Lab-7
Arrays Test Problem-1: 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:

<table>
<thead>
<tr>
<th>Number of squares/circles</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>Square_area</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Number of squares/circles</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>Square_area</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
</tr>
</tbody>
</table>
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.

**Arrays Test -2: 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:

<table>
<thead>
<tr>
<th>Number of squares/circles</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dimension</strong></td>
<td><strong>Square_area</strong></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Use EXCEL to create the above spreadsheet.

Set up a command button to enter the following code:

```vbnet
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
```
Next i
End Sub

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

<table>
<thead>
<tr>
<th>Number of squares/circles</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>Square_area</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
</tr>
</tbody>
</table>

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”
Arrays Test Problem-3: Working with multi-dimensional arrays

Objectives: Design a VB program that can multiply an $n \times n$ matrix with an $n \times 1$ matrix.

First setup the following spreadsheet to define input to the program

<table>
<thead>
<tr>
<th>Dimension</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 2 3</td>
</tr>
<tr>
<td></td>
<td>4 5 6</td>
</tr>
<tr>
<td></td>
<td>7 8 9</td>
</tr>
<tr>
<td>Mat B</td>
<td>10</td>
</tr>
<tr>
<td>Mat C</td>
<td></td>
</tr>
</tbody>
</table>

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:

```vbnet
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
```
Run the code to get the following results:

<table>
<thead>
<tr>
<th>Dimension</th>
<th>3</th>
<th>Mat B</th>
<th>Mat C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>500</td>
<td></td>
</tr>
</tbody>
</table>

Now let us review the code block-by-block

```vba
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.

```vba
' 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
```

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

```vba
' 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.

```vba
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.

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

Above code segment outputs the data back to the spreadsheet.
Arrays Test Problem-4: Sorting grades in an arrays
Develop a VB program which read 10 grade points into an array and sorts them from high to low.
First set up the spreadsheet as follows:

<table>
<thead>
<tr>
<th>Points</th>
<th>Sorted Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

Option Explicit
Option Base 1
Private Sub doSort_Click()
    Dim i As Integer, n As Integer, j As Integer
    Dim grade() As Double, temp As Double
    n = 10
    ReDim grade(n) As Double
    For i = 1 To n
        grade(i) = Cells(1 + i, 1)
    Next i
    For i = 1 To n - 1
        For j = i + 1 To n
            If (grade(j) > grade(i)) Then
                temp = grade(j)
                grade(j) = grade(i)
                grade(i) = temp
            End If
        Next j
    Next i
    For i = 1 To n
        Cells(1 + i, 2) = grade(i)
    Next i
End Sub