Lab Exercise #1
Due Date: Feb 6th 2024
Stacks

Description:
For this lab exercise, you will implement a **Stack** whose size can grow as elements are inserted into the stack. You will **build two different implementations** of this Stack. Both these implementations will be array-based. These implementations will set an **initial capacity** for the **stack** in the constructor. The only difference between these implementations will be what happens when the stack is full. For the first implementation, **StackArrayDouble**, you should double the size of the array. For the second implementation, **StackArrayLinear**, you should increase the size of the array by a constant amount.

starter.zip (Starter Code):
The following is a brief description of the starter code provided to you:

- **AbstractStack.h**
  - This header file declares an Abstract Stack class which declares many of the common Stack methods such as **push**, **isEmpty**, **size**, **top**, **pop**. You will observe that all these methods are declared as pure virtual functions. You may also notice that both the **StackArrayDouble** and the **StackArrayLinear** classes only differ in their implementations with respect to the push function. As a result, you may wonder why all the other functions are not implemented in the Base AbstractStack class instead of declaring them as pure virtual. The reason is you will be integrating your implementations from this LE in PA1 in which you will be additionally tasked with coding a third implementation of a Stack using a Linked List (which will contain very different implementations for these functions).
  - **No changes to this file is necessary**
- **StackArrayDouble.h & StackArrayLinear.h**
  - These header files declare the StackArrayDouble and StackArrayLinear classes which are derived classes of the AbstractStack class. These files declare the different methods that StackArrayDouble and StackArrayLinear must implement (as a consequence of these methods being pure virtual in the base AbstractStack class).
  - Both StackArrayDouble & StackArrayLinear.h have three private member variables:
    - **T* arr:** This is the pointer which will be used to allocate memory on the heap
    - **int length:** This variable keeps track of the ***capacity*** of the member array
    - **int topIndex:** This is the variable that keeps track of the index of the top most element of the stack
- **main.cpp**
  - You can use this file to instantiate StackArrayDouble and StackArrayLinear objects and test your code.

Contract (TASK):
Your task is to implement the methods declared in the StackArrayDouble and StackArrayLinear classes. In StackArrayDouble.h and StackArrayLinear.h, you have been provided with non-functional definitions of these methods. **You must implement these method definitions in the given header files,** by replacing the current non-functional definitions with your functional implementations. **Do not create additional cpp files to define these functions, make sure that they are part of the header files.**

[Advisory]:

- You may choose **1** to be the initial capacity of the array when allocating memory in the constructor.
- You may choose **10** to be the constant amount with which the StackArrayLinear class grows in length each time it is full
- You must throw a **std::out_of_range** exception when you come across invalid calls to the top and pop methods

Deliverables and Submissions:

In a zip folder named using the format **<FirstName>-<LastName>-<UIN>-<LE1>.zip** you must submit the following file(s) to Canvas:

- **StackArrayDouble.h** (containing the functional method definitions of the StackArrayDouble class)
- **StackArrayLinear.h** (containing the functional method definitions of the StackArrayLinear class)

Do not use angular brackets in the name of the submission folder.

Grading:
This assignment is worth 20 points. Each method of the stack will be tested using an automatic testing infrastructure. You will be provided with the testing infrastructure which you can use to evaluate your implementations.

The grading rubric is as follows:

- Each of the following methods will be tested out for both the StackArrayDouble and StackArrayLinear class:
  - Push Method: 1 point
  - Pop Method: 3 points
  - Pop Exception Case: 1 point
  - Top Method: 1 point
  - Top Exception Case: 1 point
  - Empty Method: 1 point
  - Copy Assignment: 1 point
  - Copy Constructor: 1 point

You will notice that all the above methods sum up to 10 points. As they will be tested out twice (once each for the StackArrayDouble and

StackArrayLinear classes), they will total to 20 points. In case of any memory leaks in your program, we will subtract 2 points.

testInfrastructure.zip (Test Methodology):
In order to familiarize you with the auto-grading environment on our end we are providing you with an automated test infrastructure that can be used to check the correctness of your code. The testing infrastructure is constructed as follows:

- **stack_test.cpp:** This cpp file defines all the test functions that have been mentioned in the grading rubric above. It also runs these tests and outputs your final score.
- **test.py:** This file compiles stack_test.cpp in order to produce the executable. It then runs this executable which will indicate your score (as well as the test cases you have passed or failed). It will also run valgrind on the executable file to indicate the presence of memory leaks if they exist.

Note: There is a possibility that one or more of your methods causes a segmentation fault which in turn causes the test script to crash when calling that particular method. In such a case, we will not be able to calculate your score for the assignment and you will be assigned a default grade of 0. After the grades are posted, you will be given 1 week to rectify your error(s) and resubmit the assignment. You will be subject to a 25% penalty on the grade obtained from the resulting resubmission of the assignment.

How to test your program natively (run your own test cases):
You can use main.cpp provided in the starter.zip folder to instantiate StackArrayDouble and StackArrayLinear objects. You can then test the functionality of your methods by calling these methods and checking whether what they return is what you expect.
To compile your own tests in main.cpp, **you can use the makefile provided within starter.zip to compile your program.** In your terminal:

- Run **make** on the terminal (this will create an executable named main)
- Next, run **./main** on your terminal to run your native tests
- Finally run **valgrind --leak-check=full ./main** to check whether there are memory leaks.

How to use testInfrastructure:
To check your score in this LE using the testInfrastructure, you can follow either of the two options mentioned below:

- **Option 1:** Move your AbstractStack.h, StackArrayDouble.h, and StackArrayLinear.h into the testInfrastructure directory. After this, run **python3 test.py**
- **Option 2:** Move your AbstractStack.h, StackArrayDouble.h, and StackArrayLinear.h into the testInfrastructure directory. **Ensure that the makefile in this directory is the one given to you originally in the testInfrastructure.zip folder and not in the starter.zip folder.** In your terminal:
    - Run **make** in order to compile the program (which will compile smart_test.cpp and create an executable named stack_test).

- Next, run **./stack_test** on your terminal to actually run the tests and output your score.
- Finally run **valgrind --leak-check=full ./stack_test** to check whether there are memory leaks.