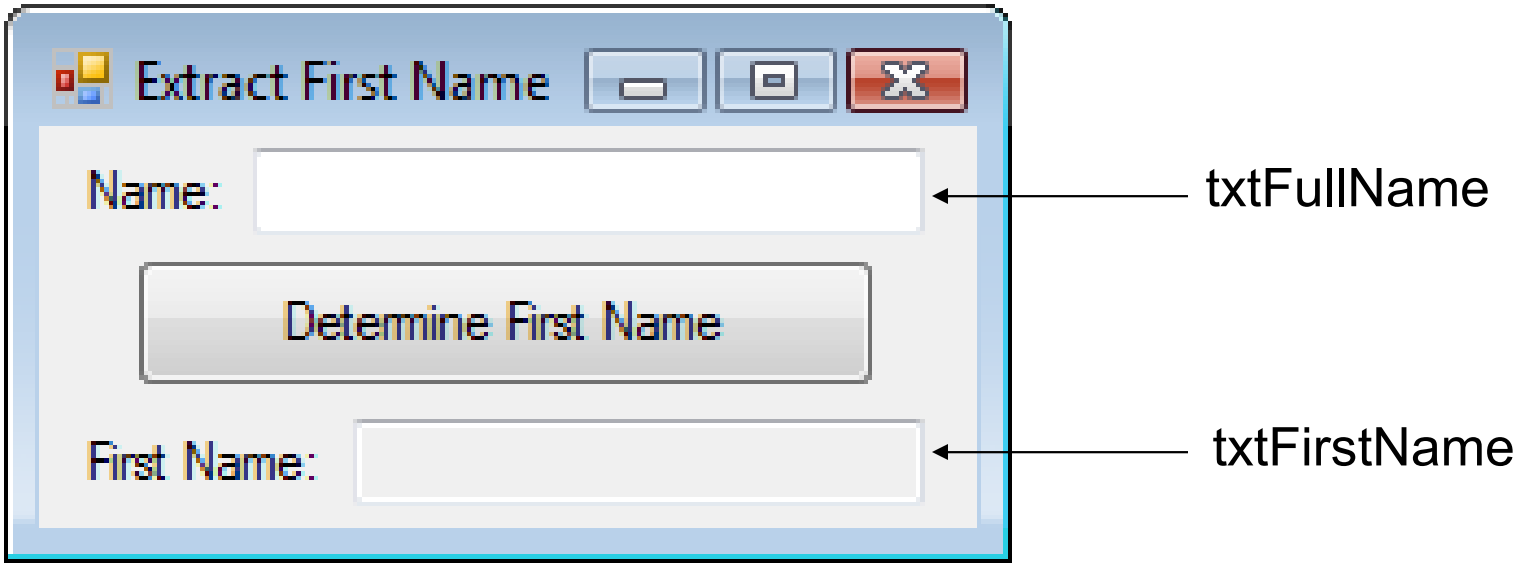
Function	Example	Input	Output
Int	Int(2.6) is 2	number	number
Math.Round	Math.Round(1.23,1) is 1.2	number, number	number
FormatPercent	FormatPercent(.12) is 12.00%	number	string
FormatNumber	FormatNumber(12345.628, 1) is 12,345.6	number, number	string

Function Procedures

- Function procedures (aka user-defined functions) always return one value
- Syntax:

```
Function FunctionName(ByVal var1 As Type1, _
                      ByVal var2 As Type2, _
                        …) As dataType
   statement(s)
   Return expression
End Function
```

EXAMPLE: FORM

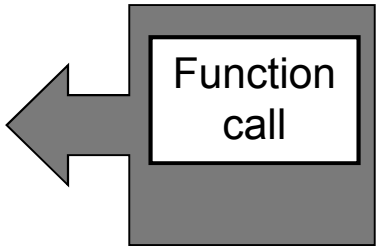


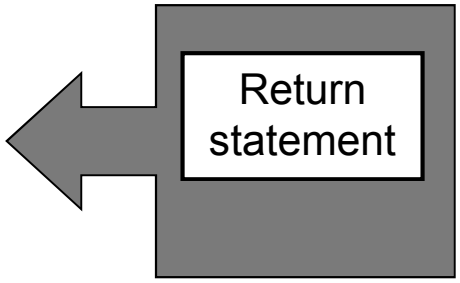
SFU

EXAMPLE: CODE

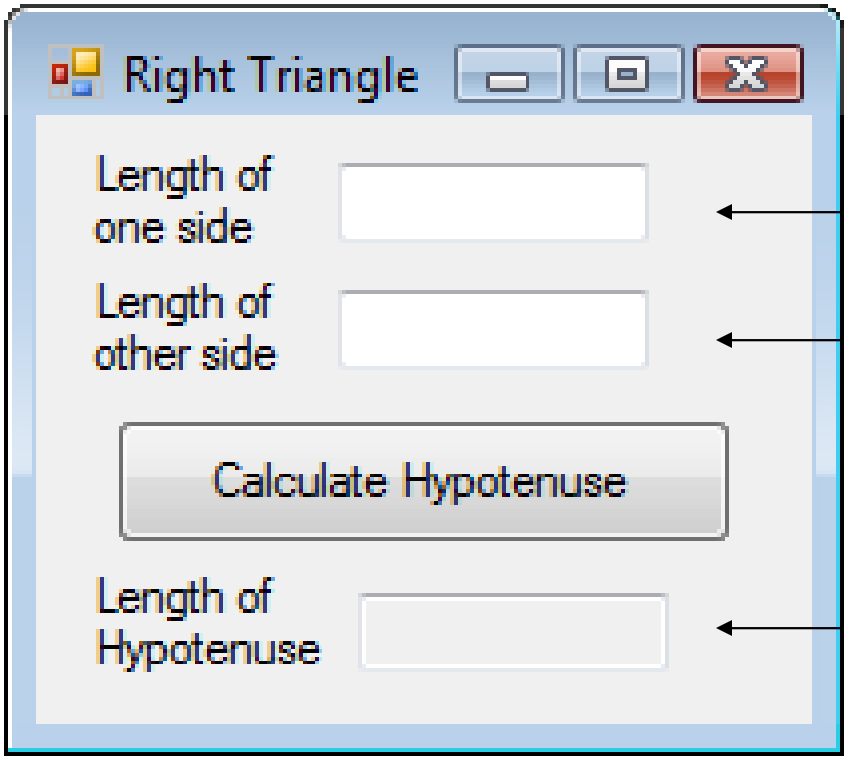
```
Private Sub btnDetermine_Click(...) _
                          Handles btnDetermine.Click
  Dim name As String
  name = txtFullName.Text
  txtFirstName.Text = FirstName(name)
End Sub

Function FirstName(ByVal name As String) As String
  Dim firstSpace As Integer
  firstSpace = name.IndexOf(" ")
  Return name.Substring(0, firstSpace)
End Function
```





EXAMPLE: FORM



SFU

EXAMPLE: CODE

```
Private Sub btnCalculate_Click(...) _
                              Handles btnCalculate.Click
  Dim a, b As Double
  a = CDbl(txtSideOne.Text)
  b = CDbl(txtSideTwo.Text)
  txtHyp.Text = CStr(Hypotenuse(a, b))
End Sub

Function Hypotenuse(ByVal a As Double, _
                    ByVal b As Double) As Double
  Return Math.Sqrt(a ^ 2 + b ^ 2)
End Function
```

Function Having No Parameters

```
Private Sub btnDisplay_Click(...) _
                Handles btnDisplay.Click
  txtBox.Text = Saying()
End Sub

Function Saying() As String
  Dim strVar As String
  strVar = InputBox("What is your" _
           & " favorite saying?")
  Return strVar
End Function
```

Comparing Function Procedures with Sub Procedures

- Subs are accessed using a Call statement
- Functions are called where you would expect to find a literal or expression
- For example:
  - result = *functionCall*
  - lstBox.Items.Add (*functionCall*)

Functions vs. Procedures

- Both can perform similar tasks
- Both can call other subs and functions
- Use a function when you want to return one and only one value

Named Constants

- Constant: value does not change during program execution (different from variable)

  (ex. Minimum wage, sales tax rate, name of a master file, mathematical constants, etc.)

Named Constants

```
Const CONSTANT_NAME As DataType = value
```

```
Const PI As Double = 3.14
Dim num As Double = 4
```

Named Constants

- Convention
  - Uppercase letters with words separated by underscore.
  - Place them in the Declaration sections.

Named Constants

```
Const INTEREST_RATE As Double = 0.04
Const MINIMUM_VOTING_AGE As Integer = 18
Const MASTER_FILE As String = 3.14

interstEarned =
  INTEREST_RATE * CDbl(txtAmount.Text)
If (age >= MINIMUM_VOTING_AGE) Then
  MessageBox.Show("You are eligible to vote")
End If
Dim sr As IO.StreamReader =
  IO.File.OpenText(MASTER_FILE)
```

“If debugging is the process of removing bugs, then programming must be the process of putting them in.”
(Edsger W. Dijkstra)

5.4 Modular Design

- Top-Down Design
- Structured Programming
- Advantages of Structured Programming

DESIGN TERMINOLOGY

- Large programs can be broken down into smaller problems
- "divide-and-conquer" approach called "stepwise refinement“
- Stepwise refinement is part of top-down design methodology

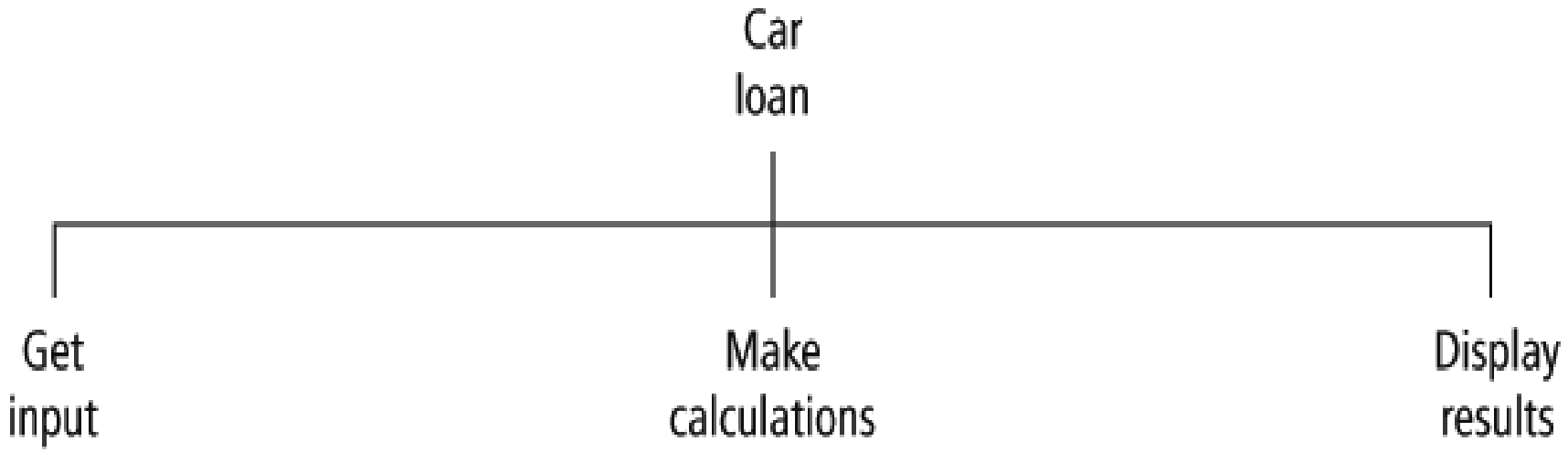
Top-Down Design

- General problems are at the top of the design
- Specific tasks are near the end of the design
- Top-down design and structured programming are techniques to enhance programmers' productivity

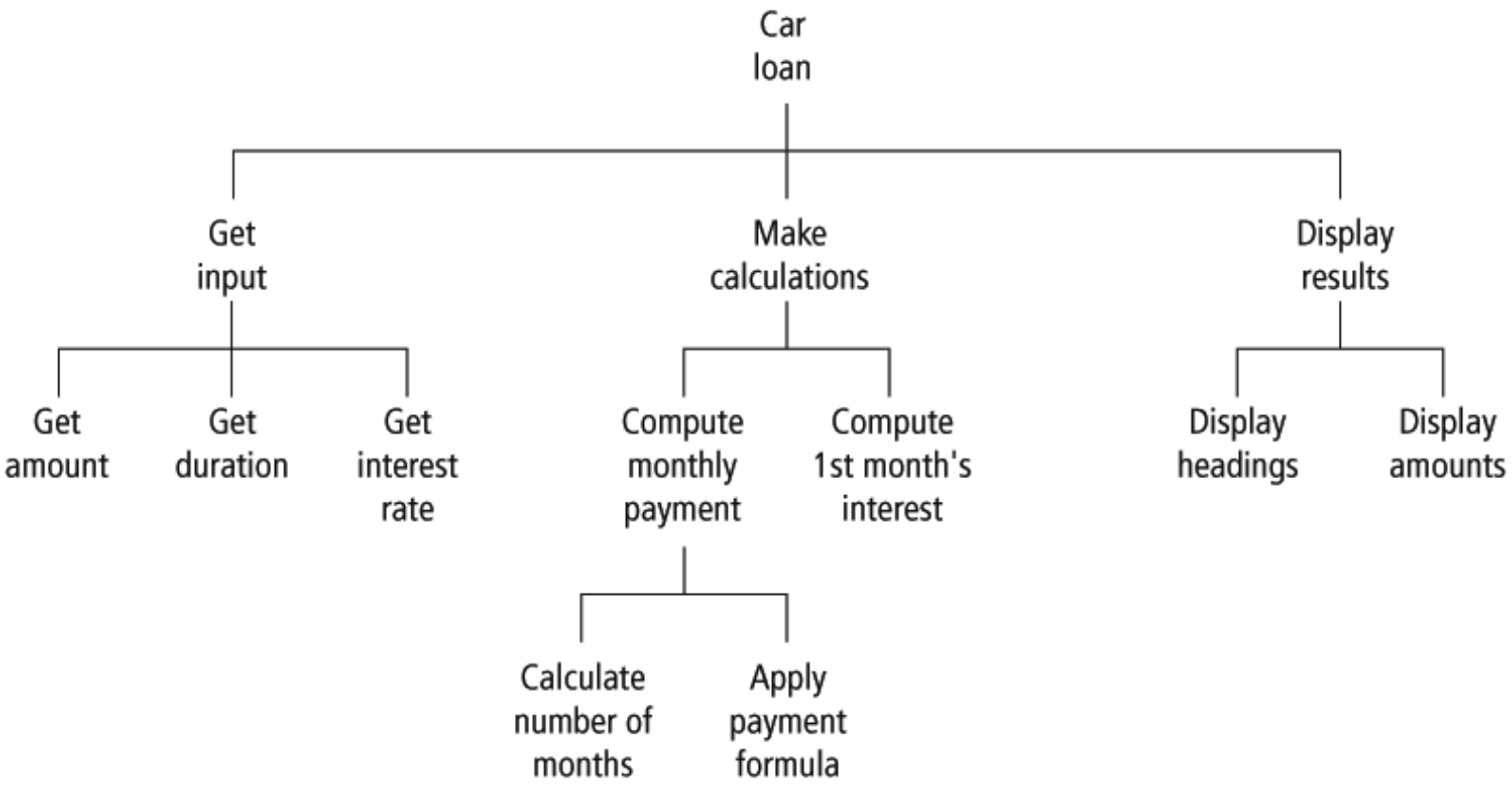
Top-Down Design Criteria

- The design should be easily readable and emphasize small module size
- Modules proceed from general to specific as you read down the chart
- The modules, as much as possible, should be single minded. That is, they should only perform a single well-defined task
- Modules should be as independent of each other as possible, and any relationships among modules should be specified

Top-Level Design Chart



Detailed Chart



SFU

Structured Programming

- Control structures in structured programming

  - ***Sequences*:** Statements are executed one after another

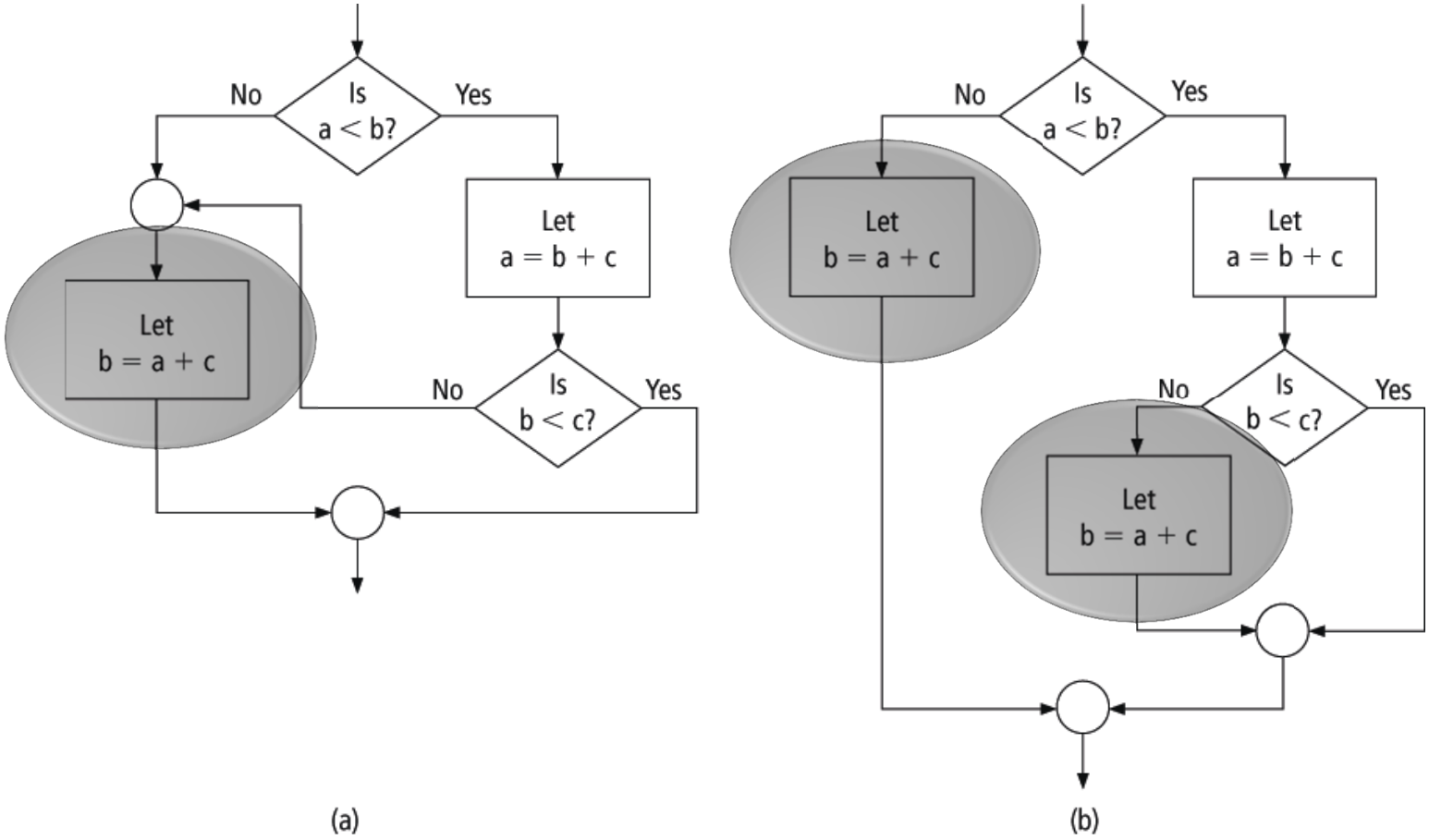
  - ***Decisions*:** One of two blocks of program code is executed based on a test for some condition

  - ***Loops (iteration)*:** One or more statements are executed repeatedly as long as a specified condition is true

Advantages of Structured Programming

- Goal to create correct programs that are easier to
  - write
  - understand
  - Modify
- "GOTO –less" programming

Comparison of Flow Charts



EASY TO WRITE

- Allows programmer to first focus on the big picture and take care of the details later
- Several programmers can work on the same program at the same time
- Code that can be used in many programs is said to be reusable

Easy to Debug

- Procedures can be checked individually
- A driver program can be set up to test modules individually before the complete program is ready
- Using a driver program to test modules (or stubs) is known as stub testing

EASY TO UNDERSTAND

- Interconnections of the procedures reveal the modular design of the program
- The meaningful procedure names, along with relevant comments, identify the tasks performed by the modules
- The meaningful variable names help the programmer to recall the purpose of each variable
- Because a structured program is self-documenting, it can easily be deciphered by another programmer

EXAMPLE

```
Private Sub btnDetermine_Click() Handles btnDetermine.Click
    Dim num As Integer
    num = 2
    MsgBox(Function1(num))
End Sub

Function Function1(ByVal num As Integer) As Integer
    Debug.Print(num)
    num = num ^ 2
    Function1 = Function2(num)
End Function

Function Function2(ByVal num As Integer) As Integer
    Debug.Print(num)
    num = num ^ 2
    Function2 = Function1(num)
End Function
```

What's the result?

Function Example

- How to calculate Factorials?
  - Ex: 5! = 5 * 4 * 3 * 2 * 1 = 120