Visual basic tutorial problems, developed by Dr. Clement, clement@auburn.edu

EXCEL Visual Basic Tutorial Problems (Version January, 2011)
Dr. Prabhakar Clement
Arthur H. Feagin Distinguished Chair Professor
Department of Civil Engineering, Auburn University
Home page: http://www.eng.auburn.edu/users/clemept/
Email: clement@auburn.edu

How to activate the Developer Tab?

When you open EXCEL (in Office 2007) you should see a tab called "Developer" in the very end next to the "View" tab. If you don't then you need to activate it. You can do this by clicking on the office menu button (round button in left, where you have the print, save commands). At the very bottom of this menu you will see Excel options (see the figure below), click on that and then select (check mark) "Show Developer tab in the ribbon." Say ok. Now you should see the Developer tab.

Visual basic tutorial problems, developed by Dr. Clement, clement@auburn.edu
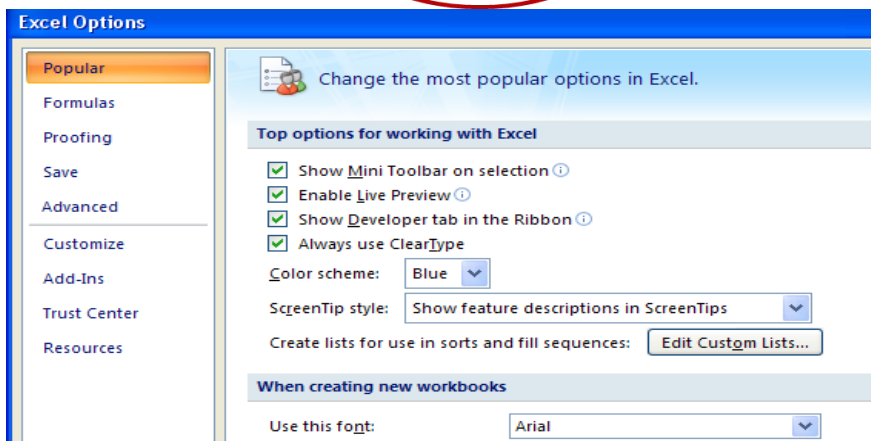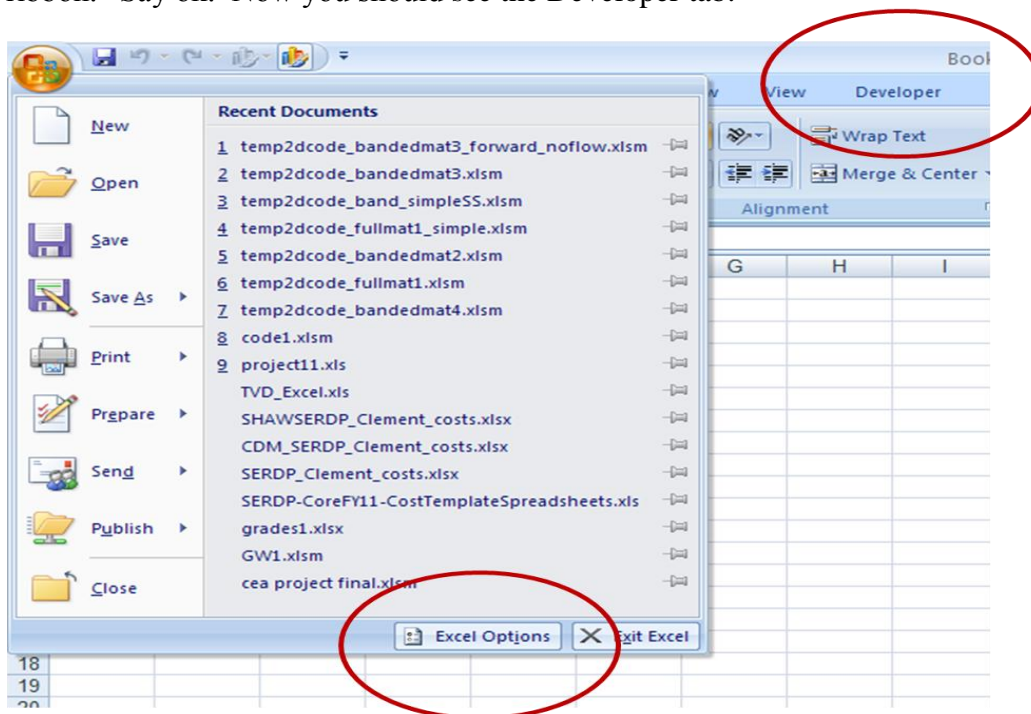
Note: Textbooks often follow different approaches for writing a VB code. The method shown here is a relatively simple and straight forward approach to write a VB code in EXCEL. The problem below shows how to create a VB button and rename it to name you like. But you don't have to rename, you can just leave it as the default name of "Sub CommandButton1_Click()" the program should still work.

Test Problem-0: Develop a simple calculator that will add or subtract numbers two numbers entered in a spreadsheet.

The final result will look like this:
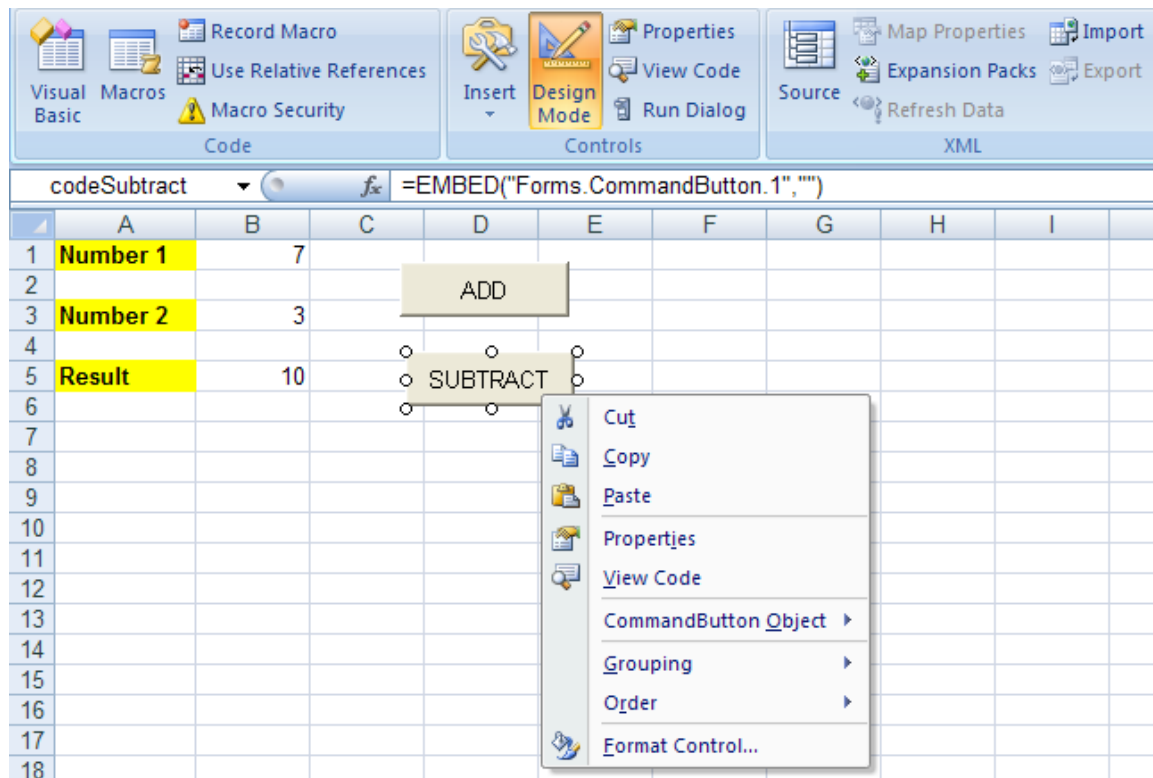
The VB code for the ADD and SUBTRACT buttons are:

```
Option Explicit
Option Base 1
Private Sub codeAdd_Click()
Dim a As Single, b As Single, c As Single
a = Cells(1, 2)
b = Cells(3, 2)
c = a + b
Cells(5, 2) = c
End Sub

Private Sub codeSubtract_Click()
Dim a As Single, b As Single, c As Single
a = Cells(1, 2)
```

```
b = Cells(3, 2)
c = a - b
Cells(5, 2) = c
End Sub
```

We will use EXCEL cells to input the values and return the values right back to one of the cells. The command "Cells(row,col)" allows us to access cell values. Here we placed the command buttons ADD and SUBTRACT directly on the spreadsheet. This can be done by clicking on the "Insert" option under the Developer tab (if you don't see the developer tab, then go to the lab page of exercise which shows how to activate this tab) and then click on the command button option under ActiveX Control. Later click within the spreadsheet to create the button. Right click on the button and select "Properties" (or directly click on "Properties" next to the Design Mode button on top of EXCEL ribbon) and set the properties name to "codeAdd" and caption to "ADD." Double click on the newly created command button and input the subroutine given above. Note, when you double clicked-- the system automatically created a subroutine codeAdd_Click(). The name "codeAdd" is the name of the button you defined under the properties menu. Repeat the steps for the subtract button. Once you are done, click on the "Design Mode" under Developer tab, this will allow you to switch to the RUN CODE mode. Now if you click on any of the command button your code will execute. If you want to modify the code then click again on the "Design Mode", which will allow you to toggle between the design and run modes.

Test Problem-1:
Problem Statement: Design a VB program to compute the areas of a rectangle. Input the dimensions of length and width of the rectangle from a spreadsheet as shown below:

Area Calculator	
Length (m)	2.210
Width (m)	4.444
Area (m^2)	

Use EXCEL to create the above spreadsheet. If the decimal numbers are not displayed you can control the format by selecting the B2 to B4 cells and then choose Format, cells and you then select the display formats for three decimal places.

This can be done by clicking on the "Insert" option under the Developer tab and then click on the command button option under ActiveX Control. Right click on the button and set the properties name to "Mycode" and caption to "RUN MY CODE." Double click on the code and enter the following visual basic code:

```
Option Explicit
Option Base 1
Private Sub Mycode_Click()
Dim Length As Single, Width As Single, Area As Single
Length = Cells(2, 2)
Width = Cells(3, 2)
Area = Length * Width
Cells(4, 2) = Area
End Sub
```

Visual basic tutorial problems, developed by Dr. Clement, clement@auburn.edu

Now you are ready to run the code. Click on the "Exit design mode" option and then click on your button to run the code. You should see the results below. Vary the input and try to run the code for other values.

Area Calculator	
Length (m)	2.210
Width (m)	4.444
Area (m^2)	9.821

Congratulations! You have successfully run your first visual basic program. Now let us look into the program and learn exactly what happened!

Option Explicit. In this class, you should use this standard text in all your programs. If your program does not have this statement you will loose points, even if your code runs! This indicates that all the variables in your program will be defined by the programmer. This is an important statement and you should include this in all your codes. You have this all the time. You can force EXCEL to insert this automatically, by entering the VBE menu selection Tools, options, editor tab and check "Require Variable Declaration."

Option Base 1. This allows you to start matrices or arrays from Row-1 as opposed to Row-zero. You should use this option in all your program

Private Sub Mycode_Click() (This is a standard line to indicate a sub routine that should be run when you click the button named "Mycode")

`Dim Length As Single, Width As Single, Area As Single`
Here you are creating three variables with names "Length", "Width" and "Area" that can store any real number with single precision. This holds signed IEEE 32-bit (4-byte) single-precision floating-point numbers ranging in value from -3.4028235E+38 through -1.401298E-45 for negative values and from 1.401298E-45 through 3.4028235E+38 for positive values. Single-precision numbers store an approximation of a real number.

Note: Another option to store real numbers in the variable "Area" is to use "Area as Double. " This will allow you to hold signed IEEE 64-bit (8-byte) double-precision floating-point numbers ranging in value from -1.79769313486231570E+308 through -4.94065645841246544E-324 for negative values and from 4.94065645841246544E-324 through 1.79769313486231570E+308 for positive values. Double-precision numbers store an higher level of approximation of a real number.

`Length = Cells(2, 2)`
`Width = Cells(3, 2)`
Here you transfer (or assign) the values you entered in your EXCEL spreadsheet cells (2,2) and (3,2) to the program variables Length and Width. In this operation the input data is following from the spreadsheet to the VB program.
Note: Some times it is easy to view your spreadsheet in RICI reference style when you use cells command to transfer data back and forth with the spreadsheet. To set RICI format, do the following:
To turn R1C1 (means row-1 and column-1) reference style on or off
   1. Click Options on the Tools menu, and then click the General tab.
   2. Under Settings, select or clear the R1C1 reference style check box.
You can go back to the standard format by removing the check mark in this box.

Area = Length * Width
This is the core computational step where your program calculates the area using the standard formal for computing areas and transfers (or assigns) the computed area to the variable named "Area."

Cells(4, 2) = Area
The above operation is similar to the earlier cell operation, expect the data flows in a reverse direction (from program to spreadsheet).   The computed value of area is transferred to your EXCEL spreadsheet cells (4,2).

Test Problem-2: Understanding simple IF-THEN-ELSE condition

Problem Statement*: Develop an advanced version of area calculator that will compute the area (as described above) and in addition it will also let the user know whether the object he is dealing with is a rectangle or a square.

Area Calculator-Ver2	
Length (m)	2.100
Width (m)	4.000
Area (m^2)	
Object Type	

Set up a command button and enter the following code:

```
Option Explicit
Option Base 1

Private Sub Mycode_Click()
Dim Length As Single, Width As Single, Area As Single
Length = Cells(2, 2)
Width = Cells(3, 2)
Area = Length * Width
Cells(4, 2) = Area
If Length = Width Then
    Cells(5, 2) = "Square"
Else
    Cells(5, 2) = "Rectangle"
End If
End Sub
```

Exit the design mode and run the code. You will see that your spreadsheet will be updated with the data shown below:

Area Calculator-Ver2	
Length (m)	2.100
Width (m)	4.000
Area (m^2)	8.400
Object Type	Rectangle

Also,

Area Calculator-Ver2	
Length (m)	2.100
Width (m)	2.100
Area (m^2)	4.410
Object Type	Square

This program is almost identical to the previous program, except you used if-then-endif structure to make a decision on the type of object you are dealing with it.

Note: It is always important to use proper indentation when you use IF and For-next conditions. It is easy to use Tab for this purpose. However, the default for tab is 4 spaces, but you can change this to two spaces by using Tools, options, Editor tab and change Tab Width to 2.

Test Problem-3: Understanding simple For-Next Loop
Problem Statement: Design a VB program to compute the areas of "n" number of squares and circles, given their characteristic dimensions (length of the side for squares or radius for circles). The value of "n" and the dimensions are entered in the spreadsheet as show below:

Number of squares/circles		
6		
Dimension	Square_area	Circle_area
1		
2		
3		
4		
5		
6		

Use EXCEL to create the above spreadsheet.
Setup a command button and enter the following code:

```
Option Explicit
Option Base 1

Private Sub Mycode_Click()
Dim i As Integer, n As Integer
Dim A As Single, result1 As Single, result2 As Single
n = Cells(2, 1)
    For i = 1 To n Step 1
        A = Cells(3 + i, 1)
        result1 = A ^ 2
        result2 = 3.14 * A ^ 2
        Cells(3 + i, 2) = result1
        Cells(3 + i, 3) = result2
    Next i
End Sub
```

Run the code and you will see that your spreadsheet will be updated with the data shown below:

Number of squares/circles		
6		
Dimension	Square_area	Circle_area
1	1	3.14
2	4	12.56
3	9	28.26
4	16	50.24
5	25	78.5
6	36	113.04

Now let us look into the program and learn exactly what happened!

Dim i As Integer, n As Integer

7

Here you declare variables i and n which are integers (in VB you have to define all the variables and also declare what type they are. **In this class, you should always use loop index or counters (e.g., i,j,k, etc) as integers (you will loose points otherwise even if your code works!).** Also, if you know certain variable will always be used to store integer numbers then declare them as integers. In this example, "n" has to be an integer since there is will be only integer value for the number of circles or square, it can never be a real number like 6.5 circles! Also, note declaring a variable as an integer allows to hold signed 32-bit (4-byte) integers ranging in value from -2,147,483,648 through 2,147,483,647.

Dim A As Single, result1 As Single, result2 As Single
Here you declare variables A, result1, and result2 as dynamically single precision real numbers. You can also declare them as double, if you want to store high precision real numbers.

n = Cells(2, 1)
Here you transfer the value of n that you have entered in your spread sheet at location (2,1, which is row 2 and column 1) into the integer variable called "n." Note, it is logical to declare n as an integer since it will always take an integer value (there will only be 3 or 4 circles not 3.2 circles!).

```
  For i = 1 To n Step 1
    A = Cells(3 + i, 1)
    result1 = A ^ 2
    result2 = 3.14 * A ^ 2
    Cells(3 + i, 2) = result1
    Cells(3 + i, 3) = result2
  Next i
```

The above code segment first initializes a loop using the statement "For i = 1 to n Step 1." This instructs the computer to begin a loop and use the counter "i" which will be incremented by 1 every time the loop is executed. The increment will continue until the value of n is exceeded.

The statement "*A = Cells(3 + i, 1)"* will transfer the dimension value in the spreadsheet cell at row number 3+i and column number 1 to the variable A.

The variable result1 and result2 will store the computed values of the areas of square and circle, respectively.

The last two statement transfers the computed areas to appropriate cells location in spreadsheet.

Test Problem-4: Understanding Loops and If-Elseif-Else-EndIf conditions
Develop a computer program to assign final letter grades in a class of ten students. The grade points are entered in the spreadsheet as below, and the code should be run to generate all the letter grades.

Points	Grade
92	
84	
77	
91	
43	
65	
71	
56	
69	
80	

```
Option Explicit
Option Base 1

Private Sub Mycode_Click()
Dim grade(10) As String
Dim Points(10) As Single
Dim i As Integer, n As Integer

n = 10

For i = 1 To n Step 1
    Points(i) = Cells(1 + i, 1)
Next i

For i = 1 To n
    If Points(i) >= 90 Then
       grade(i) = "A"
    ElseIf Points(i) >= 80 Then
       grade(i) = "B"
    ElseIf Points(i) >= 70 Then
        grade(i) = "C"
    ElseIf Points(i) >= 60 Then
        grade(i) = "D"
    Else
        grade(i) = "F"
    End If
Next i

For i = 1 To n
    Cells(1 + i, 2) = grade(i)
Next i
End Sub
```

Now run the code to see results.

Visual basic tutorial problems, developed by Dr. Clement, clement@auburn.edu

Test problem-5: Understanding accumulators used in a For-Next loop
Problem Statement: Develop a VB program that computes the average salary the staff in small consulting firm. The salary data is setup in spreadsheet as shown below:

Number of staff	7
Name	**Salary**
John	$100,000.00
Jane	$150,000.00
Jacob	$90,000.00
Josh	$25,000.00
Jay	$50,000.00
Jack	$75,000.00
Joy	$89,000.00
Average	

Note: in order display dollar sign you need to select that column and set the format using format, cells currency.   Set up a command button and enter the following code:

```
Option Explicit
Option Base 1
Private Sub Mycode_Click()
Dim Sum As Single, Average As Single
Dim i As Integer, n As Integer, Salary(7)as single
n = Cells(1, 2)
For i = 1 To n
    Salary(i) = Cells(2 + i, 2)
Next i
Sum = 0#
For i = 1 To n
    Sum = Sum + Salary(i)
Next i
Average = Sum / n
Cells(n + 4, 2) = Average
End Sub
```

Here we are initializing the variable sum as zero

Run the code and you will see that your spreadsheet will be updated with the data shown below:

Number of staff	7
Name	**Salary**
John	$100,000.00
Jane	$150,000.00
Jacob	$90,000.00
Josh	$25,000.00
Jay	$50,000.00
Jack	$75,000.00
Joy	$89,000.00
Average	$82,714.29

Note in the above code we have used the variable "sum" as an accumulator. Note when i=1, sum=0, then it is assigned a value of sum+100,000, which is 100,000. When i=2 sum = 100,000 and then it is assigned a value of 100,000+150,000.. this operation is repeated until all salaries of all the employees are added.

Test Problem-6: Understanding simple one dimensional arrays
Problem Statement: Design a VB program to compute the areas of "n" number of squares and circles, given their characteristic dimensions (length of the side for squares or radius for circles). The value of "n" and the dimensions are entered in the spreadsheet as show below:

Number of squares/circles		
6		
Dimension	Square_area	Circle_area
1		
2		
3		
4		
5		
6		

Use EXCEL to create the above spreadsheet.
Setup a command button and enter the following code:

```
Option Explicit
Option Base 1

Private Sub Mycode_Click()
Dim i As Integer, n As Integer, A As Single
Dim result1() As Single, result2() As Single
n = Cells(2, 1)
ReDim result1(n), result2(n)
For i = 1 To n Step 1
    A = Cells(3 + i, 1)
    result1(i) = A ^ 2
    result2(i) = 3.14 * A ^ 2
Next i
For i = 1 To n Step 1
    Cells(3 + i, 2) = result1(i)
    Cells(3 + i, 3) = result2(i)
Next i
End Sub
```

Run the code and you will see that your spreadsheet will be updated with the data shown below:

Number of squares/circles		
6		
Dimension	Square_area	Circle_area
1	1	3.14
2	4	12.56
3	9	28.26
4	16	50.24
5	25	78.5
6	36	113.04

This example is identical to problem-4, except we used one-dimensional arrays results1() and results2() to first store the values internally and then we output the results after finishing all the computations.

Test Problem-7: Working with one-dimensional array variables and multiple subroutines
Objectives: Design a VB program to compute the areas of "n" number of squares and circles, given their characteristic dimensions (length of the side for squares or radius for circles). Use multiple subroutines and array variable to complete all the computations and then output the computed values. The value of "n" and the dimensions are entered in the spreadsheet as show below:

Number of squares/circles		
6		
Dimension	Square_area	Circle_area
1		
2		
3		
4		
5		
6		

Use EXCEL to create the above spreadsheet.

Set up a command button to enter the following code:

```
Option Explicit
Option Base 1
Private Sub Mycode_Click()
Dim i As Integer, n As Integer
Dim A() As Single, result1() As Single, result2() As Single
n = Cells(2, 1)
ReDim A(n) As Single, result1(n) As Single, result2(n) As Single
    For i = 1 To n Step 1
        A(i) = Cells(3 + i, 1)
    Next i
    Call Square(n, A(), result1())
    Call Circ(n, A(), result2())
    For i = 1 To n Step 1
        Cells(3 + i, 2) = result1(i)
        Cells(3 + i, 3) = result2(i)
    Next i
End Sub

Sub Square(n as integer, AA() as single, result() as single)
Dim i As Integer
    For i = 1 To n
        result(i) = AA(i) ^ 2
    Next i
End Sub

Sub Circ(n as integer, AB() as single, result() as single)
Dim i As Integer
```

```
     For i = 1 To n
         result(i) = 3.14 * AB(i) ^ 2
     Next i
End Sub
```

Now run the code and you will see that your spreadsheet will be updated with the data shown below:

Number of squares/circles		
6		
Dimension	Square_area	Circle_area
1	1	3.14
2	4	12.56
3	9	28.26
4	16	50.24
5	25	78.5
6	36	113.04

Now let us look into the program and learn exactly what we did!
Option Base 1
This allows you to start 1 as the base for any array. For example if you define Amat as a 3 x 2 array the first entry in the array will be Amat(1,1). If we do not use this statement then there will entries Amat(0,0), (0,1) and Amat(0,2).. while the presence of empty values at these spots may not affect calculations, it can be confusing during the debugging process. Use of option base 1 allows to start a 1-D array at A(1), 2-D array at A(1,1), and 3-D array at A (1,1,1).. so on.

Dim i As Integer, n As Integer
Here you declare variables i and n which are integers (in VB you have to define all the variables and also declare what type they are. It is a good practice to use loop index or counters (e.g., i,j,k, etc)_ as integers. Also, if you know certain variable will always be used to store integer numbers then declare them as integers.

Dim A() As Single, result1() As Single, result2() As Single
Here you declare variables A, result1, and result2 as dynamically allocatable single precision real number arrays. You can also declare them as double, if you want to store high precision real numbers.

n = Cells(2, 1)
Here you transfer the value of n that you have entered in your spread sheet at location (2,1, which is row 2 and column 1) into the integer variable called "n." Note, it is logical to declare n as an integer since it will always take an integer value (there will only be 3 or 4 circles not 3.2 circles!).

ReDim A(n) As Single, result1(n) As Single, result2(n) As Single
Here you allocate the exact size of arrays A, result1, and result2. The computer will not set aside certain amount of memory to store values in these arrays.

For i = 1 To n Step 1
* A(i) = Cells(3 + i, 1)*
Next i

Here you run a FOR loop to read the characteristic dimensions of the circles/squares entered in your spreadsheet. Note the loop index "i" is used point the row and column number of the cells in the spreadsheet.

> ***Call Square(n, A(), result1())***
> ***Call Circ(n, A(), result2())***

Here you call the subroutines to compute the area of squares and circles. Note you send the values of n and A into these routines and the routine computes the result and sends it back to your main program. It is good to use a convention where all the input variables are listed first followed by the list the output variables. Also, parentheses are placed next to the array variables to indicate to VB that it passing array variables. Notice no parentheses are used in the receiving subroutine.

> ***For i = 1 To n Step 1***
> > ***Cells(3 + i, 2) = result1(i)***
> > ***Cells(3 + i, 3) = result2(i)***
> ***Next i***
> ***End Sub***

The loop above allows you to output the computed areas back to appropriate cell locations within your spreadsheet

Sub Square(n as integer, AA() as single, result() as single)
Dim i As Integer
> ***For i = 1 To n***
> > ***result(i) = AA(i) ^ 2***
> ***Next i***
End Sub

The code above is a subroutine (similar to MATLAB functions) that computes the area of "n" number of squares. The results are stored in the variable "result." This routine is invoked when "Call Square(n, A, result1)" is executed in the main program. Note, the value of n is transferred into the subroutine as n, value of A is transferred (or mapped) as AA, the values in the variable "result" computed within the routine is transferred (or mapped) back to the main program as a variable "result1." Visual basic, by default will pass the variable values into a subroutine "***by reference.***" Another option to pass values (not discussed in this class) is "***by value.***" When the variable names are passed by reference, the computer will use the original memory locations that have already been setup for the variables in the calling routine. The mapped variables simply refer to the same memory location, even if their names are different. One possible limitation of this strategy is that the local routine may inadvertently change the value of the variable that is not supposed to be changed. For example, if the value of n is set to, say 100, either in the routine circle or square then the value of n will become 100 in the main routine too. This type of situations can lead to serious run time bugs that are difficult to track. To avoid such problems, passing "by value" method should be used (however, we will not cover this topic in this class).

Sub Circ(n as integer, AB() as single, result() as single)
Dim i As Integer
> ***For i = 1 To n***
> > ***result(i) = 3.14 * AB(i) ^ 2***
> ***Next i***
End Sub

The code above is the second subroutine (similar to the one discussed above). This routine computes the area of "n" number of circles. The results are stored in the variable "result"

Test Problem-8: Working with multi-dimensional arrays
Objectives: Design a VB program that can multiple a n × n matrix with a n × 1 matrix.

First setup the following spreadsheet to define input to the program

Dimension	3			Mat B		Mat C
Mat A				**Mat B**		**Mat C**
1	2	3		10		
4	5	6		20		
7	8	9		30		

Note, between matrix A and B we leave a blank column, also between Matrix B and C we leave a blank column.

Set up a command button and enter the following code:

```
Option Explicit
Option Base 1

Private Sub Mycode_Click()
Dim i As Integer, j As Integer, n As Integer
Dim A() As Single, B() As Single, C() As Single
n = Cells(1, 2)
ReDim A(n, n) As Single, B(n) As Single, C(n) As Single

'Read Matrix A from spreadsheet
For i = 1 To n
    For j = 1 To n
        A(i, j) = Cells(2 + i, j)
    Next j
Next i

'Read Matrix B from spreadsheet
For i = 1 To n
    B(i) = Cells(2 + i, n + 2)
Next i

'Perform matrix multiplication to compute C
For i = 1 To n
    C(i) = 0#
    For j = 1 To n
        C(i) = C(i) + A(i, j) * B(j)
    Next j
Next i

'Output the results to spreadsheet
For i = 1 To n
    Cells(2 + i, n + 4) = C(i)
Next i

End Sub
```

Visual basic tutorial problems, developed by Dr. Clement, clement@auburn.edu

Run the code to get the following results:

Dimension	3			Mat B		Mat C
Mat A				Mat B		Mat C
1	2	3		10		140
4	5	6		20		320
7	8	9		30		500

Now let us review the code block-by-block

```
Dim A() As Single, B() As Single, C() As Single
n = Cells(1, 2)
ReDim A(n, n) As Single, B(n) As Single, C(n) As Single
```

In the above three lines, we use a powerful dynamic array allocation option to define the sized of the array. The first line simply tells the code that A, B, and C are single precision arrays. However, it does not say anything about the size or shape (dimensionality) of the arrays. In the second line we read n from the spreadsheet. In the third line, we use the value of n to define the size and the number of dimensions (also know as shape) of the array.

```
'Read Matrix A from spreadsheet
For i = 1 To n
    For j = 1 To n
        A(i, j) = Cells(2 + i, j)
    Next j
Next i
```

Note, 2 is used to skip the first two rows

In the above block we read the values of the elements in array A from the spreadsheet.

```
'Read Matrix B from spreadsheet
For i = 1 To n
    B(i) = Cells(2 + i, n + 2)
Next i
```

In the above block we read the value B.

n+2 indicates the location of the B entries

```
For i = 1 To n
    C(i) = 0#
    For j = 1 To n
        C(i) = C(i) + A(i, j) * B(j)
    Next j
Next i
```

Above code segment performs the matrix multiplication operation, which identical to what we did when we performed hand calculations.

n+4 indicates the location of the C entries

```
For i = 1 To n
    Cells(2 + i, n + 4) = C(i)
Next i
```

Above code segment outputs the data back to the spreadsheet.

Visual basic tutorial problems, developed by Dr. Clement, clement@auburn.edu

To learn more about VB, visit this nice website:
http://www.engineering.usu.edu/cee/faculty/gurro/Software_Calculators/ExcelVBA/ExcelVBAExamples.htm
Some useful information I got from this site are:
1) To get numerical data from the worksheet into your VBA code use functions *Range* or *Cells*. For example, to get the value of cell "B2" into variable *a* in your VBA code use:

a = Range("B2").Value   or   Alternatively, you could use,   a = Cells(2,2).Value
to load your data value.

To place data from your VBA code to the worksheet use the same functions *Range* and *Cells*, e.g.,

Range("C2").Value = r1 or Cells(3,2).Value = r1

To place string data from your VBA code to the worksheet use, for example:

Range("M2") = "Int. = " or   Cells(15,2) = "Int. = "

Notice that function *Range* takes as argument a string referring to the cell of interest, e.g., Range("F15"), while the function *Cells* takes two arguments, namely, the indices for the row and column of the cell of interest, e.g., *Cells(25,31)*.
To clear a value in cell D8, use the commands
Range("D8").Select
Selection.ClearContents

To clear a rectangular region of cells
   Range("D10:F12").Select
   Selection.ClearContents

Excel provides a number of *functions that are not available in Visual Basic*, e.g., *ACOS, ASIN, BESSELI,* etc. You can use the cells in the worksheet to evaluate formulas that use these functions and then bring those values into your VBA code.

To see a *list of the functions available in Excel*, click on a cell in a worksheet, place and equal sign in it and then click on the button labeled *fx* in the Excel toolbar. A listing of the functions by categories will be provided. The categories include *All*, *Date & Time*, *Math & Trig*, *Statistical,* etc. Click on any category to get a listing of functions available. [Functions list window]

There is a category of functions referred to as *Engineering functions* that include advanced engineering mathematical functions, e.g., *Bessel* functions, *complex* number conversion, *error function*, etc. If that category is not available in your Excel functions, use the option *Tools>Add Ins...* and mark the boxes *Analysis Toolpack* and *Analysis Toolpack - VBA* to add the engineering functions. Press *OK* to load the functions.