CS 210 Data Structures (Shen) - Summer 2024

Programming Assignment 1 (100 points)

Due Date – **10am, 06/06/2024, NO LATE** submission

You must do this assignment on your own.

You are maintaining the front end of an e-commerce site. Based on the daily sales stats, you are asked to sort the online inventory to move the most promoted product to the beginning of the inventory and move the least promoted product to the end of the inventory.

Part A – Coding (70 points)

Implement a function to sort a given inventory **based on the most and least** promoted items:

- **most** promoted items should be moved to the **beginning** of the inventory,
- **least** promoted items should be moved to the **end** of the inventory, and
- **the order** of all **other** items should be **preserved**.

Write your code in the given **sortinventory.cpp** (see below), you **must NOT** change the function signature:

```
vector<string> SortInventory::sortInventory(vector<string> items,
                                            string mostPromotedItem,
                                            string leastPromotedItem)
{
  // write your implementation here
  return items;
}
```

The items argument is a std::string vector.

A test input could be:

```
vector<string> items {"tv", "pen", "phone", "pen", "book", "phone", "pen", "speaker"}
string mostPromotedItem "phone"
string leastPromotedItem  "pen"
```

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

```
{"phone", "phone", "tv", "book", "speaker", "pen", "pen", "pen"}
```

Note:

1. **items** vector may contain no string, one string, or more than one string.
2. **Possible values** for strings in items vector as well as for mostPromotedItem and leastPromotedItem can be **arbitrary**.
3. Use **case insensitive comparison** when evaluating a string in the items.

Files to get you started:

- **sortinventory.h:** The header file containing the SortInventory class definition. You **must NOT** change the name of the class and signature of the sortInventory function, **otherwise**, Gradescope autograding will fail. You **may add member variables or helper functions** to the class as you see necessary.

- **sortinventory.cpp:** The implementation file where you will write your code for implementing the sortInventory function.
- You must **NOT** rename **sortinventory.h** or **sortinventory.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 and their expected output. 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.

Algorithm Requirements:

1) **Breaking any one of the requirements in 2), 3), or 4)** below would result in a **50% penalty** to your Assignment 1 grade.
2) **Computational complexity** requirements:
    a. **Time complexity**:
        i. your algorithm must run with **linear** time complexity, i.e., **O(n)**, with **n** being the size of input item array, **and**
        ii. your algorithm can **at the most** iterate through the **items** array for **TWO passes / loops**.
        iii. note if you choose to use the vector erase in your implementation, erase has an O(n) complexity. If it is used in a loop (with n iterations), the complexity would be O(n^2).
    b. **Auxiliary space complexity**: your algorithm must run with **constant** auxiliary space complexity, i.e., **O(1)**. It can **NOT** use any auxiliary space (i.e., extra space in addition to the input vector) that is proportional to the input items size. This implies your algorithm would need to perform sorting by **in place swapping** of the elements in the items vector.
3) What you may use from **standard template library** (STL):
    a. You can **ONLY** use **std::vector** class (you may also use the raw C/C++ array in the sorting implementation).
    b. You **may NOT use any other** C++ STL collection class.
4) You may **NOT use any function** from the STL <**algorithm**>.

Hint:

- Use indices to track and swap string elements in items according to most and least promoted items.

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.
- You are strongly recommended to set up your local development environment using a Linux OS (e.g., Ubuntu 20.04 or 22.04, or CentOS 8) to develop and test your code. You could also use Edoras (with CentOS Linux) to develop and test there. The gradescope autograder uses Ubuntu 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 on your code appropriately. Code with no or minimal comments is automatically lowered one grade category.
- Meaningful variable names.
- 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 (20 points)

Show your algorithm has **O(n)** time complexity (n is the size of **items**) and **O(1) auxiliary** space complexity. For time complexity, follow **zyBook 12.6.1 and 12.6.3**: you would first analyze your implementation logic to separate the constant and non-constant operations relevant to the input **items** array size, then come up with the time complexity function. For auxiliary space complexity, follow **zyBook 11.6.6** to come up with the auxiliary space complexity function with respect to the input **items** array size n. Then, simplify the time and auxiliary space complexity functions further to the O notations following **zyBook 12.5.1** (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 **sortinventory.h and sortinventory.cpp** source code files. **Do not** upload the driver.cpp, Makefile, nor any of the compiled .o files. **sortinventory.h is optional** if you do not add code to it; otherwise, submit it along with sortinventory.cpp. Note again, you may NOT change the sortInventory function signature.
  - Make sure you type the **Single Programmer Affidavit** (refer to the template on Canvas) as part of the comments at the beginning of your **sortinventory.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 relying on the autograder for testing your code. It is also

your responsibility as a programmer to sort out the test cases based on the requirement specifications instead of using 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.

- 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, github, chatgpt, etc.) and / or past submissions as part of the online sources used for 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 plagiarism.
- Note the provided source code in the code skeleton would be excluded in the plagiarism check.

SDSU's Center for Student Rights and Responsibilities have officially added plagiarism policies of using ChatGPT for academic work, as put below:

"Use of ChatGPT or similar entities [to represent human-authored work] is considered academic dishonesty and is a violation of the Student Code of Conduct. Students who utilize this technology will be referred to the Center for Student Rights and Responsibilities and will face student conduct consequences up to, and including, suspension."