CS 211 – Data Structures
Lab for Week #08

For this lab exercise, you will work either in 2-person teams or individually. You are HIGHLY ENCOURAGED to work in 2-person teams! The point here is to have teams discuss the lab and offer each other support in making sure the IDE of your choice is working.

Lab Exercise

This lab is to provide some practice using a pre-defined **ArrayStack** class, which is a stack data structure implemented using an array, to perform stack-related actions, and to expand the class's capabilities by adding functionality.

A **stack** ADT is simpler than even a linked list, but is a very useful in many situations. The concept of how a stack works is relatively simple, as it acts similarly to a stack of physical objects.
Basic actions on a stack include:

PUSH, which adds a new item to the top of the stack
POP, which removes an item from the top of the stack

In this lab, you will take existing code for a stack ADT that is implemented using an array.

Into a C++ Application project, copy the contents of the following files:

ArrayStack.cpp **ArrayStack.h** **lab08Test.cpp**

Examine the contents of the files to determine the overall function of the **ArrayStack** class. Identify the methods for the **ArrayStack** class, the arguments passed to them, and any return values the methods pass back.

At the top of **each** file, document these files as follows based on your 2-person team:

```
# CS 211 Fall 2024 – Week 08 Lab
# <Student_Name1> and <Student_Name2>
```

Substitute your own names for the **<Student_Name>** placeholders above.

The **ArrayStack** class is currently defined to use a constant value named **CAPACITY** for the stack's fixed capacity. Your task is to update the **ArrayStack** class so that it can have a stack size that has an initial value of **10**, but that is not fixed, and can increase any time the demand requires the stack increase its capacity. There are two ways you can address this – one involves adapting the code to use C++ vectors instead of arrays, and the other is to just allocate a larger array (and copy all contents to it) whenever needed.
You'll need to address these issues (among others):

- Create an **int** variable to hold the current size of the stack, and place that in the **private** area of the stack, so that the stack will always know its current size.
- Your constructor(s) should dynamically allocate the array (or use a vector) that holds the stack.
- You may want a method to retrieve the current capacity of the stack.

- The **push** method should not return **false** when the array is full; it should instead either call a method that increases the size of the array used for the stack, then push the new value onto the stack; OR, if you choose to convert the code to using vectors, let the vector handle this automatically.
- Add a mutator to change the current size of the stack, which should create a **new** instance of an array of an increased size and copy all the contents of the stack to the new array, then **delete** the old array. I recommend increasing the size of the stack by 5 elements at a time. Alternately, let the vector library handle this action automatically. Do NOT let it shrink the stack!
- Don't forget that the curious syntax to deallocate an array is: **`delete []`** ***array_name*****;**
- **clearStack** should also reset the stack capacity back to its original value of **10**, with appropriate use of **delete** and **new** as needed.
- Some other methods may need their parameters and headers changed accordingly, and calls to those headers will need an updated set of arguments.
- Update the tests in **testArray.cpp** to properly exercise and test the updated class and its methods.

There are several different ways to approach this task, so there is no one "correct answer" here. This lab is meant to give you some latitude in how you approach this problem.

EXTRA CREDIT – 20% EXTRA

If you have additional time remaining, you can attempt to implement a method that will copy the entire contents of a stack by use of an assignment statement. If implemented correctly, it will copy the entirety of a stack with a statement like **stack2 = stack1;**

Do what you can, but beware – it's harder than it sounds! Partial extra credit will be generously awarded :-). Note that to get any extra credit, it must be submitted as a team by the end of the lab period!

ONCE FINISHED, ALL STUDENTS SHOULD SUBMIT ALL THE FILES VIA CANVAS.
You should send copies of CPP and H files to all team members for submission.
Submit ALL the CPP files and H files using the "multiple file submission" feature of Canvas.
Do NOT submit the CPP and the H files separately, and do not "zip" them!