Visual Basic Lecture Notes 3

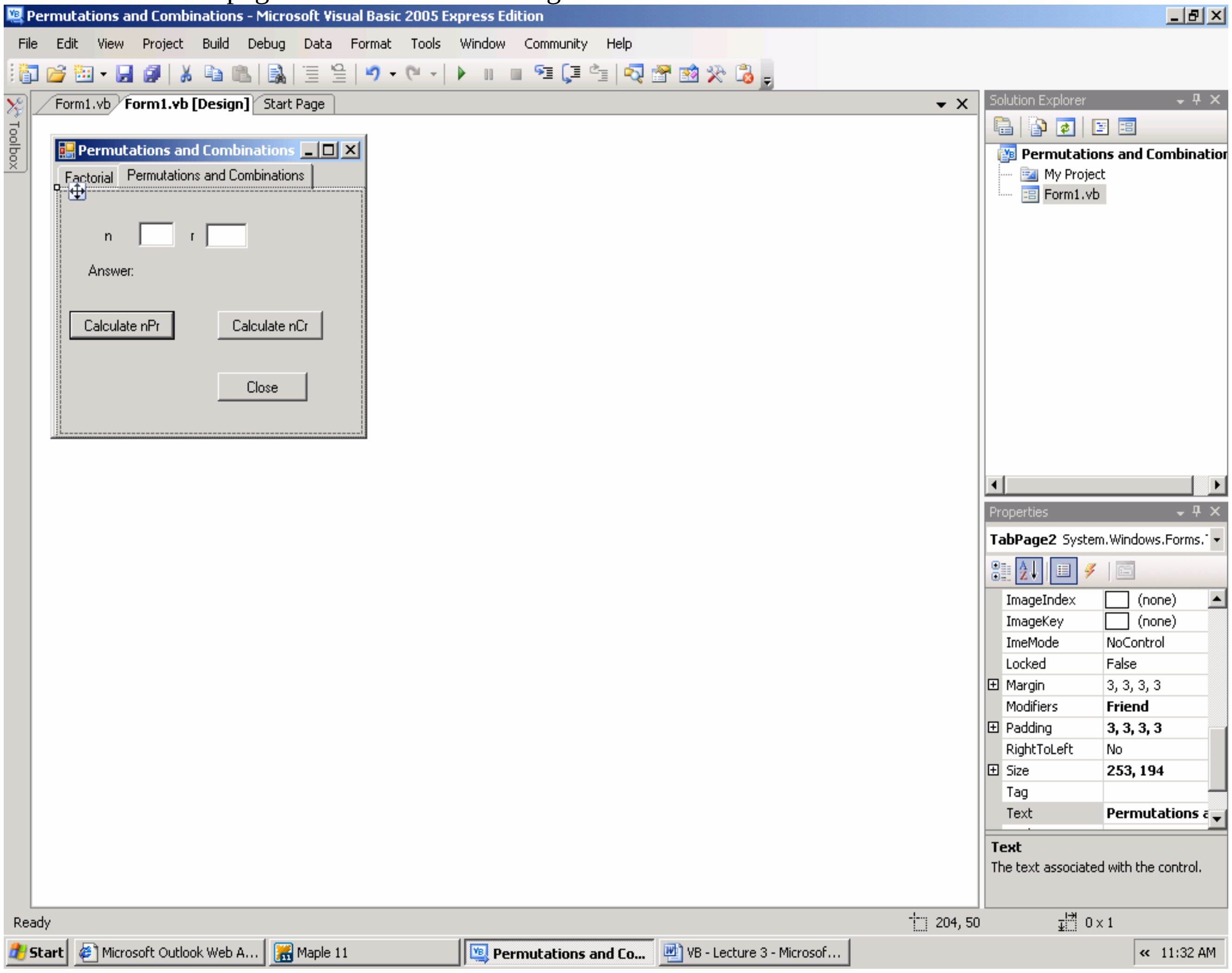
When we studied Maple, we often wrote procedures that could be called upon to perform a function and return a value to the program. In this section, we will study how to create procedures in VB.

Recall from your algebra class the terms permutation and combination. The number of permutations of n objects taken r at a time (often denoted nPr) is given by n!/(n-r)!. This result is used when the order of the objects is not important. The number of combinations of n objects taken r at a time (denoted by nCr) is given by n!/[r!(n-r)!].

In our last section of notes, we created a program that calculated the factorial of a number. We can call upon it to help us out now. Create a new project in VB called Permutations and Combinations.

Change the name of the Form to Permutations and Combinations. You may need to expand the size of the form in order to see the whole name.

Let's incorporate the use of **Tabbed** windows. From the Toolbox, in the section called Containers, select **Tab Control**. Place it on the form and resize it to fill the entire form. On the first tab page, re-create the factorial program from Lecture 2 Homework. Name that tab page **Factorial.** This page should include a text box to receive user input, labels to output the solution to and display the appropriate text, a Calculate button, and a close button Make the second tab page look like the following:



Permutations and Combinations - Microsoft Visual Basic 2005 Express Edition
File Edit View Project Build Debug Data Format Tools Window Community Help
Form1.vb Form1.vb [Design] Start Page
Toolbox
Permutations and Combinations
Factorial
Permutations and Combinations
n
r
Answer:
Calculate nPr
Calculate nCr
Close
Solution Explorer
Permutations and Combinatior
My Project
Form1.vb
Properties
Ready
204, 50
0 x 1
Start
Microsoft Outlook Web A...
Maple 11
Permutations and Co...
VB - Lecture 3 - Microsof...
11:32 AM

TabPage2 System.Windows.Forms.

ImageIndex	(none)
ImageKey	(none)
ImeMode	NoControl
Locked	False
Margin	3, 3, 3, 3
Modifiers	**Friend**
Padding	**3, 3, 3, 3**
RightToLeft	No
Size	**253, 194**
Tag	
Text	**Permutations a**

Text
The text associated with the control.

Change its name the **Permutation and Combinations**, and add the appropriate labels, textboxes, and buttons. Make sure to add an extra label after the one whose text is "Answer:" so we have a place to output our result to.

Whether we simply want to calculate the factorial of a number or calculate permutations or combination, we will need the code for computing factorials. This time, let's create a procedure that we will reference for each of the buttons. In VB, a procedure that returns a value to the caller is called a **Function.** Go to the **Code View** and enter all of the following code:

```
Function Factorial(ByVal x As Integer)
  Dim i, fac As Integer
Try
   fac = Convert.ToInt32(x)
     If n = 0 Then
          Return 1
     Else
        For i = x - 1 To 1 Step -1
            fac *= i
        Next
     End If
Catch formatexceptionparameter As FormatException
MessageBox.Show("You must enter an integer", "Invalid Number Format", _
MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
Return fac
End Function
```

Here we have said that **Factorial** is a function (procedure) that receives the variable **x** which is an integer, and uses the loop to calculate the factorial. Notice that we did not have to declare the variables i and fac to be local as we did in Maple. Since we are using a function, all variables created within the function are automatically local to the procedure.

The lines Try and Catch are used to handle exceptions. Exceptions are events that can crash the program. For instance, since we are trying to retrieve an integer number from the user, we use the line **fac=Convert.ToInt32(x)** to take the number x that the procedure receives and convert it to type Integer 32-bit. If the number is not already an integer, this conversion would cause the program to crash, which is where the Catch line comes in. This particular type of exception is called a Format Exception, because the variable is not in the correct format for the conversion. With this code, the Catch will stop the **Format Exception Error** from crashing the program and display a **Message Box** telling the user to enter an integer.

The Factorial function is the only computational code that we will really require. Now we only need to add functionality to the rest of the buttons. Consider the button that is labeled **Calculate nPr**. Clicking on this button should cause the computer to retrieve the values of n and r from the text boxes, send the appropriate numbers to the Factorial function, and output the result to the label. Remember that by definition, nPr = n!/(n-r)!. The code to accomplish this goes like this:

```
Private Sub cmdnpr_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdnpr.Click
        Dim solution As Integer
        n = txtn.Text
        r = txtr.Text
        solution = (Factorial(n) / Factorial(n - r))
        lblsolution2.Text = solution
End Sub
```

Note that I changed the name of the button to **cmdnpr**. It is a good programming technique to name the objects according to what they do. I also changed the name of the text boxes to **txtn** and **txtr**, and the label for the solution to **lblsolution2**. The complete body of code for the program should look like this:

```
Imports System.Windows.Forms.Form
Public Class frmPermandComp
    Dim n, r As Integer
    Private Sub cmdfactorial_Click(ByVal sender As System.Object, ByVal e As
     System.EventArgs) Handles cmdfactorial.Click
        n = txtbx1.Text 'retrieve number for computing factorial
        lblsolution1.Text = Factorial(n)    'call the factorial function
    End Sub

  Private Sub cmdnpr_Click(ByVal sender As System.Object, ByVal e As
     System.EventArgs) Handles cmdnpr.Click
        Dim solution As Integer
        n = txtn.Text   'retrieve n
        r = txtr.Text   'retrieve r
        solution = (Factorial(n) / Factorial(n - r))    'calculate nPr
        lblsolution2.Text = solution    'display solution
    End Sub

  Private Sub cmdncr_Click(ByVal sender As System.Object, ByVal e As
     System.EventArgs) Handles cmdncr.Click
        Dim solution As Integer
        n = txtn.Text   'retrieve n
        r = txtr.Text   'retrieve r
        solution = Factorial(n) / (Factorial(r) * Factorial(n - r)) 'calculate nCr
        lblsolution2.Text = solution    'display solution
    End Sub

  Function Factorial(ByVal x As Integer)  'procedure to calculate factorials
        Dim fac,i As Integer  'variables to manipulate the solution
        Try     'handle exceptions
            fac = Convert.ToInt32(x)    'convert type to catch exceptions
            If fac = 0 Or fac = 1 Then  'handle 0! and 1!
                Return 1                'send 1 in these cases
            Else                         every other case
                For i from fac-1 to 1 Step -1     'count down to 1
                    fac*=i      'multiply the factorial
                Next
            End If
        Catch formatexceptionparameter As FormatException   'message box if Format
     Exception occurs
            MessageBox.Show("You must enter an integer", "Invalid Number Format", _
            MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
        Return fac
    End Function

  Private Sub cmdclose1_Click(ByVal sender As System.Object, ByVal e As
     System.EventArgs) Handles cmdclose1.Click
        Me.Close()  'close the program from the Factorial tab
    End Sub

  Private Sub cmdclose2_Click(ByVal sender As System.Object, ByVal e As
     System.EventArgs) Handles cmdclose2.Click
```

```
        Me.Close()  'close the program from the second tab
    End Sub
End Class
```

Now, let's consider a slightly different way of writing a factorial program. Our previous method involved the use of a loop to multiply the number by every number less than it. Let's consider another method that involves a **recursive procedure**. A recursive procedure is one which calls itself. Consider this property of factorials: n! = n * (n-1)! For example:

5! = 5*4*3*2*1
  = 5*4!

because 4! = 4*3*2*1. We can define the factorial function to call itself in this manner. We define 0! and 1! to equal 1. Any other number we send it will return n*(n-1)! Here's how to do this:

```
Function Factorial(ByVal x As Integer)  'procedure to calculate factorials
        Dim fac As Integer  'variable to manipulate as the solution
        Try     'handle exceptions
            fac = Convert.ToInt32(x)    'convert type to catch exceptions
            If fac = 0 Or fac = 1 Then  'handle 0! and 1!
                Return 1                'send 1 in these cases
            End If
        Catch formatexceptionparameter As FormatException   'message box if Format
        Exception occurs
            MessageBox.Show("You must enter an integer", "Invalid Number Format", _
            MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
        Return fac * Factorial(fac - 1)  'recall the factorial function
    End Function
```

Now the factorial function will continually call itself until it reaches 1, then compute the product. You may wonder if this method is more efficient that using the loop structure. It does, in fact, run more quickly than a For…Loop structure. Even though to us it seems that the two are identical, they are very different in terms of how the computer does the computations.

Homework: The Fibonacci sequence is defined by the following recursive algorithm:
$F_n = F_{n-1} + F_{n-2}$, where F_0=1 and F_1 =1. This gives the sequence 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, …

Write a program that takes the number n and returns the nth Fibonacci number. Use a recursive procedure to do this. If you want to be extremely creative, try to output all of the Fibonacci numbers up to the number n.