CS 210 Data Structures (Shen) - Fall 2022

Programming Assignment 2 (100 points)

Due Date – **Beginning of the class**, **10/17/2022, NO LATE** submission

You **must** do this assignment on your own.

A car company promotes a particular model of its electric cars every month.

Part A – Coding

Write a class to keep track of the monthly promoted models and their prices with a **stack** data structure. The class needs to support:

1) Pushing the latest promoted model onto the stack.
2) Popping the latest promoted model off the stack.
3) Peeking the latest promoted model at the top of the stack (without popping).
4) Getting the lowest priced model and highest priced model among the past promoted models.

Files to get you started:

- **promotedCarModelStack.h:** The header file containing the PromotedModel and PromotedCarModelStack class definitions. You **must NOT** change the class names and the function signatures, otherwise, Gradescope autograding will fail. You may **add member variables** or helper functions to the PromotedCarModelStack class as you see necessary.
- **promotedCarModelStack.cpp:** The implementation file where you will write your code for implementing the **stack** functions.
    - You must **NOT** rename the promotedCarModelStack.h and promotedCarModelStack.cpp; otherwise, Gradescope autograding will fail.
- **driver.cpp:** A file you can run to test your code. It includes test examples for basic tests; however, you may want to write more tests to test your code according to the prompt.
- **Makefile:** A makefile to allow you to automate the compilation of your code. Check out the "Frequently Asked Questions" link in the C/C++ programming in Linux / Unix reference page on Canvas for simple Makefile reference.
    - Use the "make" command to compile. Use "make clean" to delete compiled object code or executable. Cleaning before each make is important as you can end up running old code even after doing a new make command.
    - Use **./carstack** to execute.

Assumptions:

1. Prices of the promoted models in different months will be different.

Algorithm Requirements:

1) Breaking either one of the requirements in 3), 4), or 5) below would cap your assignment 2 grade to 50% of the total assignment 2 grade, meaning the most you could get from assignment 2 would be 50 points.
2) The **`pop`**, **`peek`**, **`getHighestPricedPromotedModel`**, **`getLowestPricedPromotedModel`** should throw a **logic_error exception** with an error message "Promoted car model stack is empty" if the PromotedCarModelStack is empty.
3) You **must NOT** use the **C++** standard template library **std::stack** class, you **must** implement your **own Stack**.

a. Whether you use dynamic array-based list (vector) or linked list for implementing the stack (see hint below), you can use **at most two vectors** or **two linked lists** or [**one vector and one linked list**] in your **Stack** class for tracking the monthly promoted models, and the highest and lowest price models over the time.

4) All methods should run in constant **O(1) time** complexity.
5) All methods should run with constant auxiliary **O(1) space**. Auxiliary space **does NOT** include the space for the method arguments and class member variables.

Hint:

1) For push, pop, peek operations, think which C++ STL collection type (other than the stack class) would give constant time and space for accessing or updating the beginning or the end of the data collection. You could use an implementation of the list ADT such as a dynamic array (amortized O(1) time for appending and removing at the end of the array) or a linked list type for implementing your Stack and its operations.
2) You need to be able to retrieve the highest and lowest priced models from the stack in **constant time** at any time, which data structure would give constant-time lookup time?
3) Note the highest and lowest priced models in the stack could change with each push or pop, you basically would need to track the highest and lowest priced model at every point of the stack.

Programming and testing:

- Please refer to C/C++ programming in Linux / Unix page.
- You may use **C++ 11** standard for this assignment, see the given Makefile.
- We strongly recommend you set up your local development environment under a Linux environment (e.g., Ubuntu 18.04 or 20.04, or CentOS 8), develop and test your code there first, then port your code to Edoras (e.g., filezilla or winscp) to compile and test to verify. The gradescope autograder will use a similar environment as Edoras to compile and autograde your code.

Grading:

Passing 100% auto-grading may NOT give you a perfect score for this Part A. The satisfaction of the algorithm requirements (see above), your code structure, coding style, and commenting will also be part of the rubrics (**see Syllabus Course Design - assignments**). Your code shall follow industry best practices:

- Be sure to comment your code appropriately. Code with no or minimal comments are automatically lowered one grade category.
- NO hard code – Magic numbers, etc.
- Have proper code structure between .h and .c / .cpp files, do not #include .cpp files.
- Design and implement clean interfaces between modules.

Part B – Complexity Analysis

Show your algorithm has the time and auxiliary space complexities as required. For each function / method implemented, you would need to analyze the implementation logic for the **worst-case** scenario to separate the constant and non-constant operations with respect to the stack size for tracking monthly promoted models and their prices, then show and explain the time complexity and space complexity in the O notations. Follow **zyBook 12.7.1 and 12.7.3** for **time** complexity analysis, follow **zyBook 11.6.6** for **space** complexity analysis.

Turning In

You need to submit the following program artifacts on Gradescope. Make sure that all submitted files contain your **name** and **Red ID**.

- **Part A:**
  - You should only submit promotedCarModelStack.h and promotedCarModelStack.cpp source code files. **Do not** upload the driver, Makefile, nor any of the compiled .o files.
  - Make sure you type the **Single Programmer Affidavit** (refer to the template on Canvas) as part of the comments at the beginning of your **promotedCarModelStack.cpp file**.
- **Part B: One PDF file**. It is recommended that you type it up, however, a scanned copy is allowed so long as the handwriting is legible. You can convert Word (.docx) files to PDF using the "Save as" functionality. You must only submit a **single** PDF file, image files (.jpg, etc) **will not** be accepted.
- **Important**:
  - Upload your files directly to Gradescope.
  - Do NOT **compress / zip** files into a ZIP file and submit, submit all files as they are.
  - Do NOT submit any .o files or test files.
- **Number of submissions:**
  - Please note the autograder submission count when submitting on Gradescope. For this assignment, you are **limited to a maximum of 10 submissions.** As stressed in the class, you are supposed to do the testing in your own dev environment instead of using the autograder for testing your code. It is also the responsibility of you as a programmer to sort out the test cases based on the requirement specifications instead of relying on the autograder to give the test cases.

Academic honesty

Posting this assignment to any online learning platform and asking for help is considered academic dishonesty and will be reported.

An automated program structure comparison algorithm will be used to detect code plagiarism.

- The plagiarism detection generates similarity reports of your code with your peers as well as from online sources. **We will also include solutions from the popular learning platforms (such as Chegg, etc.) as part of the online sources used for the plagiarism similarity detection**. Note not only the plagiarism detection checks for matching content, but also it checks the structure of your code.
- Refer to Syllabus for penalties on plagiarisms.
- Note the provided source code in the code skeleton would be excluded in the plagiarism check.