CS 210 Data Structures (Shen) - Fall 2022

Programming Assignment 1 (80 points)

Due Date – **11:59pm, 09/22/2022, NO LATE** submission

You must do this assignment on your own.

A car company manufactures three models of electrical cars: **mx, my, mz**. From time to time, the company would like to promote these models with a particular order, for example, **{**"my", "mx", "mz"**}**. The promotion order would then be used to sort the company's online car inventory.

Part A – Coding

Implement a function to sort a given car inventory based on a given promotion order.

Write your code in the given **sortcar.cpp** (see below):

```
vector<string> SortCar::sortCarInventory(vector<string> carInventory,
                                         vector<string> promotionOrder)
{
   // Write your implementation here
   return carInventory;
}
```

Arguments **carInventory** and **promotionOrder** are std::string vectors.

A test input could be:

vector<string> carInventory {"mz", "my", "my", "mx", "mz", "mx", "my", "mz"}

vector<string> promotionOrder {"mz", "mx", "my"}

The **sample output** returned from calling sortCarInventory(…) with the above input would be:

{"mz", "mz", "mz", "mx", "mx", "my", "my", "my"}

Assumptions:

1. promotionOrder vector **always contains 3 distinct** strings in "mx", "my", and "mz" of a particular order.
2. carInventory vector: possible car model strings contained in carInventory are "mx", "my", or "mz". **Note** carInventory **may** contain no string, one string, or more than one string.

Algorithm Requirements:

1. Your algorithm must run with **linear** time complexity, i.e., **O(n)**, with n being the size of input carInventory array.
2. You can **ONLY** use std::vector class (you may also use the raw C/C++ array), using any C++ STL collection class **other than** the **std::vector**, or using any sorting related functions from the STL <algorithm> is **NOT** allowed.
3. Your algorithm can **at the most** iterate through the carInventory array for **TWO passes / loops**.
4. Your algorithm must run with **constant auxiliary space** complexity, i.e., **O(1)**, meaning your algorithm would need to perform sorting of the carInventory array in place by mutating the carInventory vector. It can **NOT** use any auxiliary space (in addition to the input vector) that is proportional to the input car inventory size.
5. You can **NOT** use the **bucket-sorting** algorithm. The bucket sorting like algorithm for solving this assignment would be like the following steps (if you are counting occurrences of promotedModels, anything similar to those counts, you are doing a similar method of this):

a. Loop through the carInventory array once, count the number of occurrences of each promoted car model from the promotionOrder array, these counts of the promoted models would be like the buckets: say numOfFirstPromotedModel, numOfSecondPromotedModel, numOfThirdPromotedModel.
b. Then loop through the carInventory array a second time: starting from the beginning and adding a firstPromotedModel while decrementing numOfFirstPromotedModel until it reaches 0, repeating this process for the remaining two promotedModels.
c. Return carInventory.

6. Breaking either one of the requirements in 1), 2), 3), 4), or 5) would result in a **50%** penalty to your Assignment 1 grade.

Hint:

- Use indices to track and swap string elements in carInventory according to promotionOrder.

Files to get you started:

- **sortcar.h:** The header file containing the SortCar class definition. You **must NOT** change the name of the class and signature of the **sortCarInventory** function, **otherwise**, Gradescope autograding will fail. You **may add member variables or helper functions** to the class as you see necessary.
- **sortcar.cpp:** The implementation file where you will write your code for implementing the **sortCarInventory** function.
    - You must **NOT** rename the sortcar.cpp, **otherwise**, Gradescope autograding will fail.
- **driver.cpp:** A file you can run to test your code. It includes three basic tests; however, you may want to write more tests based on the possible combinations of the input arguments. The autograder will test exhaustively with many combinations of the input arguments.
- **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 at command line prompt 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.

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), you 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. (This may not be applicable to this first assignment)

Part B – Complexity Analysis

Show your algorithm has **O(n)** time complexity (n is the size of carInventory) and **O(1) auxiliary** space complexity. For time complexity, follow **zyBook 12.7.1 and 12.7.3**: you would first analyze your implementation logic to separate the constant and non-constant operations relevant to the input carInventory array size, then come up with the time complexity function. For auxiliary space complexity, follow zyBook 11.6.6 to come up with the space complexity function with respect to the input carInventory array size n. Then, simplify the time and auxiliary space complexity functions further to the O notations following **zyBook 12.6.1** (note: Big O analysis considers the worst-case scenarios).

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 **sortcar.h and sortcar.cpp** source code files. **Do not** upload the driver.cpp, Makefile, nor any of the compiled .o files. sortcar.h is optional to submit, if you do not include it, the autograder will use the default one provided to you.
  - Make sure you type the **Single Programmer Affidavit** (refer to the template on Canvas) as part of the comments at the beginning of your **sortcar.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 will be allowed **99** submissions, but **future assignments will be limited to around 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 the 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.