

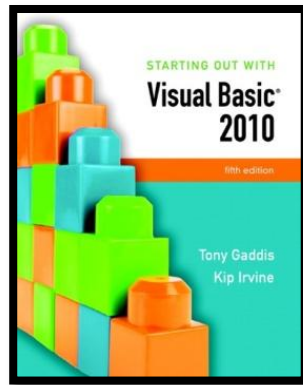


STARTING OUT WITH

# Visual Basic® 2010

fifth edition

Tony Gaddis  
Kip Irvine



# Chapter 6

## Procedures and Functions

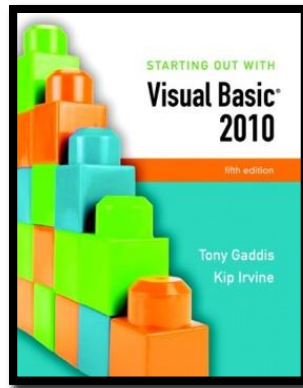
**Addison Wesley**  
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

# Introduction

- A **procedure** is a collection of statements that performs a task
  - Event handlers are a type of procedure
- A **function** is a collection of statements that performs a task and returns a value to the part of the program that executed it
  - You have already worked with Visual Basic's built-in functions, such as **CInt** and **IsNumeric**
- A **method** can be either a procedure or a function



## Section 6.1

# PROCEDURES

You can write your own general purpose procedures that perform specific tasks. General purpose procedures are not triggered by events, but are called from statements in other procedures.

**Addison Wesley**  
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

# Procedure Uses

- An event handler is a type of procedure
  - Automatically executed when an event such as a mouse click occurs
- General purpose procedures are triggered by statements in other procedures, not by events
- Procedures help simplify & **modularize** code by:
  - Breaking it into small, manageable pieces
  - Performing a task that is needed repeatedly
  - Dividing a program into a set of logical tasks
- Tutorial 6-1 examines an application with a procedure

# Declaring a Procedure

- The general format of a **procedure declaration** is as follows:

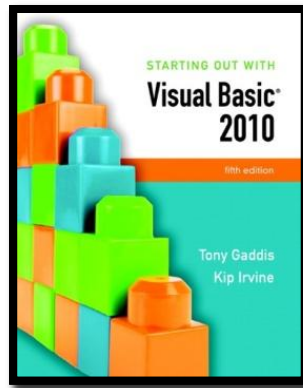
***[AccessSpecifier] Sub ProcedureName ([ParameterList])  
[Statements]***

**End Sub**

- ***AccessSpecifier*** is optional and establishes accessibility to the program
- **Sub** and **End** are keywords
- ***ProcedureName*** used to refer to procedure
  - Use **Pascal casing** to capitalize 1st character of the name and each new word in the name
- ***ParameterList*** is a list of variables or values being passed to the sub procedure
- Tutorial 6-2 guides you through the process of writing procedures

# Static Local Variables

- Variables needed only in a procedure, should be declared within that procedure
  - Creates a local variable with scope only within the procedure where declared
  - Local variable values are not saved from one procedure call to the next
- To save value between procedure calls, use **Static** keyword to create a **static local variable**
  - Static VariableName As DataType**
    - Scope is still only within the procedure
    - But variable exists for lifetime of application



## Section 6.2

# PASSING ARGUMENTS TO PROCEDURES

When calling a procedure, you can pass it values known as arguments.

**Addison Wesley**  
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.



# Arguments

- An **Argument** is value passed to a procedure
- For example:

**CInt(txtInput.Text)**

- Calls the **CInt** function
- Passes **txtInput.Text** as an argument
- Two ways to pass arguments
  - **by value** is a temporary copy of the original argument
  - **by reference** is the original argument and can be changed

# Passing Arguments By Value

```
DisplayValue(5)           ' Call DisplayValue procedure
    |
    |_____↓
Sub DisplayValue(ByVal intNumber As Integer)
    ' This procedure displays a value in a message box.
    MessageBox.Show(intNumber.ToString)
End Sub
```

- **intNumber** declared as an integer argument
- Storage location **intNumber** created by procedure
- A value, **5** in this case, must be supplied and is copied into the storage location for **intNumber**
- The **DisplayValue** procedure then executes
- Tutorial 6-3 demonstrates passing arguments

# Passing Multiple Arguments

`ShowSum(intValue1, intValue2)`      ' Call ShowSum procedure

```
Sub ShowSum(ByVal intNum1 As Integer, ByVal intNum2 As Integer)
    Dim intSum As Integer 'Local variable to hold a sum
    'Get the sum of the two arguments.
    intSum = intNum1 + intNum2
    'Display the sum.
    MessageBox.Show("The sum is " & intSum.ToString())
End Sub
```

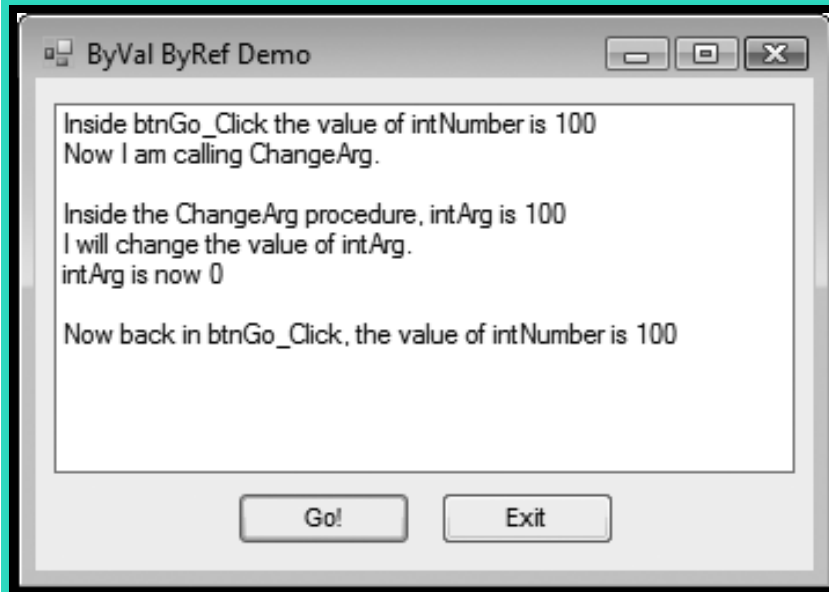
- Multiple arguments separated by commas
- Value of first argument is copied to first
- Second to second, etc.

# More about Passing Arguments by Reference

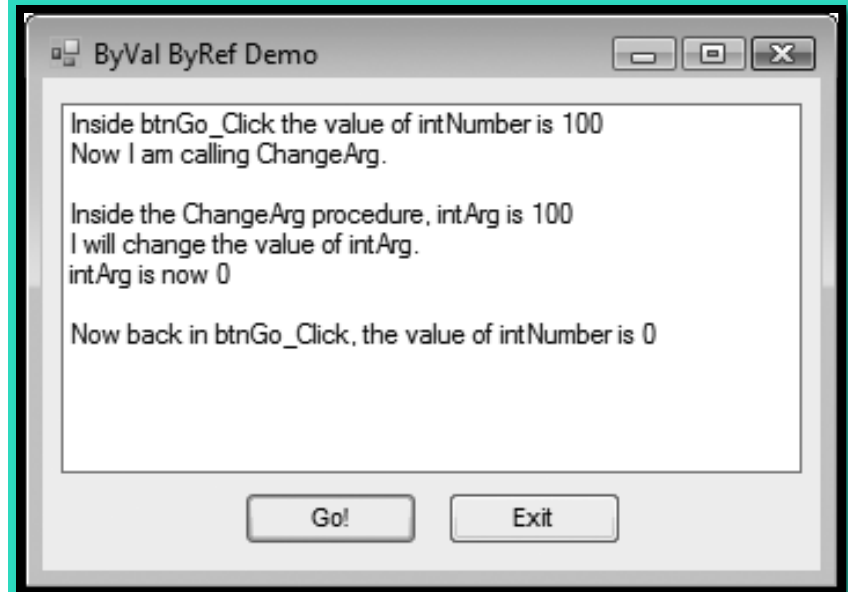
- Arguments are usually passed **ByVal**
  - New storage location created for procedure
  - Storage location gets a copy of the value
  - Any changes in value are made to the copy
  - Calling procedure won't "see" the changes
- Arguments can also be passed **ByRef**
  - Procedure points to (references) argument's original storage location
  - Any changes are made to the original value
  - Calling procedure "sees" the changes
- Tutorial 6-4 demonstrates the difference between **ByVal** and **ByRef**

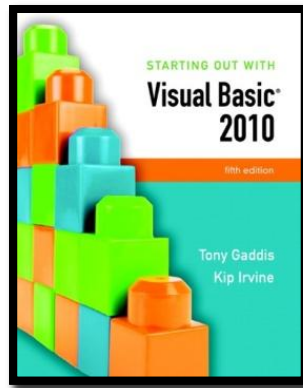
# Working with ByVal and ByRef

- Passing the argument **ByVal**
  - Does not change the value of **intNumber**



- Passing the argument **ByRef**
  - Allows the value of **intNumber** to change





## Section 6.3

# FUNCTIONS

A function returns a value to the part of the program that called the function.

**Addison Wesley**  
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

# Declaring a Function

```
[AccessSpecifier] Function FunctionName ([ParameterList]) As DataType  
    [Statements]  
End Function
```

- New keyword **Function**
- Also new is **As DataType** which states the data type of the value to be returned
- Return value is specified in a **Return** expression

# Function Call Example

```
dblTotal = Sum(dblValue1, dblValue2)
Function Sum(ByVal dblNum1 As Double, ByVal dblNum2 As Double) As Double
Return dblNum1 + dblNum2
End Function
```

- The **Sum** function
  - Passes the variables **dblValue1** and **dblValue2** as arguments
  - Data types must agree with parameter list
  - Assigns the value returned by the **Sum** function to the variable **dblTotal**, agrees with return value
- Tutorial 6-5 demonstrates function use



# Returning Nonnumeric Values

- Functions can return nonnumeric values, such as strings and Boolean values

```
strCustomer = FullName("John", "Martin")
```

```
Function FullName(ByVal strFirst As String,  
                  ByVal strLast As String) As String
```

```
' Local variable to hold the full name
```

```
Dim strName As String
```

```
' Append the last name to the first
```

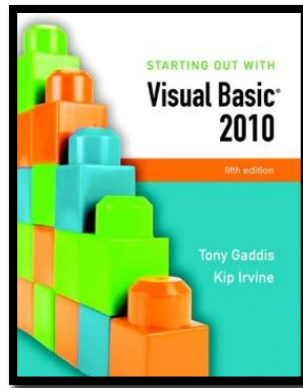
```
' name and assign the result to strName.
```

```
strName = strFirst & " " & strLast
```

```
' Return the full name.
```

```
Return strName
```

```
End Function
```



## Section 6.4

# MORE ABOUT DEBUGGING: STEPPING INTO, OVER, AND OUT OF PROCEDURES AND FUNCTIONS

Visual Basic debugging commands allow you to single-step through applications with procedure and function calls. The *Step Into* command allows you to single-step through a called procedure or function. The *Step Over* command allows you to execute a procedure or function call without single-stepping through its lines. The *Step Out* command allows you to execute all remaining lines of a procedure or function you are debugging without stepping through them.

**Addison Wesley**  
is an imprint of



© 2011 Pearson Addison-Wesley. All rights reserved.

# The Step Into Command

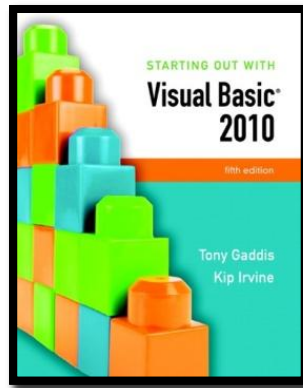
- The **Step Into** command
  - Continue to debug by single-stepping through a procedure
    - Press the F8 key
    - Select *Debug* from the menu bar, and then select *Step Into* from the *Debug* menu
    - Click the *Step Into* button on the *Debug Toolbar*, if the toolbar is visible
- Tutorial 6-6 demonstrates the **Step Into** command

# The Step Over Command

- The **Step Over** command
  - Run procedure without single-stepping, continue single-step after the call
    - Press the Shift + F8 key
    - Select *Debug* from the menu bar, and then select *Step Over* from the *Debug* menu
    - Click the *Step Over* button on the *Debug Toolbar*, if the toolbar is visible
- Tutorial 6-7 demonstrates the **Step Over** command

# The Step Out Command

- The **Step Out** command
  - End single-stepping in procedure, continue single-step after the call
    - Press the Ctrl + Shift + F8 key
    - Select *Debug* from the menu bar, and then select *Step Out* from the *Debug* menu
    - Click the *Step Out* button on the *Debug Toolbar*, if the toolbar is visible
- Tutorial 6-8 demonstrates the **Step Out** command



## Section 6.5

# FOCUS ON PROGRAM DESIGN AND PROBLEM SOLVING: BUILDING THE *BAGEL AND COFFEE PRICE CALCULATOR* APPLICATION

In this section you build the *Bagel and Coffee Price Calculator* application. It uses procedures and functions to calculate the total of a customer order.

**Addison Wesley**  
is an imprint of



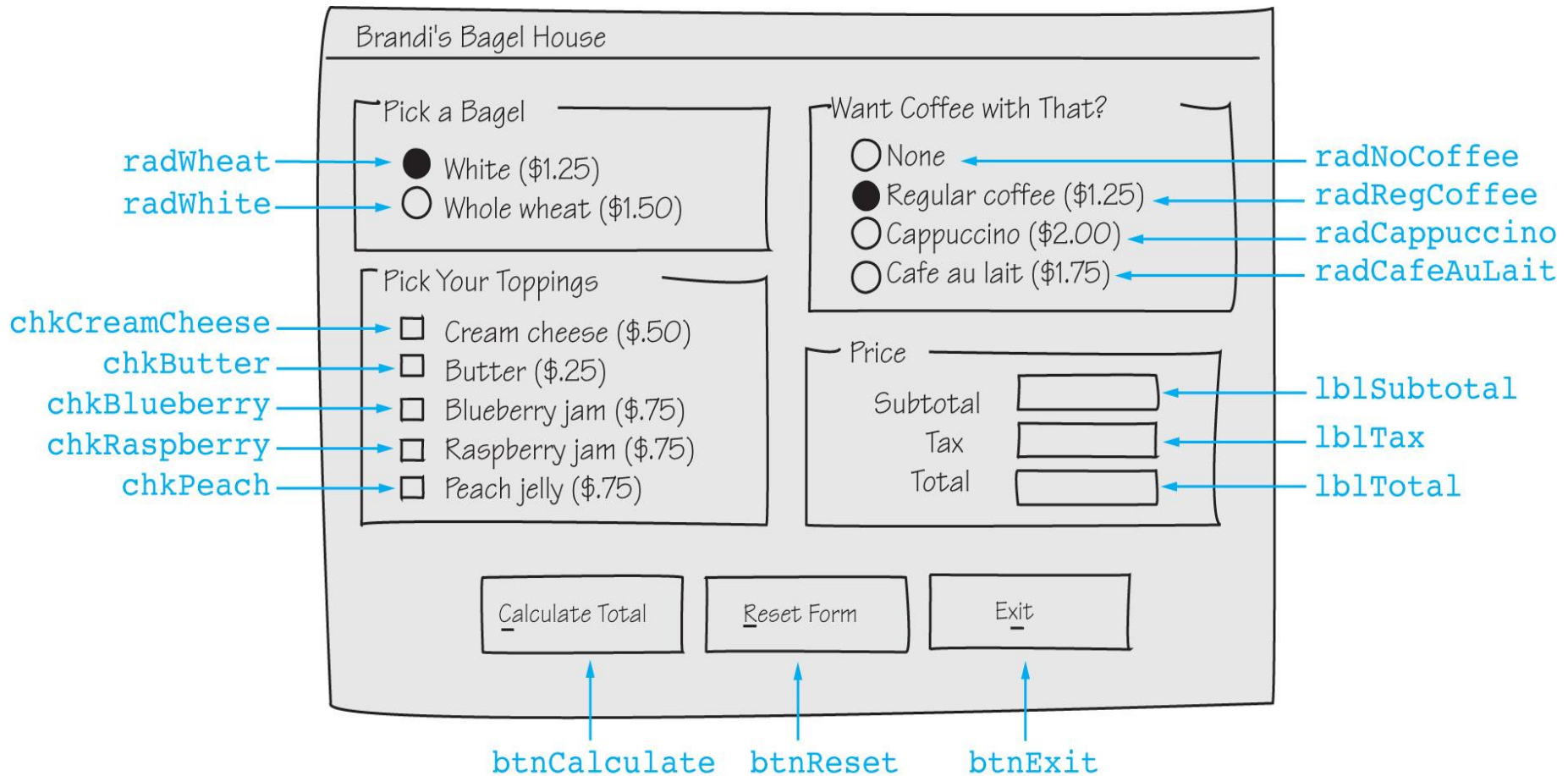
© 2011 Pearson Addison-Wesley. All rights reserved.

# Introduction

- The owner of Brandi's Bagel House has asked you to write an application that her staff can use to record an order as it is called in
- Customers may call in and order
  - White and whole wheat bagels with a variety of toppings
  - Three different types of coffee
- The application should display
  - The total of the order, including **6%** sales tax

- **Bagels:**
    - White bagel \$1.25
    - Whole wheat bagel \$1.50
  - **Toppings:**
    - Cream cheese \$0.50
    - Butter \$0.25
    - Blueberry jam \$0.75
    - Raspberry jam \$0.75
    - Peach jelly \$0.75
  - **Coffee:**
    - Regular coffee \$1.25
    - Cappuccino \$2.00
    - Café au lait \$1.75
- (Note: Delivery for coffee alone is not offered.)*

# Sketch of Brandi's Bagel House Form



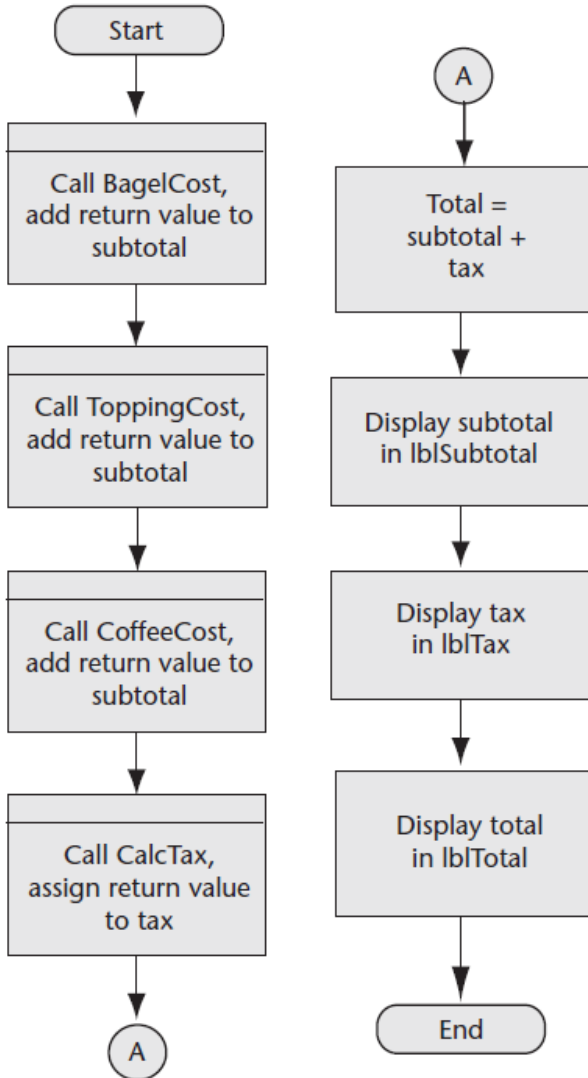


# Description of Click Event Handlers

Name	Description
<b>btnCalculate_Click</b>	Calculates and displays the total of an order Calls the following functions: <b>BagelCost, CoffeeCost, ToppingCost, and CalcTax</b>
<b>btnExit_Click</b>	Ends the application
<b>btnReset_Click</b>	Resets the controls on the form to their initial values Calls the following procedures: <b>ResetBagels, ResetToppings, ResetCoffee, and ResetPrice</b>

# btnCalculate\_Click

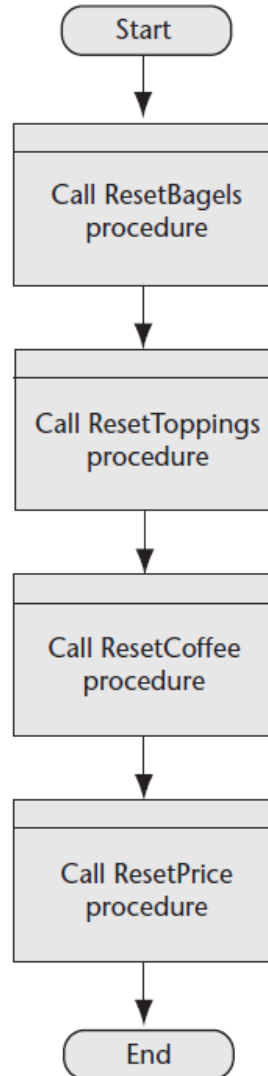
## Flowchart and Pseudocode



***subtotal = BagelCost() + ToppingCost() + CoffeeCost()***  
***tax = CalcTax(subtotal)***  
***total = subtotal + tax***  
***lblSubtotal.Text = subtotal***  
***lblTax.Text = tax***  
***lblTotal.Text = total***

# btnReset\_Click

## Flowchart and Pseudocode

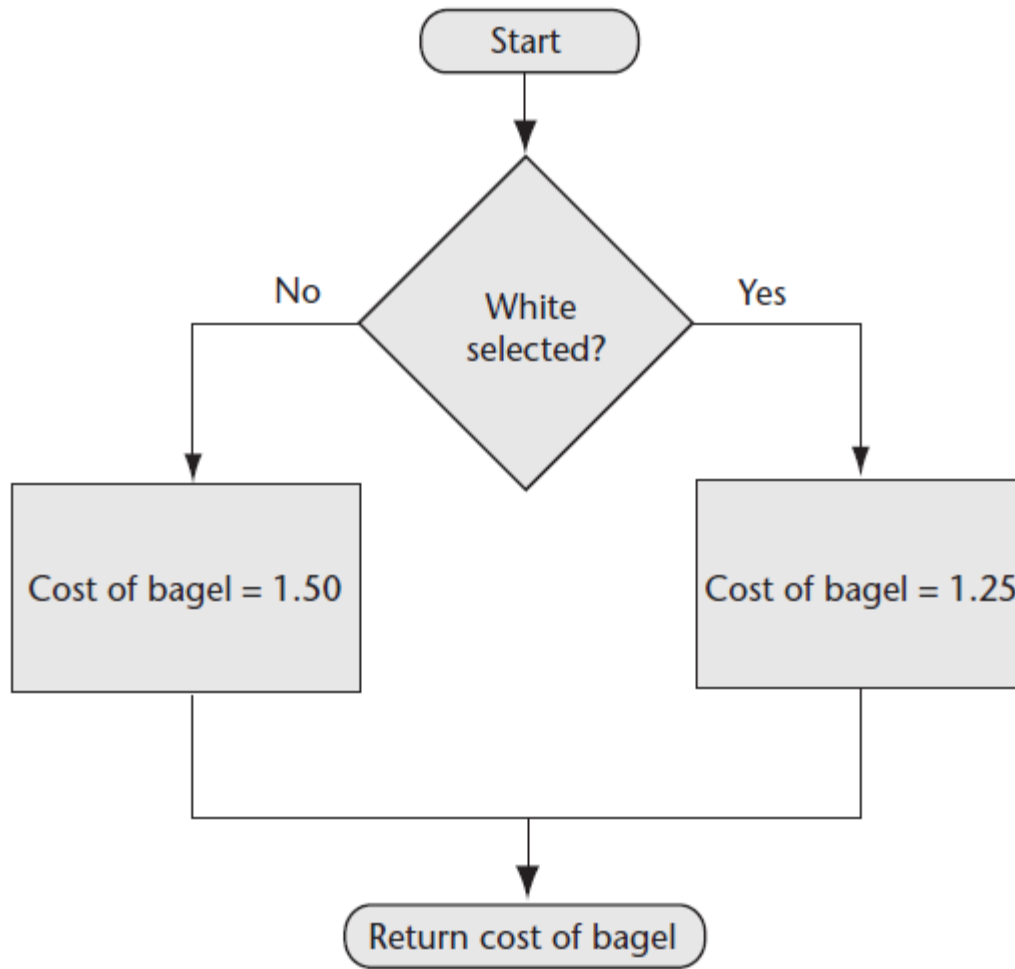


***ResetBagels()***  
***ResetToppings()***  
***ResetCoffee()***  
***ResetPrice()***

# Description of Functions

Name	Description
<b>BagelCost</b>	Returns the price of the selected bagel
<b>ToppingCost</b>	Returns the total price of the selected toppings
<b>CoffeeCost</b>	Returns the price of the selected coffee
<b>CalcTax</b>	Accepts the amount of a sale as an argument Returns the amount of sales tax on that amount The tax rate is stored in a class-level constant, <b>decTAX_RATE</b>

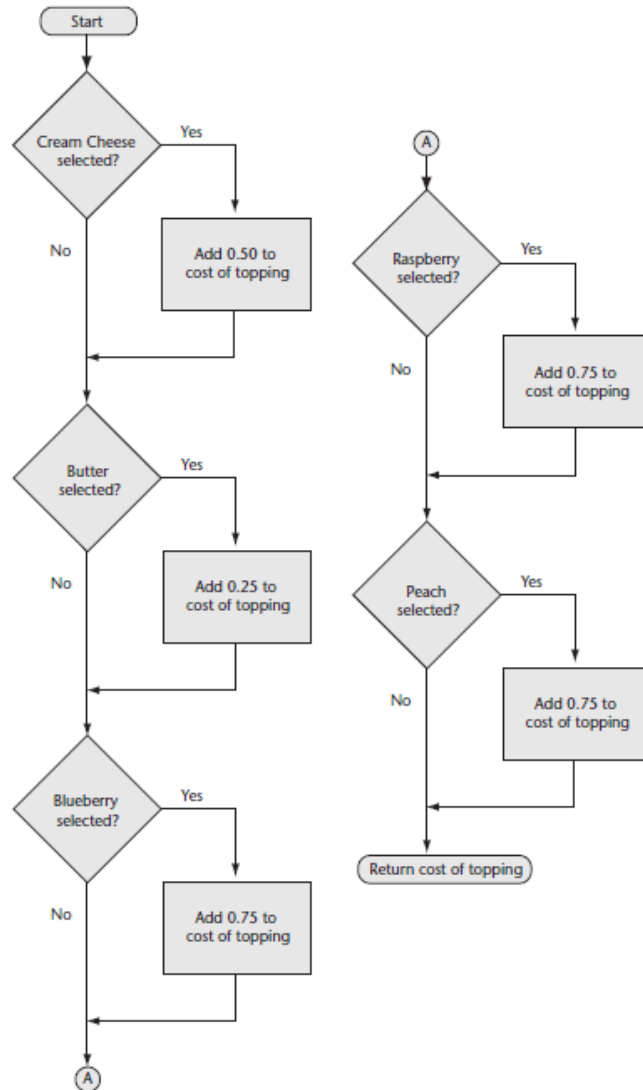
# BagelCost Function Flowchart and Pseudocode



***If White Is Selected Then  
cost of bagel = 1.25  
Else  
cost of bagel = 1.5  
End If  
Return cost of bagel***

# ToppingCost Function

## Flowchart and Pseudocode



***cost of topping = 0.0***

***If Cream Cheese Is Selected Then***

***cost of topping += 0.5***

***End If***

***If Butter Is Selected Then***

***cost of topping += 0.25***

***End If***

***If Blueberry Is Selected Then***

***cost of topping += 0.75***

***End If***

***If Raspberry Is Selected Then***

***cost of topping += 0.75***

***End If***

***If Peach Is Selected Then***

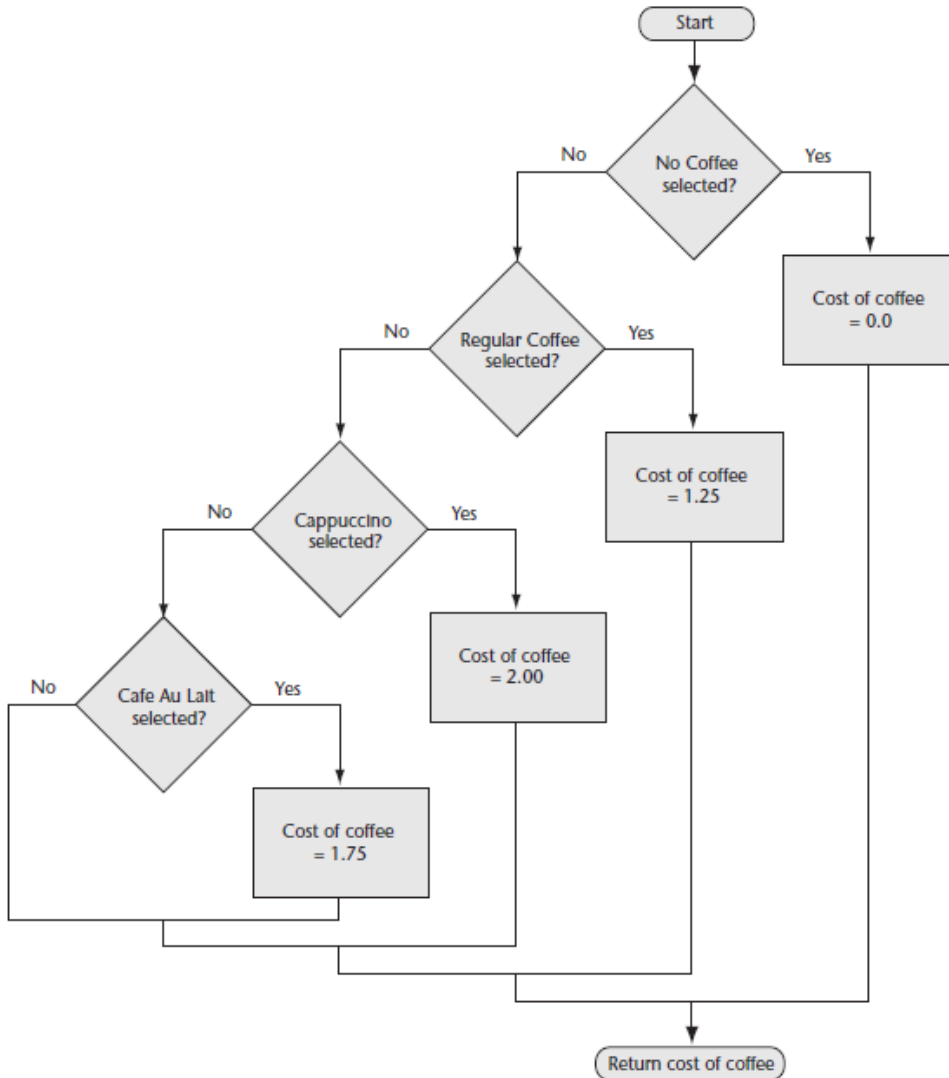
***cost of topping += 0.75***

***End If***

***Return cost of topping***

# CoffeeCost Function

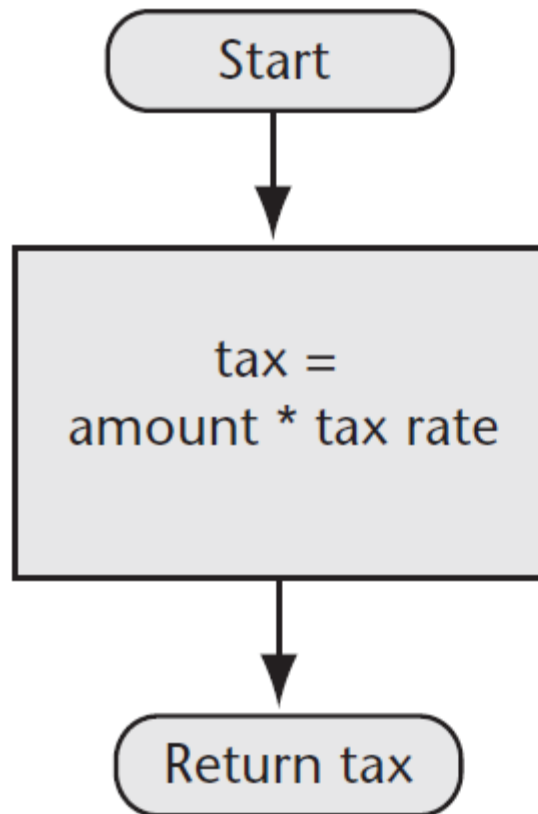
## Flowchart and Pseudocode



***If No Coffee Is Selected Then  
cost of coffee = 0  
Elseif Regular Coffee Is Selected Then  
cost of coffee = 1.25  
Elseif Cappuccino Is Selected Then  
cost of coffee = 2  
Elseif Café Au Lait Is Selected Then  
cost of coffee = 1.75  
End If  
Return cost of coffee***

# CalcTax Function

## Flowchart and Pseudocode



***sales tax = amount \* tax rate***  
***Return sales tax***

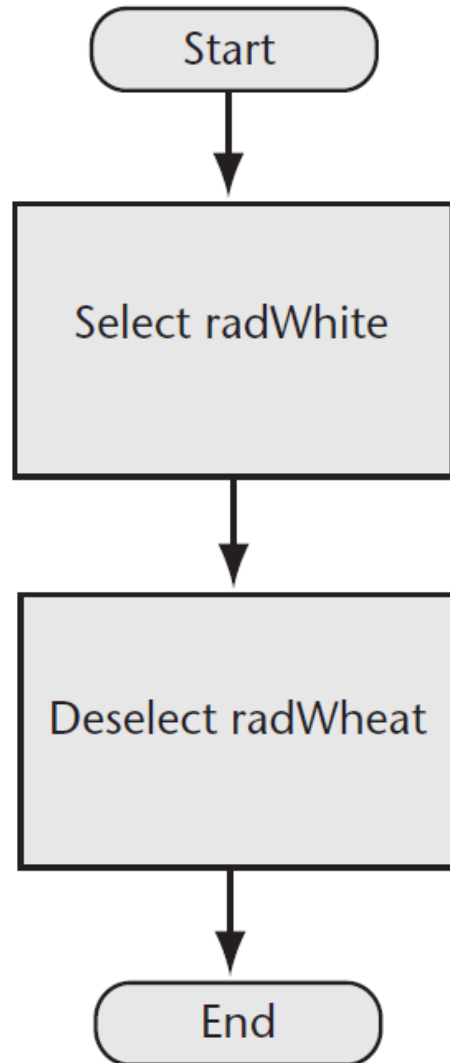
Note: amount is the  
function parameter



# Description of Procedures

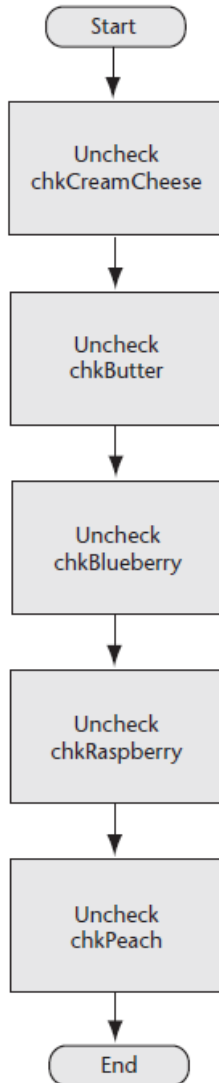
Name	Description
<b>ResetBagels</b>	Resets the bagel type radio buttons to their initial value
<b>ResetToppings</b>	Resets the topping check boxes to unchecked
<b>ResetCoffee</b>	Resets the coffee radio buttons to their initial values
<b>ResetPrice</b>	Sets the <b>Text</b> property of the <b>lblSubtotal</b> , <b>lblTax</b> , and <b>lblTotal</b> labels to <b>String.Empty</b>

# ResetBagels Procedure Flowchart and Pseudocode



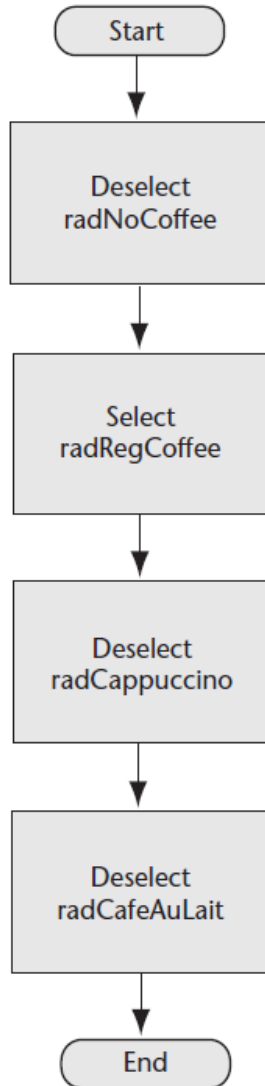
***radWhite = Selected***  
***radWheat = Deselected***

# ResetToppings Procedure Flowchart and Pseudocode



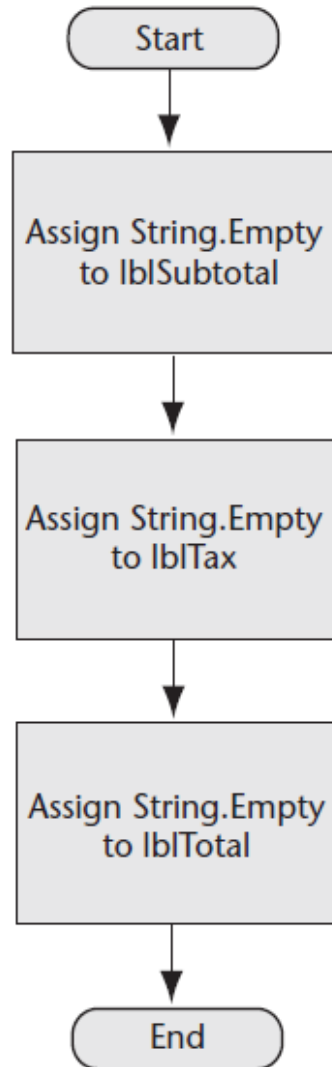
***chkCreamCheese = Unchecked***  
***chkButter = Unchecked***  
***chkBlueberry = Unchecked***  
***chkRaspberry = Unchecked***  
***chkPeach = Unchecked***

# ResetCoffee Procedure Flowchart and Pseudocode



***radNoCoffee = Deselected***  
***radRegCoffee = Selected***  
***radCappuccino = Deselected***  
***radCafeAuLait = Deselected***

# ResetPrice Procedure Flowchart and Pseudocode



***lblSubtotal.Text = String.Empty***  
***lblTax.Text = String.Empty***  
***lblTotal.Text = String.Empty***

# Brandi's Bagel House Form

The screenshot shows a Windows application window titled "Brandi's Bagel House". The window contains a form with the following sections:

- Pick a Bagel:** Two radio buttons are present. The first is labeled "White (\$1.25)" and is selected. The second is labeled "Whole Wheat (\$1.50)".
- Pick Your Toppings:** Five checkboxes are listed. None are selected. The options are: "Cream Cheese (\$.50)", "Butter (\$.25)", "Blueberry Jam (\$.75)", "Raspberry Jam (\$.75)", and "Peach Jelly (\$.75)".
- Want Coffee with That?:** Three radio buttons are present. The first is labeled "None". The second is labeled "Regular Coffee (\$1.25)" and is selected. The third is labeled "Cappuccino (\$2.00)". The fourth is labeled "Cafe au lait (\$1.75)".
- Price:** Three text input fields are stacked vertically. The first is labeled "Subtotal", the second "Tax", and the third "Total".

At the bottom of the window, there are three buttons: "Calculate Total", "Reset Form", and "Exit".