CSE 1321L: Programming and Problem Solving I Lab
Lab 8
1 and 2 Dimensional Arrays

What students will learn:

- Using the debugger
- Basic operations with 1D and 2D arrays

Overview: In this lab, you're going to do two things - learn how to use the debugger and focus on 1D and 2D arrays.

Debugger

If you've had a problem with your code in the past, you might have included print statements to better understand the program's behavior. Depending on the situation, this technique could work. However, when things get complex, it's better to use the debugger.

A debugger is a built-in tool in most IDEs that enables you trace (i.e., "walk through") your code and see the values of variables. Both IntelliJ and Visual Studio have a built-in debugger. It requires you to run the code in a different "mode" called Debug Mode.

There are a few basic concepts about debuggers that you need to know:

- You need to set at least one breakpoint – which is where the debugger will pause your code and allow you to see the state of your variables. You set a breakpoint by clicking in the column to the left of the code. You will see a red dot appear, meaning that the breakpoint is set.
- When you start your program in debug mode, the program will pause where the breakpoint is.
- You can also stop debugging – even in the middle of debugging. You can fix your code then start debugging again.
- Finally, you walk through the code by using either Step Into or Step Over. Every time you do this, you advance through the code – looking at the state of variables along the way.

Before starting this lab, we recommend watching the video on Using the Debugger in IntelliJ (https://www.youtube.com/watch?v=Po3POCfBJPw) or using the Debugger in Visual Studio (https://www.youtube.com/watch?v=MLHxDMZIt5M).

1D Arrays

Arrays are a complex data type which can be used to store information of the same data type. For example, you may fill an array with integers. In order to do this you must first declare the array, this is similar to declaring other variables you have used in the past:

```
int [] example;
```

In the example above, `int` is our data type, `[]` represents it is going to be an array, and `example` is the name of the array. Next we need to give a little more information about the size of the array. This can be done a couple of different ways, but normally it will look something like this:

```
example = new int [5];
```

Now we stating that example is going to be a new array full of integers that has 5 elements. When dealing with arrays we focus on 2 parts, elements and index. Elements are any data stored inside a portion of the array. The index is a numbering system which keeps track of the elements inside of the array.

Index	0	1	2	3	4
Element					

Above is a graphical example of the array we previously created. It is currently 5 elements long and has no information stored in each index. It is very important to notice that an index value will always start at 0, so the size may be 5 but the highest index you can access or use will be 4.

Once we have created an array with an appropriate size, we cannot change the size. Arrays are immutable which means the memory allocated to each array cannot be changed once is has been created. Many times, this will lead to creating a new array if you need to add additional elements to the existing array.

In order to modify information at a particular index in the array you will need to specify which index you want to change. For example if I wanted to add data to index 3, it would look like this:

```
example [3] = 42;
```

Index	0	1	2	3	4
Element				42	

As you can see, we have changed the value of the element at index 3 to hold 42. This process would take some time and many lines of code if we had to fill an array with 100 indices. However, this process commonly uses a for loop to iterate through the array. Because you can initialize a variable in the loop to 0 (think int i = 0) this variable is commonly used in place of an index. See the example below for some idea of how to fill an array of values. Just remember, this line would be inside of a loop that would terminate at the final index.

```
example[i] = 42;
```

Index	0	1	2	3	4
Element	42	42	42	42	42

The area most people run into issues is when to terminate the loop. Remember, the final index of an array is 1 less than the size of the array.

Lab8A:
For this exercise you will be creating 2 different arrays. Both should store integers and be able to hold 5 elements. The user should be able to fill both up with different values. Once the arrays have been created and data is added to each array, you should add the elements at each index array 1 to the element at each index from the array 2. Please store the addition of each index in a new array and print this array out like shown below.

As always:

- Remember, the class name should be Lab8A.
- The user input is indicated in bold.

Sample Output #1:

```
Please enter 5 integers for the first array:
Integer 1: 4
Integer 2: 10
Integer 3: 50
Integer 4: 21
Integer 5: 5

Please enter 5 integers for the second array:
Integer 1: 5
Integer 2: 1
Integer 3: 16
Integer 4: 4
Integer 5: 6

The resulting sums are 9|11|66|25|11|
```

2D Arrays

Now we will move on from 1-D arrays onto 2-D arrays; please keep in mind that while with 1-D arrays we used one for-loop to traverse the array, with 2-D arrays we need to use a nested for-loop to traverse it. Ideally, a rule of thumb is that the outer loop (the "i" loop) would be representative of the row number, and the inner loop (the "j" loop) would be representative of the column number. Together the row number and column number decide what cell is being looked at.

At the end of this lab, we've provided some additional information to assist you with iterating through 2D arrays. Additionally, you should watch the videos posted in the Module 5 section of D2L. This is especially important for **C++ students**, as creating dynamically allocated arrays (which you'll use in the next exercises) require a little more work in that language than Java or C#.

Lab8B: N by N

Create a simple program that asks the user for a row and column size for a 2-D integer array, then have The program create such an array. At this point this array's cells should be filled with zeros since nothing has been done to change the cell values.

Next, have that program dynamically adapt to the sizes of the array (that were provided by the user) and fill the array with numbers ascending from 1 → n (n being the number of the nth cell in that 2-D Array depending on the size). Please then print the array out. Remember, you cannot just print an array out, you have to use loops and print each cell individually.

Please refer to the sample output below for visualization and match its style.

As always:

- Remember, the class name should be Lab8B.
- The user input is indicated in bold.

Sample Output #1:

```
Please enter the number of rows: 4
Please enter the number of columns: 4

I have 4 rows and 4 columns. I need to fill-up 16 spaces.

The 4x4 array:
1|2|3|4|
5|6|7|8|
9|10|11|12|
13|14|15|16|
```

Sample Output #2:

```
Please enter the number of rows: 3
Please enter the number of columns: 2

I have 3 rows and 2 columns. I need to fill-up 6 spaces.

The 3x2 array:
1|2|
3|4|
5|6|
```

Lab8C: 2D or not 2D

For this exercise you will be writing a program that will build upon Lab8B. So please make a new class file, copy, and rename the code you have already written for Lab8B. Please also remember to rename the file and (and in the case of Java and C# students) class name to Lab8C.

For this exercise, please take the array that was filled up with values and flatten it i.e., convert the 2-D array to a 1-D array that contains the same values. Please do not just print the 2-D array to look like a 1-D array

Hint: You will have to take the 2-D array's total number of cells and make a 1-D array of the same size. Please refer to the sample output below for visualization and match its style.

As always:

- Remember, the class name should be Lab8C.
- The user input is indicated in bold.

Sample Output #1:

```
Please enter the number of rows: 4
Please enter the number of columns: 4

I have 4 rows and 4 columns. I need to fill-up 16 spaces.

The 4x4 array:
1|2|3|4|
5|6|7|8|
9|10|11|12|
13|14|15|16|

The 4x4 2-D array flattened into a 16 cell 1-D array:
1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|
```

Sample Output #2:

```
Please enter the number of rows: 3
Please enter the number of columns: 2

I have 3 rows and 2 columns. I need to fill-up 6 spaces.

The 3x2 array:
1|2|
3|4|
5|6|

The 3x2 2-D array flattened into a 6 cell 1-D array:
1|2|3|4|5|6|
```

Instructions:

- Programs must be working correctly.
- Programs must be saved in files with the correct file name.
- If working in Java or C#, class names must be correct.
- Programs (only .java, .cs or .cpp files) must be uploaded to Gradescope by due date.

Appendix – Referencing row or column length

C# `int[,] grid = new int[4,5];`

To get the row length use arrayname.GetLength(0)
E.g.: `grid.GetLength(0)`

To get the column length use arrayname.GetLength(1)
E.g.: `grid.GetLength(1)`

Java:

```
int[][] grid = new int[4][5];
```

To get the row length use arrayname.length
E.g.: `grid.length`

To get the column length use arrayname[0].length
E.g.: `grid[0].length`

C++ `int grid[4][5];`