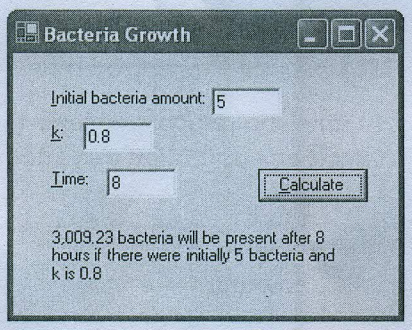
Exercise 13 — Bacteria Growth

The formula $y = ne^{kt}$ can be used for estimating growth where:

- y is the final amount
- n is the initial amount
- k is a constant
- t is the time

For example, this formula could be used for estimating population growth in a region or for estimating cell grown in a lab experiment. Create a Bacteria Growth application that calculates how many bacteria will be present based on this formula. The application interface should look similar to:

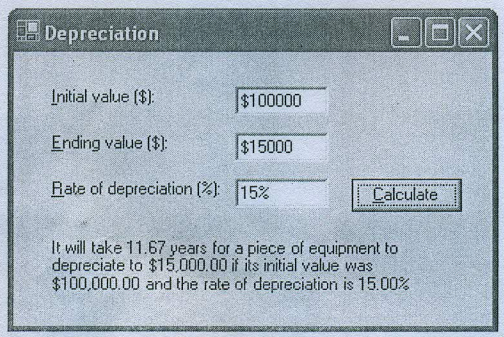


Exercise 14 — Depreciation

The formula $V_n = P (1 + r)^n$ is can be used to estimate depreciation or appreciation where:

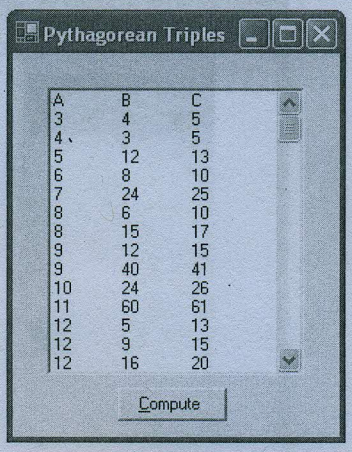
- V_n is the value at the end of n years
- P is the initial value of the equipment
- r is the rate of appreciation (positive) or depreciation (negative)
- n is the time in years

For example, this formula could be used to determine the current value of a mainframe that a company has owned for 10 years. From this formula you can also determine how long it will take a piece of equipment to depreciate to a specific value using the formula: $n = \log (V_n / P) / \log (1 + r)$. Create a Depreciation application that calculates how long it will take a piece of equipment to depreciate using this formula. The application interface should look similar to:



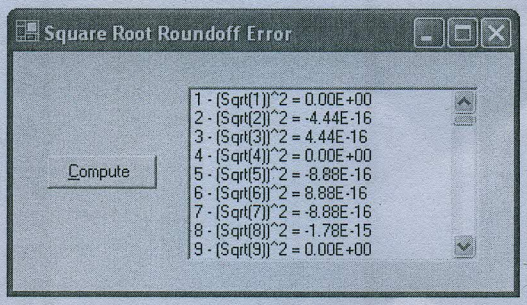
Exercise 5 — Pythagorean Triples

A Pythagorean triple is a set of three integers that solves the equation $a^2 + b^2 = c^2$. Create a Pythagorean Triples application that displays all Pythagorean triples with values of A and B less than 100. The program code should include a PerfectSquare() function like the one created in Exercise 1. The application interface should look similar to the following after clicking Compute:



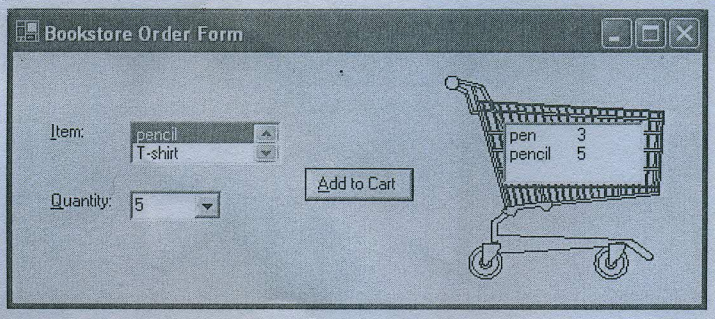
Exercise 6 — Square Root Roundoff Error

Create a Square Root Roundoff Error application that compares the square of the square root of a number with the number itself for all integers from 1 to 100. This difference in values is due to the computer's rounding error. The application interface should look similar to the following after clicking Compute:



Exercise 7 — Bookstore Order Form

Create a Bookstore Order Form application the prompts the user to select different items and quantities to purchase and then displays the order information in a list box that is on top of a cart.gif graphic, included in the data files for this text. The application interface should look similar to:



LowestToHighest() is passed the addresses of intNum1 and intNum2. Since intLowest has a greater value than intHighest, intTemp is assigned 30, then intLowest is assigned 12, and finally intHighest is assigned 30, the value of intTemp.

The following points are important to keep in mind when working with procedures with reference parameters:

- The order of the arguments corresponds to the order of the parameters.
- **ByRef** parameters accept only variable arguments. For example, a run-time error is generated when LowestToHighest() is called with constants, as in the statement:

  ```
  Call LowestToHighest(5, 1)        'Bad Call Statement
  ```

- Variable arguments passed by reference may be changed by the procedure.

Review 3

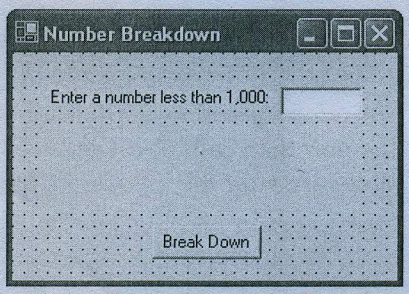
In this review you will create the Number Breakdown application, which displays the separate digits of a number that contains up to three digits.

① CREATE A NEW PROJECT

Create a Windows application named Number Breakdown.

② CREATE THE INTERFACE

Refer to the form below to add, position, and size objects. Use the table below to set properties.



Object	Name	Text
Form1		Number Breakdown
Label1	lblPrompt	Enter a number less than 1,000:
TextBox1	txtNumber	*empty*
Label2	lblDigits	*empty*
Button1	btnBreakDown	Break Down

③ WRITE THE APPLICATION CODE

a. Display the Code window.

b. Add comments that include your name and today's date.

c. Create a btnBreakDown_Click event procedure and then add the statements:

```
Dim intNumberEntered As Integer
Dim intOnesDigit As Integer
Dim intTensDigit As Integer
Dim intHundredsDigit As Integer

intNumberEntered = Val(Me.txtNumber.Text)
If intNumberEntered < 10 Then
  Me.lblDigits.Text = "The first digit is: " & intNumberEntered
ElseIf intNumberEntered < 100 Then
  Call TwoDigits(intNumberEntered, intTensDigit, intOnesDigit)
  Me.lblDigits.Text = "The first digit is: " & intTensDigit & _
    vbCrLf & "The second digit is: " & intOnesDigit
ElseIf intNumberEntered < 1000 Then
  Call ThreeDigits(intNumberEntered, intHundredsDigit, intTensDigit, intOnesDigit)
  Me.lblDigits.Text = "The first digit is: " & intHundredsDigit & _
    vbCrLf & "The second digit is: " & intTensDigit & _
    vbCrLf & "The third digit is: " & intOnesDigit
Else
  Me.lblDigits.Text = "Invalid entry."
End If
```

d. Add the TwoDigits procedure:

```
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'The digits of a two-digit number are returned in separate parameters
'
'pre: intNum is a number less than 100 and greater than -100.
'post: intFirstDigit is a number between 0 and 9 inclusive.
'intSecondDigit is a number between 0 and 9 inclusive.
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Sub TwoDigits(ByVal intNum As Integer, ByRef intFirstDigit As Integer, _
ByRef intSecondDigit As Integer)

  intFirstDigit = intNum \ 10
  intSecondDigit = intNum Mod 10

End Sub
```

e. The ThreeDigits procedure uses integer division to determine the third digit (the hundreds digit) of the number and then calls TwoDigits to get the first two digits of a number. Add the ThreeDigits procedure after the TwoDigits procedure:

```
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'The digits of a three-digit number are returned in separate parameters
'
'pre: intNum is a number less than 1000 and greater than -1000.
'post: intFirstDigit is a number between 0 and 9 inclusive.
'intSecondDigit is a number between 0 and 9 inclusive.
'intThirdDigit is a number between 0 and 9 inclusive.
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Sub ThreeDigits(ByVal intNum As Integer, ByRef intFirstDigit As Integer, _
ByRef intSecondDigit As Integer, ByRef intThirdDigit As Integer)

  intFirstDigit = intNum \ 100
  intNum = intNum Mod 100
  Call TwoDigits(intNum, intSecondDigit, intThirdDigit)

End Sub
```

f. Add a TextChanged event procedure for the text box.


```
Public Class Form1
    Inherits System.Windows.Forms.Form

Windows Form Designer generated code

    '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    'Updates the price of a hot dog.
    '
    'pre: sender has a valid Tag expression.
    'post: The price has been updated in the label on the form.
    '~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    Private Sub Toppings_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles chkRelish.Click, chkKraut.Click, chkCheese.Click

        Const decRELISH As Decimal = 0.1
        Const decKRAUT As Decimal = 0.25
        Const decCHEESE As Decimal = 0.5
        Static decPrice As Decimal = 2              'hot dog base price

        Dim chkSelectedTopping As CheckBox = sender 'object that raised event
        If chkSelectedTopping.Checked Then          'topping selected
            Select Case chkSelectedTopping.Tag
                Case "Relish"
                    decPrice = decPrice + decRELISH
                Case "Kraut"
                    decPrice = decPrice + decKRAUT
                Case "Cheese"
                    decPrice = decPrice + decCHEESE
            End Select
        Else                                        'topping cleared
            Select Case chkSelectedTopping.Tag
                Case "Relish"
                    decPrice = decPrice - decRELISH
                Case "Kraut"
                    decPrice = decPrice - decKRAUT
                Case "Cheese"
                    decPrice = decPrice - decCHEESE
            End Select
        End If

        Me.lblCurrentPrice.Text = "$" & decPrice    'display updated price

    End Sub

End Class
```

The Tag property for each of the check boxes was set to a descriptive string. This property value is then used to determine which action to take.

Review 6

In this review, you will create the Shell Game application. The Shell Game displays pictures of three shells. Under one shell is a "hidden" pearl. The user guesses which shell is hiding the pearl by clicking a shell. The hidden pearl is then displayed along with a message telling the player if a correct guess was made. The pearl is hidden again after each try so that the game can be played again and again. An algorithm for implementing this kind of guessing game is:

1. Generate a random number between 1 and 3. Use this number to determine which shell is "hiding" the pearl.
2. Show a pearl picture below the shell that corresponds to the random number.
3. Using one procedure that handles click events for all three shells, determine if the user clicked the shell that corresponds to the random number.
4. Display a message to the player. The player won, if the shell clicked corresponds to the generated random number.
5. Make the pearl picture no longer visible so that the game can be played again without quitting and running the application again.

Review 1

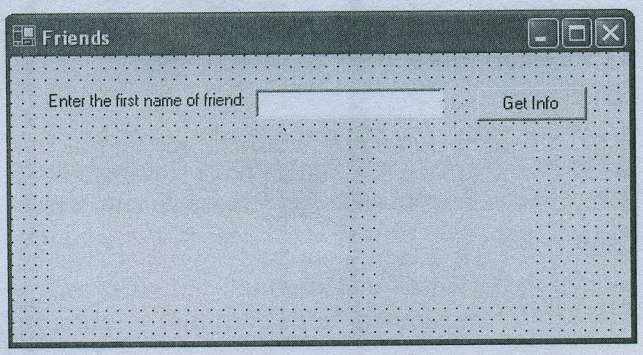
In this review you will create the Friends application with **Sub** procedures. Note that the image files should be placed in the bin folder located in the project folder after creating the project.

① CREATE A NEW PROJECT

Create a Windows application named Friends.

② CREATE THE INTERFACE

Refer to the form below to add, position, and size objects. Use the table below to set properties.



Object	Name	Text	Image	Size
Form1		Friends		
Label1	lblPrompt	*see interface*		
TextBox1	txtFriendName	*empty*		
Label2	lblFriendInfo	*empty*		
PictureBox1	picFriendPhoto		*empty*	100, 100
Button1	btnGetInfo	Get Info		

③ WRITE THE APPLICATION CODE

a. Display the Code window.

b. Add comments that include your name and today's date.

c. Create a btnGetInfo_Click event procedure and then add the following statements. Note that a blue wavy underline will appear below the procedure names because they do not yet exist:

```
Dim strFriendName As String
strFriendName = Me.txtFriendName.Text

'Show friend info
Select Case strFriendName.ToUpper
  Case "SHANA"
    Call SHANAInfo()
  Case "LUIZ"
    Call LUIZInfo()
  Case "CRIS"
    Call CRISInfo()
  Case Else
    MessageBox.Show("Sorry, no information available.")
End Select
```

d. After the btnGetInfo_Click event procedure, type **Sub** SHANAInfo and then press Enter. An **End Sub** statement is added and the insertion point is placed in the body of the new procedure. Add the following statements to the procedure. Note that your filename path may need to be different:

```
Sub SHANAInfo()

    Me.picFriendPhoto.Image = Image.FromFile("shana.bmp")
    Me.lblFriendInfo.Text = "Shana's birthday is June 24. Her favorite animal" & _
    " is the dolphin, her favorite color is blue, and she likes to do extreme inline skating."

End Sub
```

e. Create a LUIZInfo() procedure:

```
Sub LUIZInfo()

    Me.picFriendPhoto.Image = Image.FromFile("luiz.bmp")
    Me.lblFriendInfo.Text = "Luiz's birthday is August 21. His favorite animal" & _
    " is the tiger, his favorite color is green, and he likes to do gymnastics."

End Sub
```

f. Create a CRISInfo() procedure:

```
Sub CRISInfo()

    Me.picFriendPhoto.Image = Image.FromFile("cris.bmp")
    Me.lblFriendInfo.Text = "Cris' birthday is September 20. His favorite animal" & _
    " is any kind of bird, his favorite color is yellow, and he likes to play the guitar."

End Sub
```

④ RUN THE APPLICATION

a. Save the modified Friends project and then run the application. Type Shana and then click Get Info. A picture and information are displayed. Display information for Luiz, Cris, and then a name that does not have a procedure.

b. Close the Friends application.

⑤ PRINT THE CODE AND THEN CLOSE THE PROJECT

7.3 Value Parameters

parameter passing

A procedure often needs data in order to complete its task. Data is given, or *passed*, to a procedure by enclosing it in parentheses in the procedure call. For example, the statement below calls a procedure named GiveHint() and passes it two values, intSecretNumber and intGuess:

```
Call GiveHint(intSecretNumber, intGuess)
```

argument

A variable or value passed to a procedure is called an *argument*. In the statement, intSecretNumber and intGuess are the arguments to be used by the procedure.

A procedure that requires arguments is declared with *parameters* and takes the following form:

```
Sub ProcedureName(ByVal parameter1 As type, ...)
  statements
End Sub
```

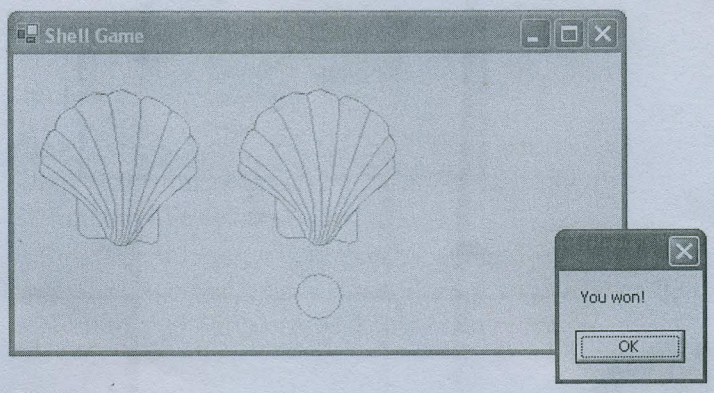
ProcedureName is the name describing the procedure. **ByVal** indicates that the parameter is a value parameter, *parameter1* is the name of the parameter, and *type* is the data type of the expected value. A value parameter is only used as a local variable by the called procedure. This

Exercise 11 — Shell Game

The Shell Game application from Review 6 gives the user just one chance at choosing the shell with the pearl. Modify the Shell Game application to give the user a better chance of finding the pearl:

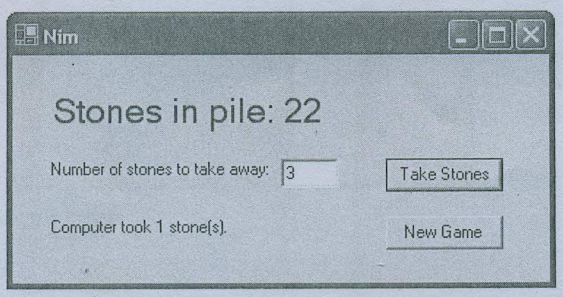
- After the user selects a shell but before the hidden pearl is displayed, remove (hide) one of the other two shells that does not contain the pearl.
- Use an input box to ask the user if he or she wants to keep the original guess or choose the remaining shell as the new guess.
- Display the result in a message box.

Improve the Shell Game application by appropriately separating tasks into procedures and functions, including using the RndInt() from Review 10 to generate a random number from 1 to 3 for the shell that hides the pearl. The application interface should look similar to the following after playing one game:



Exercise 12 — Nim

The game of Nim starts with a random number of stones between 15 and 30. Two players alternate turns and on each turn may take either 1, 2, or 3 stones from the pile. The player forced to take the last stone loses. Create a Nim application that allows the user to play against the computer. In this version of the game, the application generates the number of stones to begin with, the number of stones the computer takes, and the user goes first. The application interface should look similar to:



The program code should:

- prevent the user and the computer from taking an illegal number of stones. For example, neither should be allowed to take three stones when there are only 1 or 2 left.
- include ValidEntry() from Section 7.9 to check user input.
- include RndInt() from Review 10 to generate a random number from 1 to 3 for the computer's turn to remove stones from the pile.
- include separate procedures for the user's turn and the computer's turn.

Review 3

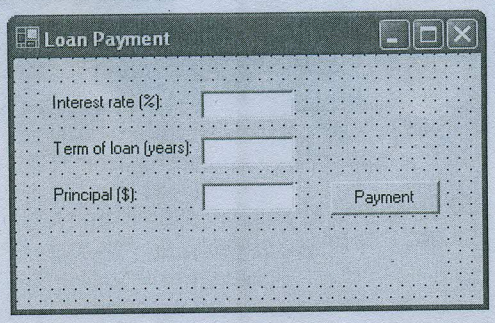
In this review you will create the Loan Payment application.

① CREATE A NEW PROJECT

Create a Windows application named Loan Payment.

② CREATE THE INTERFACE

Refer to the form below to add, position, and size objects. Use the table below to set properties.



Object	Name	Text
Form1		Loan Payment
Label1	lblRatePrompt	*see interface*
TextBox1	txtRate	*empty*
Label2	lblTermPrompt	*see interface*
TextBox2	txtTerm	*empty*
Label3	lblPrincipalPrompt	*see interface*
TextBox3	txtPrincipal	*empty*
Label4	lblMonthlyPayment	*empty*
Button1	btnPayment	Payment

③ WRITE THE APPLICATION CODE

a. Display the Code window.

b. Add comments that include your name and today's date.

c. Create a btnPayment_Click event procedure and then add the statements:

```
Dim sngRate, sngPrincipal As Single
Dim intTerm As Integer
Dim sngPayment As Single
Dim blnValidData As Boolean

'Get interest rate
GetPercentAmount(Me.txtRate, sngRate, blnValidData)

'Get term if interest rate is valid
If blnValidData Then
   intTerm = Val(Me.txtTerm.Text)
   If intTerm <= 0 Then
      blnValidData = False
   End If
End If

'Get principal if interest rate and term are valid
If blnValidData Then
   GetDollarAmount(Me.txtPrincipal, sngPrincipal, blnValidData)
End If
```

```
'Calculate payment if all data entered by user is valid
If blnValidData Then
   sngPayment = Pmt(sngRate / 12, intTerm * 12, -sngPrincipal)
   Me.lblMonthlyPayment.Text = "The monthly payment for a loan of " & _
   Format(sngPrincipal, "Currency") & " at " & Format(sngRate, "Percent") _
   & " for " & intTerm & " years is " & Format(sngPayment, "Currency")
Else
   Me.lblMonthlyPayment.Text = "Enter valid data."
End If
```

d. Add the GetPercentAmount() and GetDollarAmount() procedures shown in Section 8.6.

e. Create an event procedure that handles TextChanged events for all three text boxes. Change the procedure name to NewDataEntered and add the statement:

```
Me.lblMonthlyPayment.Text = Nothing
```

④ RUN THE APPLICATION

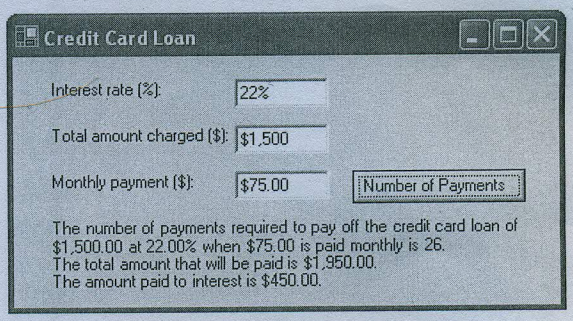
a. Save the modified Loan Payment project and then run and test the application.

b. Close the Loan Payment application.

⑤ PRINT THE CODE AND THEN CLOSE THE PROJECT

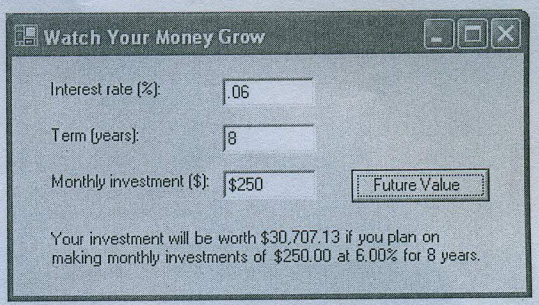
Review 4

Create a Credit Card Loan application that prompts the user for the interest rate of the card, the total amount charged, and the minimum payment desired. Clicking Number of Payments displays the number of payments required to pay off the credit card loan, the total amount that will be paid, and the amount that will be paid to interest. The application interface should look similar to that shown on the right.



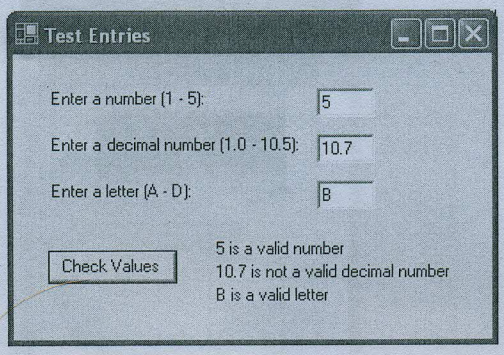
Review 5

Create a Watch Your Money Grow application that prompts the user for the interest rate, term, and the amount invested each month and then displays the value of the investment (future value) when Future Value is clicked. The application interface should look similar to that shown on the right.



Exercise 7 — Test Entries

Validating user input is often required in programs. Create a Test Entries application that prompts the user for an integer, decimal number, and letter and then determines if the values are valid. The application interface should look similar to:



The program code should include:

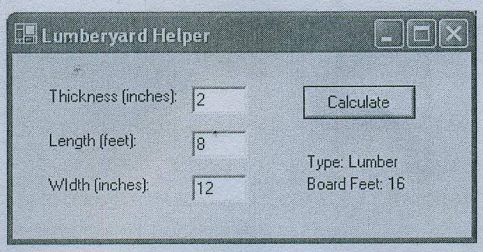
- a ValidInt() function that has intHighNum, intLowNum, and intNumber parameters and returns True if intNumber is in the range intLowNum to intHighNum, and False otherwise.
- a ValidSingle() function that has sngHighNum, sngLowNum, and sngNumber parameters and returns True if sngNumber is in the range sngLowNum to sngHighNum, and False otherwise.
- a ValidChar() function that has chrHighChar, chrLowChar, chrCharacter parameters and returns True if chrCharacter is in the range chrLowChar to chrHighChar, and False otherwise.

Exercise 8 — Lumberyard Helper

The basic unit of lumber measurement is the board foot. One board foot is the cubic content of a piece of wood 12 inches by 12 inches by 1 inch thick. For example, a board that is 1 inch thick by 8 feet long by 12 inches wide is 8 board feet:

((1 * (8 * 12) * 12) / (12 * 12 * 1)) = 8

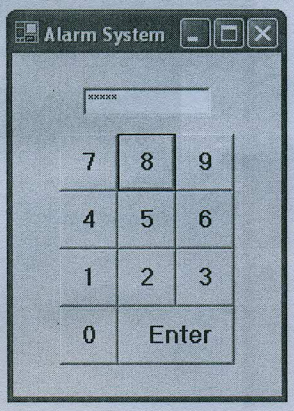
Milled wood is cut to standardized sizes called board, lumber, and timber. A board is one-inch thick or less, timber is more than four inches thick, and lumber is anything between one and four inches thick. Create a Lumberyard Helper application that prompts the user for the thickness, length, and width of a piece of wood and then displays the board feet and the classification of the cut. The application interface should look similar to:



The program code should include a BoardFeet() function that has sngThickness, sngLength, and sngWidth parameters and returns the number of board feet and a CutClassification() function that has a sngThickness parameter and returns the classification of the cut.

Exercise 5 — Alarm System

An office building uses an alarm system that is turned off by entering a master code and then pressing Enter. The master code is 62498. Create an Alarm System application that displays a message box with an appropriate message after a code is typed and then Enter is clicked. The application interface should look similar to the following after clicking five number buttons:



Exercise 6 — Metric Conversion

The following formulas can be used to convert English units of measurement to metric units:

inches * 2.54 = centimeters
feet * 30 = centimeters
yards * 0.91 = meters
miles * 1.6 = kilometers

Create a Metric Conversion application that prompts the user for a number and then converts it from inches to centimeters, feet to centimeters, yards to meters, and miles to kilometers and vice versa when a button is clicked. The program code should include separate functions to perform the conversions. The application interface should look similar to: