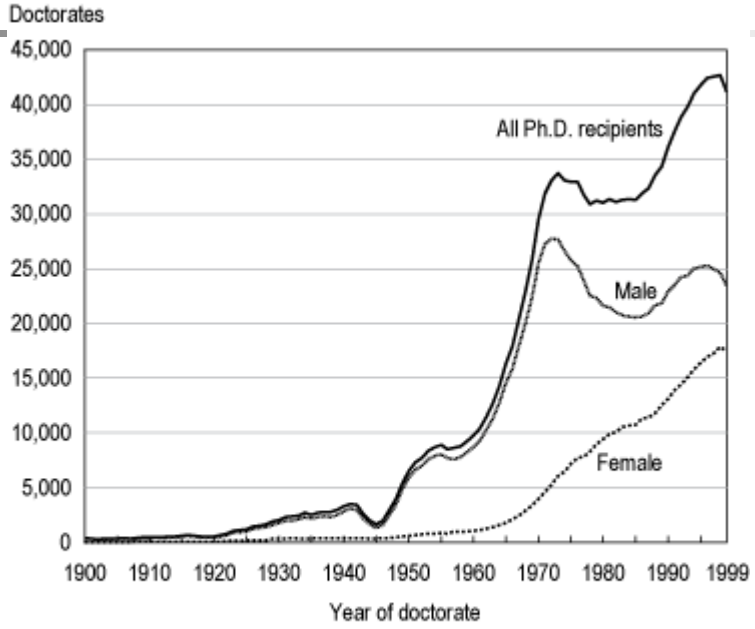
Visual Basic - Chapter 5

FIGURE 3-2. Doctorates awarded, by sex of recipient: 1900–99



SOURCES: NSF/NIH/USED/NEH/USDA/NASA, Survey of Earned Doctorates and Doctorate Records File (1920–99); U.S. Office of Education annual and biennial reports (1900–19).

Mohammad Shokoohi

* Adopted from An Introduction to Programming Using Visual Basic 2010, Schneider

Chapter 5 - General Procedures

5.1 Function Procedures

5.2 Sub Procedures, Part I

5.3 Sub Procedures, Part II

5.4 Modular Design

Devices for Modularity

Visual Basic has two devices for breaking problems into smaller pieces:

- Function procedures
- Sub procedures

5.1 Function Procedures

- User-Defined Functions Having One Parameter
- User-Defined Functions Having Several Parameters
- User-Defined Functions Having No Parameters
- User-Defined Boolean-Valued Functions

Some Built-In Functions

Function: Int
Example: Int(2.6) is 2
Input: number
Output: number

Function: Math.Round
Example: Math.Round(1.23, 1) is 1.2
Input: number, number
Output: number

Some Built-In Functions (continued)

Function: FormatPercent
Example: FormatPercent(0.12, 2) is 12.00%
Input: number, number
Output: string

Function: FormatNumber
Example: FormatNumber(12.62, 1) is 12.6
Input: number, number
Output: string

Function Procedures

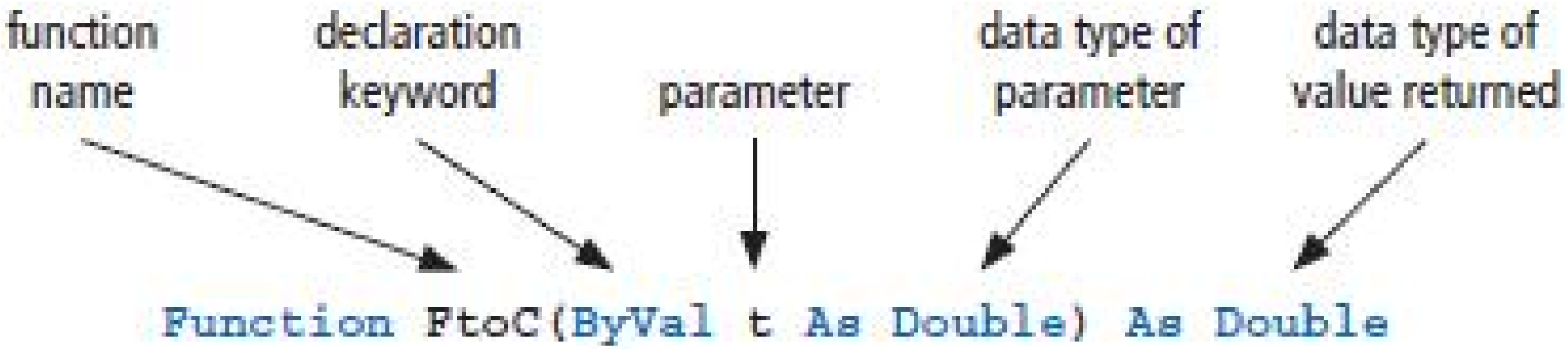
- Function procedures (aka user-defined functions) always return one value
- Syntax:

```
Function FunctionName(ByVal var1 As Type1,
                      ByVal var2 As Type2,
                      ...) As ReturnDataType
  statement(s)
  Return expression
End Function
```

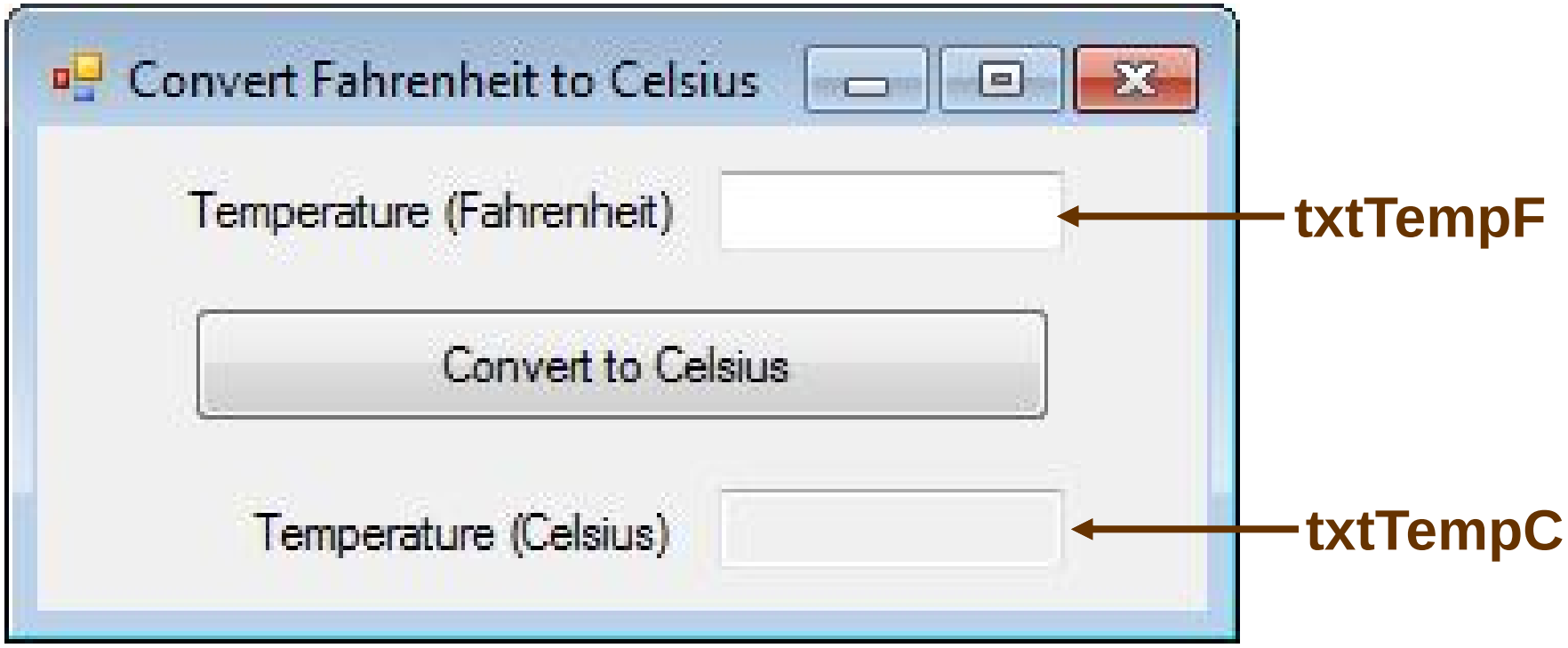
Example With One Parameter

```
Function FtoC(ByVal t As Double) As Double
   'Convert Fahrenheit temp to Celsius
   Return (5 / 9) * (t - 32)
End Function
```

Header of the FtoC Function Procedure



Example 1: Form

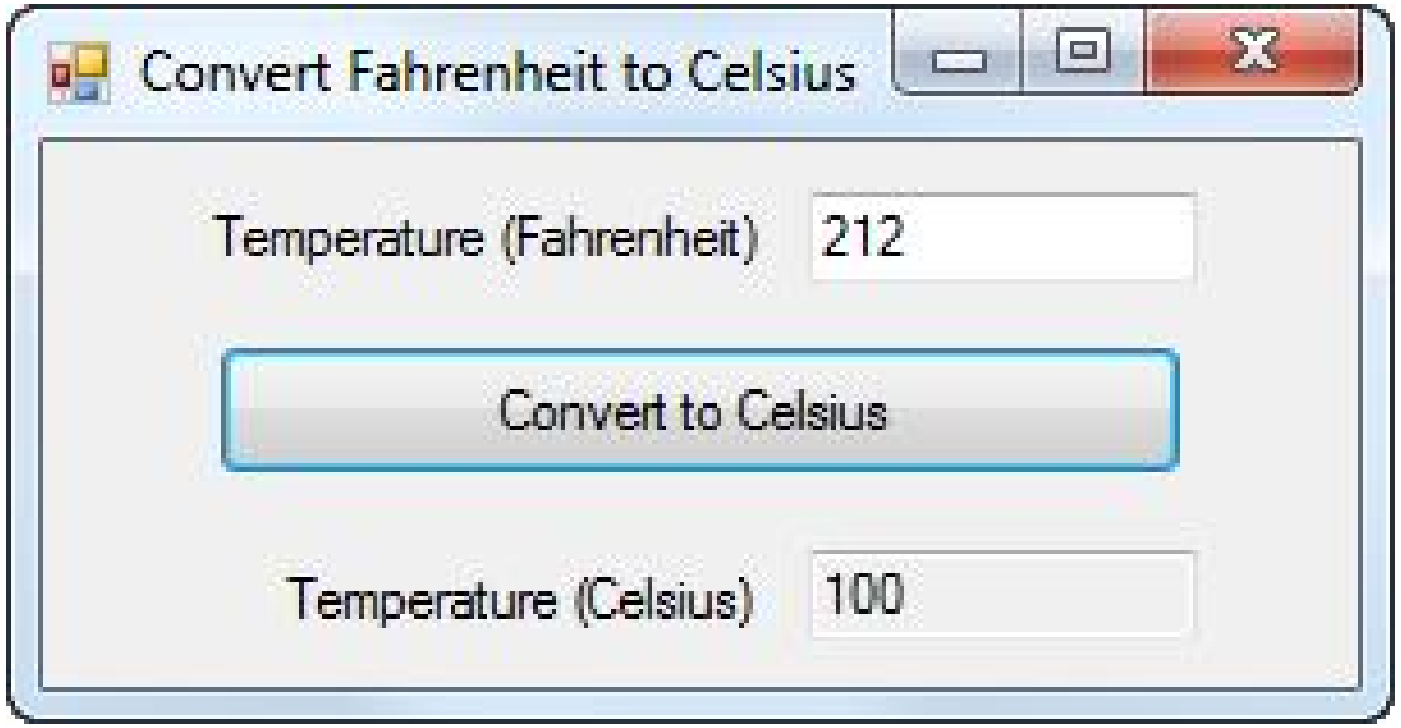


Example 1: Code

```
Private Sub btnConvert_Click(...) _
                        Handles btnConvert.Click
  Dim fahrenheitTemp, celsiusTemp As Double
  fahrenheitTemp = CDbl(txtTempF.Text)
  celsiusTemp = FtoC(fahrenheitTemp)
  txtTempC.Text = CStr(celsiusTemp )
End Sub

Function FtoC(ByVal t As Double) As Double
  Return (5 / 9) * (t - 32)
End Function
```

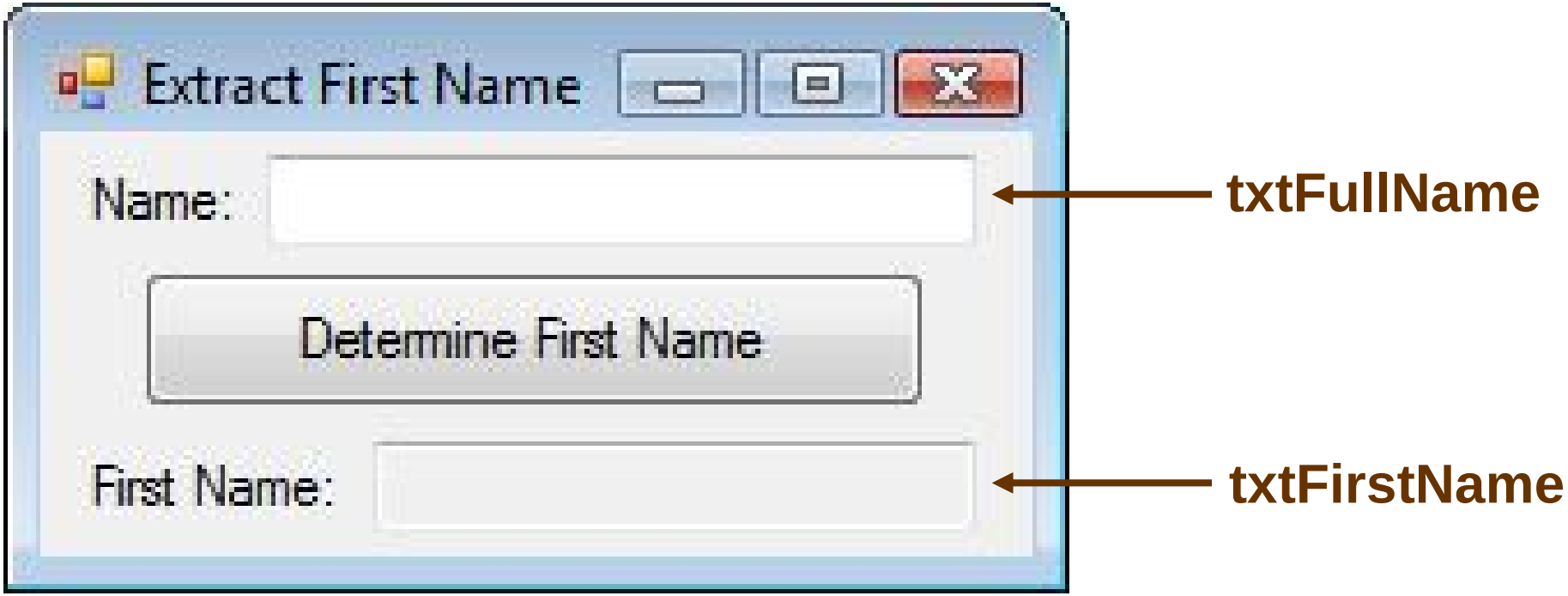
Example 1: Output



Example With One Parameter

```
Function FirstName(ByVal fullName As
                               String) As String
  'Extract first name from full name
  Dim firstSpace As Integer
  firstSpace = fullName.IndexOf(" ")
  Return fullName.Substring(0, firstSpace)
End Function
```

Example 2: Form

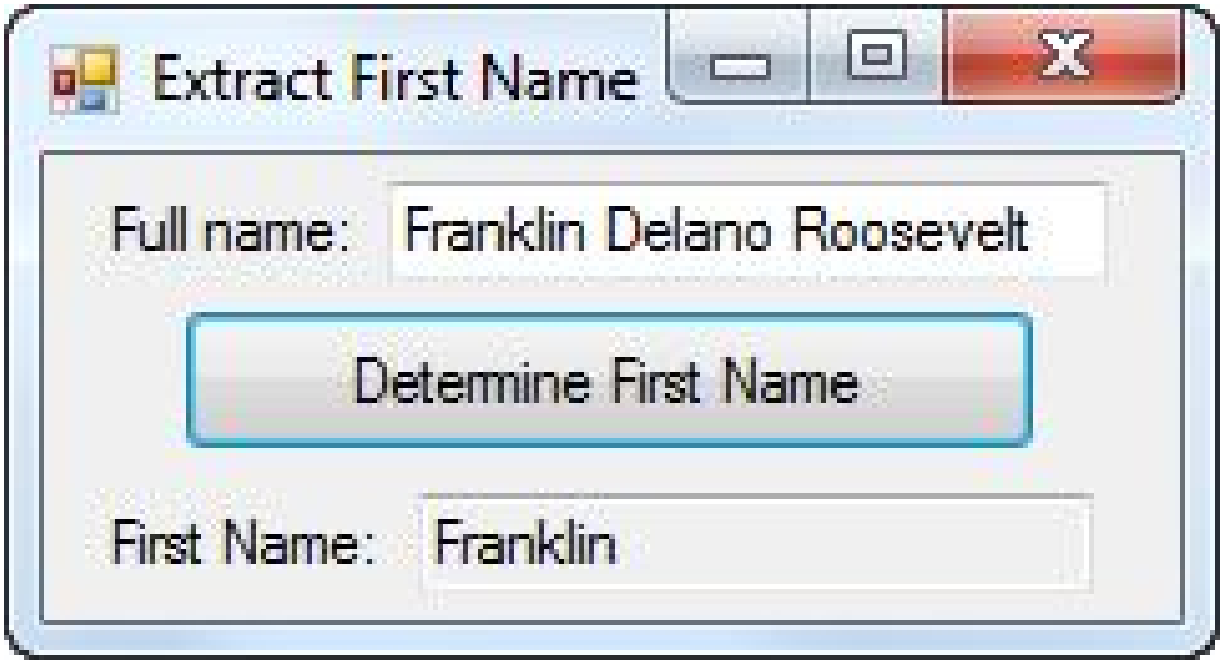


Example 2: Code

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

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

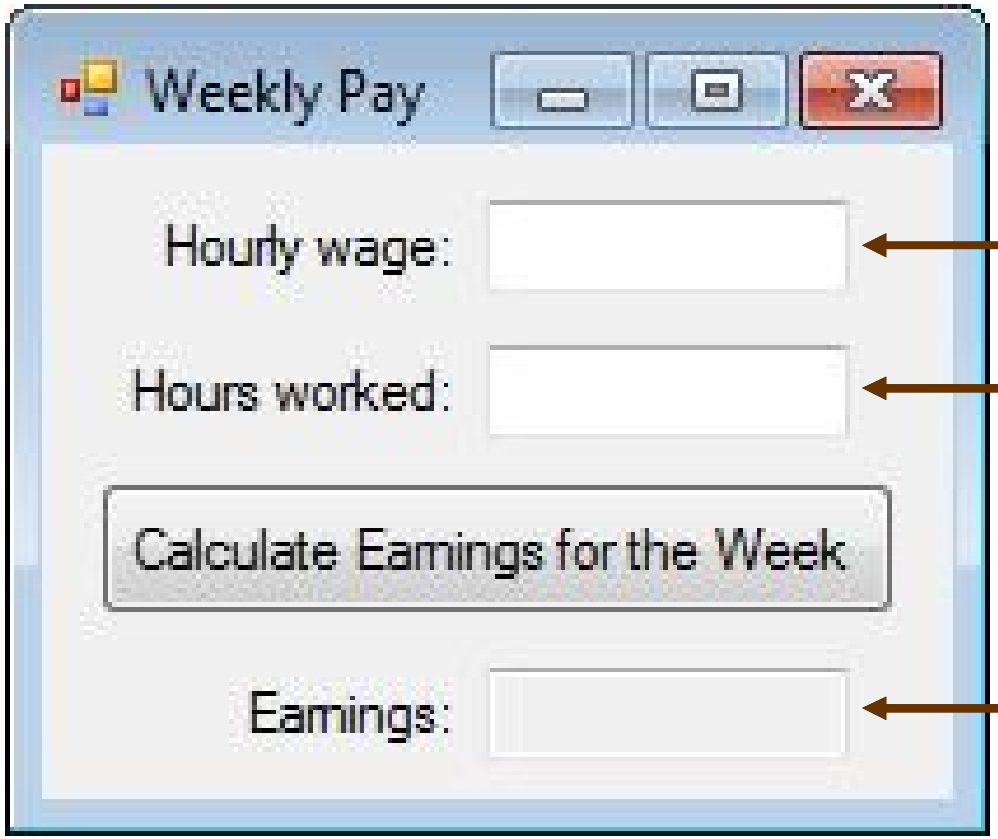
Example 2: Output



User-Defined Function Having Several Parameters

```
Function Pay(ByVal wage As Double,
             ByVal hrs As Double) As Double
  Dim amt As Double  'amount of salary
  Select Case hrs
    Case Is <= 40
      amt = wage * hrs
    Case Is > 40
      amt = wage * 40 +(1.5 * wage * (hrs – 40))
  End Select
  Return amt
End Function
```

Example 3: Form

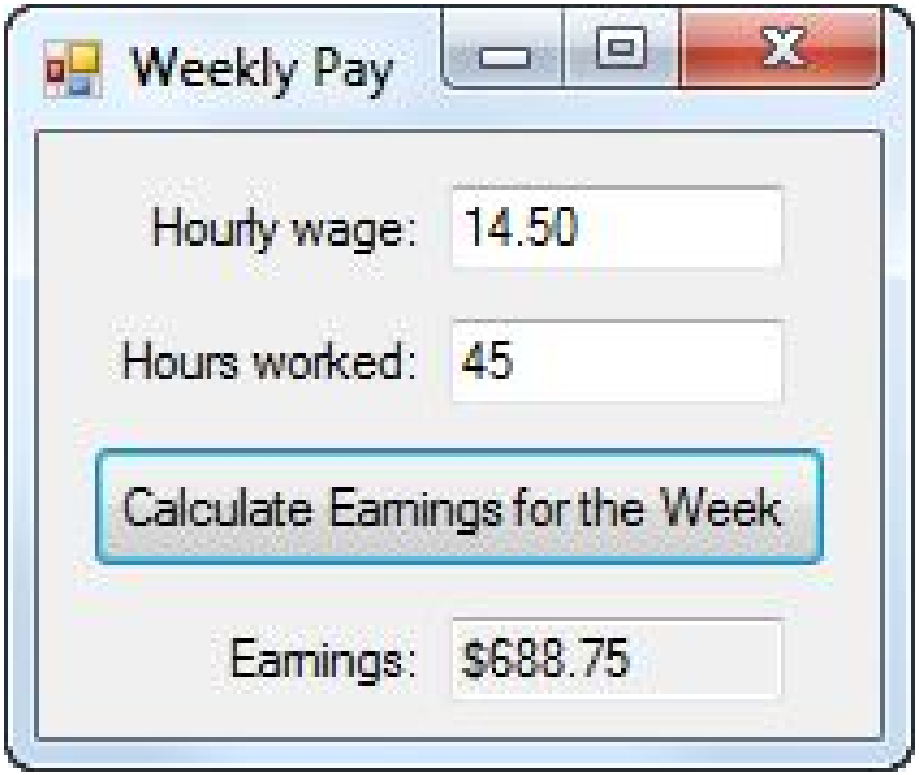


Example 3: Partial Code

```
Private Sub btnCalculate_Click(...) _
                          Handles btnCalculate.Click
  Dim hourlyWage, hoursWorkded As Double
  hourlyWage = CDbl(txtWage.Text)
  hoursWorked = CDbl(txtHours.Text)
  txtEarnings.Text =
     FormatCurrency(Pay(hourlyWage, hoursWorked))
End Sub
```

Function call

Example 3: Output



User-Defined Function Having No Parameters

```
Function CostOfItem() As Double
  Dim price As Double = CDbl(txtPrice.Text)
  Dim quantity As Integer =
                          CDbl(txtQuantity.Text)
  Dim cost = price * quantity
  Return cost
End Function
```

User-Defined Boolean-Valued Function

```
Function IsVowelWord(ByVal word As String) As
                                          Boolean
  If word.IndexOf("A") = -1 Then
    Return False
  End If
  .
  .
  If word.IndexOf("U") = -1 Then
    Return False
  End If
  Return True
End Function
```

5.2 Sub Procedures, Part I

- Defining and Calling Sub Procedures
- Variables and Expressions as Arguments
- Sub Procedures Calling Other Sub Procedures

General Form of Sub Procedure

```
Sub ProcedureName(ByVal par1 As Type1,
                  ByVal par2 As Type2,
                     :
                  ByVal parN As TypeN)
   statement(s)
End Sub
```

Calling a Sub Procedure

- The statement that invokes a Sub procedure is referred to as a **calling statement**.
- A calling statement looks like this:

```
ProcedureName(arg1, arg2,..., argN)
```

Naming Sub Procedures

The rules for naming Sub procedures are the same as the rules for naming variables.

Passing Values

```
                        DisplaySum(2, 3)
Sub DisplaySum(ByVal num1 As Double, ByVal num2 _
                                          As Double)
  Dim z As Double
  z = num1 + num2
  lstOutput.Items.Add("The sum of " & num1 &
                "  " and " & num2 & " is " & z & ".")
End Sub
```

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

Arguments and Parameters

```
Sum(2, 3)
```

arguments

parameters

```
Sub DisplaySum(ByVal num1 As Double, ByVal num2 _
                                        As Double)
```

displayed automatically

Several Calling Statements

```
DisplaySum(2, 3)
DisplaySum(4, 6)
DisplaySum(7, 8)
```

Output:

```
The sum of 2 and 3 is 5.
The sum of 4 and 6 is 10
The sum of 7 and 8 is 15.
```

Passing Strings and Numbers

```
                    Demo("CA", 38)
Sub Demo(ByVal state As String, ByVal pop _
                                    As Double)
  lstOutput.Items.Add = state &
          " has population " & pop & " million."
End Sub
```

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 as Arguments

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

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

Note: The argument names need not match the parameter names. For instance, *s* versus *state*.

Sub Procedure Having No Parameters

```
Sub DescribeTask()
   lstBox.Items.Clear()
   lstBox.Items.Add("This program displays")
   lstBox.Items.Add("the name and population")
   lstBox.Items.Add("of a state.")
End Sub
```

Sub Procedure Calling Another Sub Procedure

```
Private Sub btnDisplay_Click(...) Handles _
                                    btnDisplay.Click
  Demo("CA", 37)
End Sub

Sub Demo(ByVal state As String, ByVal pop _
                                    As Double)
  DescribeTask()
  lstOutput.Items.Add("")
  lstOutput.Items.Add = state &
       " has population " & pop &  " million."
End Sub
```

Output

```
This program displays
the name and population
of a state.

CA has population 37 million.
```

5.3 Sub Procedures, Part II

- Passing by Value
- Passing by Reference
- Sub Procedures that Return a Single Value
- Lifetime and Scope of Variables and Constants
- 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
```

Output: `4`

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
```

Output: `4`

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
```

Output: **12**

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
```

Output: `12`

Most Common Use of ByRef: Get Input

```
Sub InputData(ByRef wage As Double,
              ByRef hrs As Double)
  wage = CDbl(txtWage.Text)
  hrs = CDbl(txtHours.Text)
End Sub
```

Sub Procedures that Return a Single Value with ByRef

- Should be avoided
- Usually can be replaced with a Function procedure

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
- Each Sub procedure can be checked individually before being placed into the program

Comparing Function Procedures with Sub Procedures

- Sub procedures are accessed using a calling 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 procedures
- Use a function when you want to return one and only one value

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

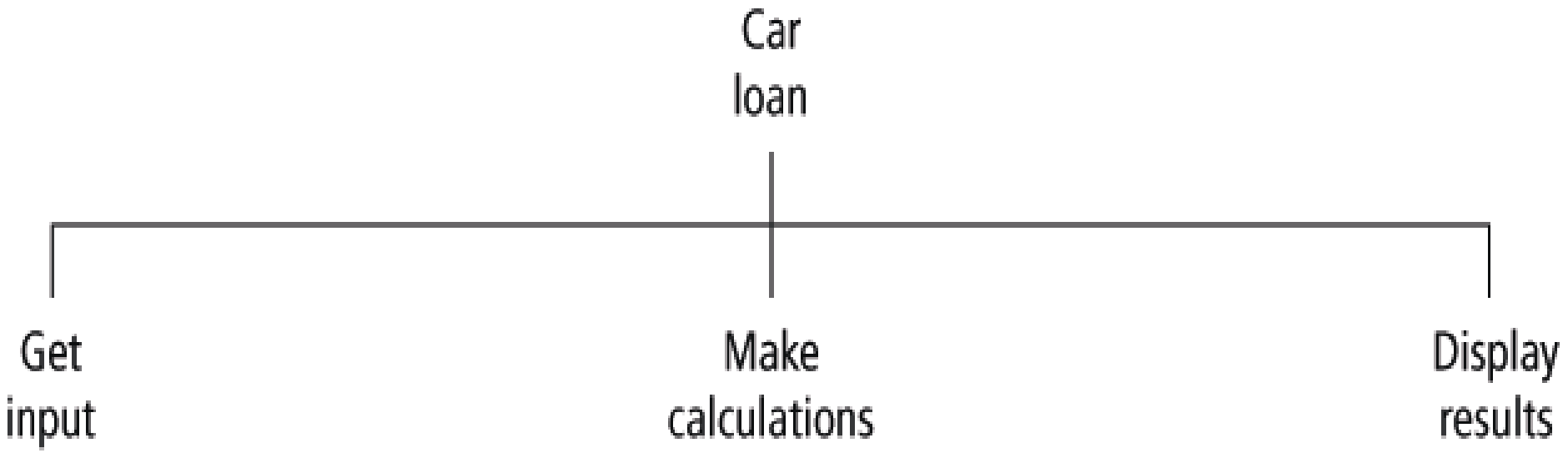
Top-Down Design

- 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

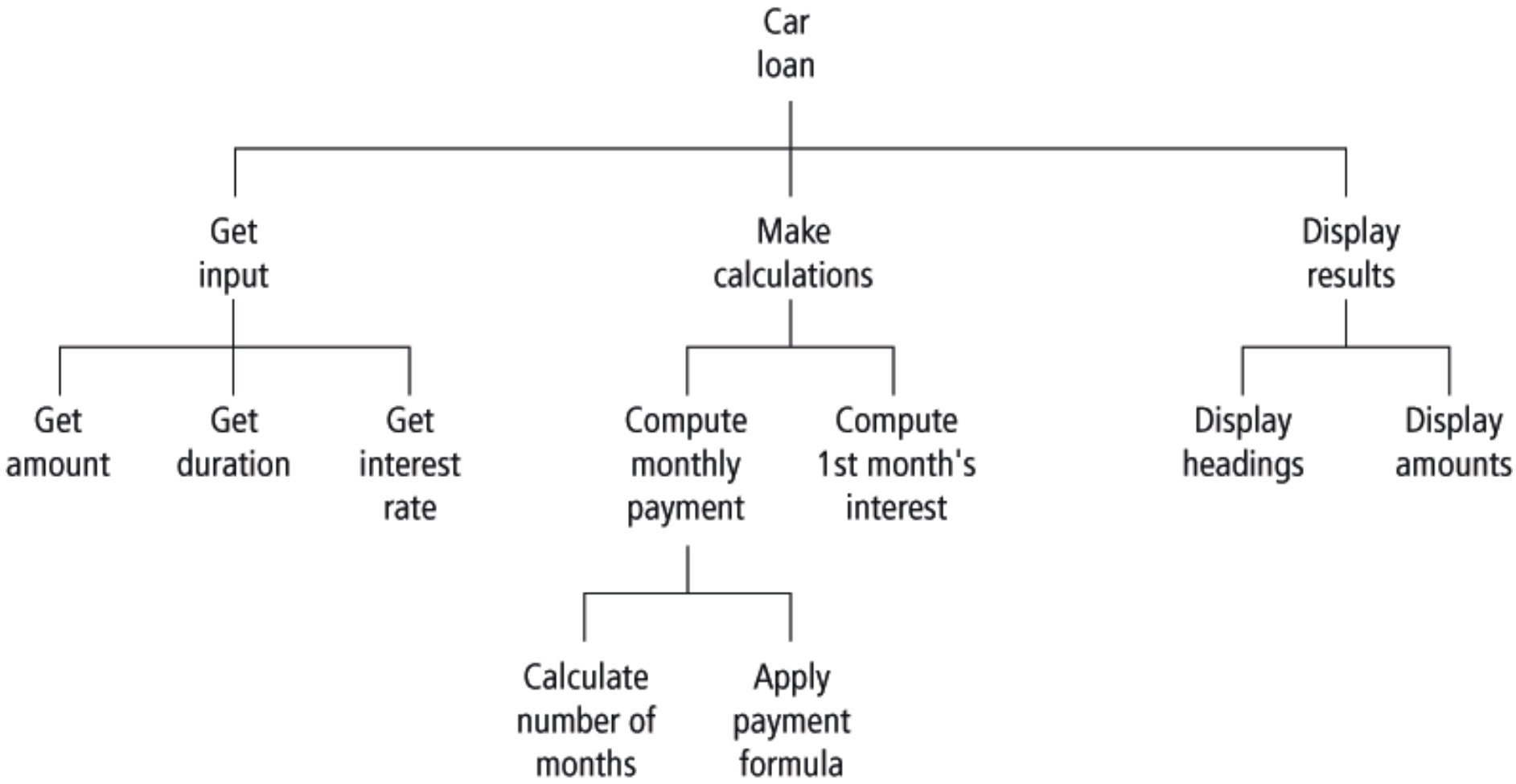
Top-Down Design Criteria

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

Beginning of Hierarchy Chart



Detailed Hierarchy Chart



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 of a 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

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 procedure names help the programmer recall the purpose of each procedure.

Easy to Change

- Because a structured program is **self-documenting**, it can easily be deciphered by another programmer.

Object-Oriented Programming

- an encapsulation of data and code that operates on the data
- objects have properties, respond to methods, and raise events.