Lab 05: "2DArrays & Arrays of Pointers to Functions"

Due: **07:59 AM** on **Friday, Mar 03**
Points: **100** points (50 points each part).

Project 01: **2D Arrays Processing**

Objective:

- To design and implement functions to process 2DArrays.

Instructions:

- Be sure to document your code (add comments on top of each function).
- In the comments add your name, date, course, homework number, and statement of problem.
- Once you are done, upload your final solution through Blackboard.
- No need for input validation.
- Write a project called **`Arrays2D.c`**

Steps:

- It is up to you to choose what the return type should be. Be creative!
- Functions should not have pointers implementations.
- **`Arrays2D.c`** has the following functions

Part01 [50 points]: (2-Dimensional Array Functions)

1. **[10 points]** Write a function called `max` that returns the maximum value in the 2d array.
2. **[10 points]** Write a function called `rowSum` returns the sum of the elements in Row x of the 2d array.
3. **[10 points]** Write a function called `columnSum` returns the sum of the elements in Column x of the 2d array.
4. **[10 points]** Write a function called `isSquare` that checks if the array is square (i.e. every row has the same length as the 2d array itself).
5. **[10 points]** Write a function called `displayOutputs` that displays the 2 dim-array elements.

Part02 (Testing main) `Arrays2D`

- First declare a 2-dim array. How to do that? You need to read the number of rows and the number of columns from the user, and then it reads a corresponding entries to that size. E.g., if a user enters 3 for the number of rows, and enters 3 for the number of columns, then we declare an array of 9 and then read 9 entries and store them in the array.
- Make calls to all functions in part 01 to match the sample demo below.
- Make sure you display the same exact messages.
- Use blank lines to separate outputs and make then more readable.

Sample run:

```
Let's create a 2Dim array!

      How many rows? 2
      How many columns? 3

      enter [0][0]: 11
      enter [0][1]: 22
      enter [0][2]: 33
      enter [1][0]: 44
      enter [1][1]: 55
      enter [1][2]: 66

Sum of row 1 = 66
Sum of row 2 = 165

Sum of column 1 = 55
Sum of column 2 = 77
Sum of column 3 = 99

This is not a square array.

Here is your 2Dim array:
[11, 22, 33]
[44, 55, 66]
```

```
Let's create a 2Dim array!

      How many rows? 3
      How many columns? 3

      enter [0][0]: 10
      enter [0][1]: 20
      enter [0][2]: 30
      enter [1][0]: 40
      enter [1][1]: 50
      enter [1][2]: 60
      enter [2][0]: 70
      enter [2][1]: 80
      enter [2][2]: 90

Sum of row 1 = 60
Sum of row 2 = 150
Sum of row 3 = 240

Sum of column 1 = 120
Sum of column 2 = 150
Sum of column 3 = 180

This is a square array.

Here is your 2Dim array:
[10, 20, 30]
[40, 50, 60]
[70, 80, 90]
```

Project 02: **Arrays of Pointers to Functions**

Objective:

- To design and implement arrays of pointers to functions.

Instructions:

- Be sure to document your code (add comments on top of each function).
- In the comments add your name, date, course, homework number, and statement of problem.
- Once you are done, upload your final solution through Blackboard.
- No need for input validation.
- Write a project called **`ArraysofPtrs.c`**

Steps:

- Rewrite the program of Fig. 6.22 from Chapter 06 (C Arrays) to use a menu-driven interface. The Program should offer the user four options as follows:

```
Enter a choice:
      0  Print the array of grades
      1  Find the minimum grade
      2  Find the maximum grade
      3  Print the average on all tests for each student
      4  End Program
```

Part01 [50 points]: (2-Dimensional Array Functions)

1. One restriction on using arrays of pointers to functions is that all the pointers must have the same type. The pointers must be to functions of the same return type that receive arguments of the same type.
2. **[10 points]** For this reason, the functions in Fig. 6.22 must be modified so that they each return the same type and take the same parameters.
3. **[20 points]** Modify functions `minimum` and `maximum` to print the minimum or maximum value and return nothing. For option 3, modify function `average` of Fig. 6.22 t output the average of each student (not a specific student).
4. **[10 points]** Function `average` should return nothing and take the same parameters as `printArray`, `minimum`, and `maximum`.
5. **[10 points]** Store the pointers to the four functions in array `processGrades` and use the choice made by the user as the index into the array for calling each function.

Part02 (Testing main) `ArraysofPtrs`

- Here, as long as you display the same exact message seen above, then you are free to do your own testing.