Programming Assignment #1
Incentive Date: Feb 6th 2024
Due Date: Feb 9th 2024
Stacks

Description:
For this programming assignment, you will implement a **Stack** whose size can grow as elements are inserted into the stack. You will **build three different implementations** of this Stack. **Two** of these implementations will be **array-based.** These two 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. Finally, for the **third implementation,** you should implement a Stack using a **Linked List.**

[Advisory]:

- **As you have been tasked with creating the StackArrayDouble and StackArrayLinear classes in LE1, you may simply copy the implemented code for these classes from LE1 to PA1.**

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.
  - **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
- **StackLinkedList.h**
  - This header file declares the StackLinkedList class which is also a derived class of the AbstractStack class. This file declares the different methods that StackLinkedList must

implement (as a consequence of these methods being pure virtual in the base AbstractStack class).
    - In the StackLinkedList.h file, we have also declared a **Node class** that contains the following two public members:
        - **T data:** This will be the actual data that the Node will be storing.
        - **Node<T>* next:** This is a pointer to the next Node in the Linked List. This must be set to nullptr if the concerned Node is the last node.
    - The StackLinkedList class has two private member variables:
        - **Node<T>* head:** This is the pointer to the first Node in the Linked List. This must be set to nullptr if the Linked List is empty.
        - **int length:** This variable keeps track of the size of the Linked List (and thereby the Stack).
- **main.cpp**
    - You can use this file to instantiate StackLinkedList, StackArrayDouble, and StackArrayLinear objects and test your code.

Contract (TASK):
Your task is to implement the methods declared in the StackLinkedList, StackArrayDouble, and StackArrayLinear classes. For each of these classes, you have been provided with non-functional definitions of these methods in their respective header files. **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 for the StackArrayDouble and StackArrayLinear classes.
- 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>-<PA1>.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)

- **StackLinkedList.h** (containing the functional method definitions of the StackLinkedList class)
- **PA1_Report.pdf** (containing your report - details mentioned below)

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

Grading:

This assignment is worth 80 points. The 80 points will be split up between Coding (50 points) and a Report (30 points).

Coding (50 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 each of the three classes:
  - 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

Your final score will be determined using the following formula:

- **StackArrayDouble + StackArrayLinear + (3 * StackLinkedList)**

For example, if you received:

- **8 points for methods of StackArrayDouble**
- **10 points for methods of StackArrayLinear**
- **7 points for methods of StackLinkedList**

Your final score will be 8 + 10 + (3 * 7) = 39 / 50

In case of any memory leaks in your program, we will subtract 5 points.

Report (30 points)

You will write a brief report that includes theoretical analysis, a description of your experiments, and discussion of your results. At a minimum, your report should include the following sections:

1. ***Introduction (5 points)***: In this section, you should describe the objective of this assignment.
2. ***Theoretical Analysis (10 points)***: In this section, you should provide an analysis of the complexity of a push operation to the extent that has been presented in lecture discussion. Describe the effect of a push operation and the advantages and disadvantages of the three strategies. What is the complexity of a push (on average) for the different implementations? What

is the worst case complexity for the different implementations?
Advisory: It is understood that a formal discussion of algorithmic complexity is yet to come in later modules. With that in mind, treat this as a discussion oriented section and feel free to lean on providing a qualitative analysis based on knowledge of complexity gained thus far from lecture content.

3. ***Experimental Results (15 points)***: In this section, you should compare the performance (running time) of the push() operation in the three different implementations to one another and to their theoretical complexity.
    a. Make a plot showing the running time (y-axis) vs. the number of push operations (x-axis) for the three implementations. You must use some electronic tool (matlab, gnuplot, excel, …) to create the plot. **Hand-written plots will NOT be accepted.**
    b. Provide a discussion of your results, which includes but is not limited to:
        i. Which of the three Stack implementations performs the best? Does it depend on the input? Provide some reasoning behind your observation.
        ii. To what extent does the theoretical analysis agree with the experimental results? Attempt to understand and explain any discrepancies you note.

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, StackArrayLinear, and StackLinkedList 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 using testInfrastructure, you can follow either of the two options mentioned below:

- **Option 1:** Move your AbstractStack.h, StackLinkedList.h, StackArrayDouble.h, and StackArrayLinear.h into the testInfrastructure directory. After this, run **python3 test.py**
- **Option 2:** Move your AbstractStack.h, StackLinkedList.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.