Sabanci University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Summer 2022-2023

Take-Home Exam 3
The Hard Way

Due: 17 August 2023 11.55pm (SHARP)

DISCLAIMER:

Only checking the sample run cases might not be sufficient as your solution will be checked against a variety of samples different from the provided samples; however checking and studying these cases is highly encouraged and recommended.

You can NOT collaborate with your friends and discuss your solutions with each other. You have to write down the code on your own. Plagiarism will not be tolerated AND cooperation is not an excuse!

Introduction

The aim of this THE is to practice on stack structures. You will implement a program for finding the common words of two given files. It might seem like a trivial CS201 task at first but you **must** do this by using ***dynamic stacks***.

Input to Your Program

Your program will first ask for two file names. Afterwards, it will ask an option question, to which the user can only respond with a "1" or "2".

Format of the Inputs

At the very beginning, your program will ask for the name of the first input file. Your program should keep asking the same question until a valid file name is given. The same procedure shall be applied for the second file name as well. Once you get both file names correctly, you can start parsing the information out of the files.

The files may consist of more than one line and there might be empty lines as well. Besides, the words within the files might be separated by spaces, tabs or multiple of these. However, there won't be any punctuation marks and/or digits within the files. They will just consist of a bunch of words.

Lastly, your program will ask the user to choose the option "1" or "2". Option "1" means that the common words will be displayed in the order that they appear in the first file, whereas Option "2" means that they will be displayed in the order that they appear in the second file. If the option is invalid, your program should print an appropriate error message and ask again until a valid option is entered by the user. Details of these operations can be inspected in the Sample Runs section.

Data Structure to be Used

You ***must use dynamic stacks*** for this THE. Indeed, you are not allowed to use any other aggregate data type other than dynamic stacks (***array, vector, queue, static stack etc. may not be used anywhere in your code***). Moreover, you are ***not allowed to read the files more than once***. You need to think on how to utilize dynamic stacks to store linear information. We inspect your codes other than testing them with test cases; so any attempt of using an alternative data structure or reading the files more than once will be easily spotted and your grade will be set to zero.

Note: A dynamic stack is a stack which is implemented in a dynamic linked list with dynamic memory allocation and deallocation operations. You are not allowed to implement a static stack which uses arrays or vectors internally, for this THE.

Details of the Tasks and Outputs of Your Program

Your program should find the words that appear in both of the files. For this, your program should first read the files and store the words in dynamic stack(s). You may want to design your dynamic stack class in a way that you can store the occurrences of the words along with them, because your program should be displaying that information in the end as well.

Depending on the user's choice, you may display the common words in two different orders. If the user chooses "1", your program will display the common words in the order that they appear in the first file. If option "2" is chosen, the same will apply for the second file. Keeping the information in stacks will help you not to lose track of the order of those words in the files.

In the end, your program will display the words which occur in both of the files and their ***minimum occurrence count*** (i.e if a word occurs 5 times in a file and 3 times in another file, you should display that the word occurs at least 3 times). The detail is given in Sample Runs.

You already have an implementation of the DynIntStack class that you know from the lectures; however this class is not fully suitable for your task. You should be deciding on what kind of information you want to store in your dynamic stacks and edit the class methods and such accordingly. Moreover, you ***must implement a destructor*** for the dynamic stack class such that it will deallocate the dynamically created memory for that dynamic stack object.

You are encouraged to carefully check the sample runs as they complement the textual description, demonstrate different cases that your code should handle, and they also show the expected behavior of your program.

Sample Runs

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are the standard inputs (cin) taken from the user (i.e., like ***this***). You have to display the required information in the same order and with the same words as here.

Sample Run 1

```
This program finds the common words of two files using stacks.
---
Enter the first file name:
file
Enter the first file name:
incorrect
Enter the first file name:
```

file1_v3.txt
Enter the second file name:
file
Enter the second file name:
invalid
Enter the second file name:
file2_v3.txt
Choose with respect to which file the result will be sorted to (1: first file, 2: second file):
1

The word "remember" occurred at least 1 time(s) in both files.
The word "spicy" occurred at least 1 time(s) in both files.
The word "gifted" occurred at least 1 time(s) in both files.
The word "watch" occurred at least 1 time(s) in both files.
The word "caring" occurred at least 1 time(s) in both files.
The word "temper" occurred at least 1 time(s) in both files.
The word "harsh" occurred at least 1 time(s) in both files.
The word "mice" occurred at least 1 time(s) in both files.
The word "representative" occurred at least 1 time(s) in both files.
The word "bell" occurred at least 4 time(s) in both files.
The word "stretch" occurred at least 1 time(s) in both files.
The word "yam" occurred at least 1 time(s) in both files.
The word "theory" occurred at least 1 time(s) in both files.
The word "paper" occurred at least 1 time(s) in both files.
The word "listen" occurred at least 1 time(s) in both files.
The word "profuse" occurred at least 1 time(s) in both files.
The word "nest" occurred at least 4 time(s) in both files.
The word "notebook" occurred at least 1 time(s) in both files.
The word "dime" occurred at least 1 time(s) in both files.
The word "turn" occurred at least 1 time(s) in both files.
The word "face" occurred at least 1 time(s) in both files.
The word "foamy" occurred at least 2 time(s) in both files.
The word "nonchalant" occurred at least 3 time(s) in both files.
The word "muddled" occurred at least 1 time(s) in both files.
The word "health" occurred at least 2 time(s) in both files.
The word "garrulous" occurred at least 2 time(s) in both files.
The word "vanish" occurred at least 1 time(s) in both files.
The word "labored" occurred at least 1 time(s) in both files.

```
The word "excited" occurred at least 1 time(s) in both files.
```

Sample Run 2

```
This program finds the common words of two files using stacks.
---
Enter the first file name:
file2_v3.txt
Enter the second file name:
file4_v3.txt
Choose with respect to which file the result will be sorted to
(1: first file, 2: second file):
aaa
---
Invalid choice
---
Choose with respect to which file the result will be sorted to
(1: first file, 2: second file):
0
---
Invalid choice
---
Choose with respect to which file the result will be sorted to
(1: first file, 2: second file):
-1
---
Invalid choice
---
Choose with respect to which file the result will be sorted to
(1: first file, 2: second file):
2
---
```

Sample Run 3

```
This program finds the common words of two files using stacks.
---
Enter the first file name:
file3_v3.txt
Enter the second file name:
file2_v3.txt
```

```
Choose with respect to which file the result will be sorted to
(1: first file, 2: second file):
2
---
The word "bell" occurred at least 3 time(s) in both files.
The word "nest" occurred at least 5 time(s) in both files.
The word "lackadaisical" occurred at least 1 time(s) in both
files.
The word "yam" occurred at least 1 time(s) in both files.
The word "excited" occurred at least 1 time(s) in both files.
The word "learn" occurred at least 1 time(s) in both files.
The word "jar" occurred at least 1 time(s) in both files.
The word "offer" occurred at least 1 time(s) in both files.
The word "theory" occurred at least 1 time(s) in both files.
The word "soup" occurred at least 1 time(s) in both files.
The word "perform" occurred at least 1 time(s) in both files.
The word "future" occurred at least 1 time(s) in both files.
The word "playground" occurred at least 1 time(s) in both files.
The word "vanish" occurred at least 1 time(s) in both files.
The word "pot" occurred at least 1 time(s) in both files.
The word "lumpy" occurred at least 1 time(s) in both files.
The word "stretch" occurred at least 1 time(s) in both files.
The word "outrageous" occurred at least 1 time(s) in both files.
The word "crash" occurred at least 1 time(s) in both files.
The word "intend" occurred at least 1 time(s) in both files.
The word "terrify" occurred at least 1 time(s) in both files.
The word "important" occurred at least 1 time(s) in both files.
The word "watch" occurred at least 1 time(s) in both files.
```

Sample Run 4

```
This program finds the common words of two files using stacks.
---
Enter the first file name:
file3_v3.txt
Enter the second file name:
file2_v3.txt
Choose with respect to which file the result will be sorted to
(1: first file, 2: second file):
1
```

```
---
The word "watch" occurred at least 1 time(s) in both files.
The word "yam" occurred at least 1 time(s) in both files.
The word "excited" occurred at least 1 time(s) in both files.
The word "learn" occurred at least 1 time(s) in both files.
The word "jar" occurred at least 1 time(s) in both files.
The word "offer" occurred at least 1 time(s) in both files.
The word "theory" occurred at least 1 time(s) in both files.
The word "soup" occurred at least 1 time(s) in both files.
The word "perform" occurred at least 1 time(s) in both files.
The word "future" occurred at least 1 time(s) in both files.
The word "playground" occurred at least 1 time(s) in both files.
The word "vanish" occurred at least 1 time(s) in both files.
The word "pot" occurred at least 1 time(s) in both files.
The word "lumpy" occurred at least 1 time(s) in both files.
The word "stretch" occurred at least 1 time(s) in both files.
The word "outrageous" occurred at least 1 time(s) in both files.
The word "crash" occurred at least 1 time(s) in both files.
The word "intend" occurred at least 1 time(s) in both files.
The word "nest" occurred at least 5 time(s) in both files.
The word "bell" occurred at least 3 time(s) in both files.
The word "terrify" occurred at least 1 time(s) in both files.
The word "important" occurred at least 1 time(s) in both files.
The word "lackadaisical" occurred at least 1 time(s) in both
files.
```

Some Important Rules Regarding Submission

In order to get full credit, your program must be efficient, modular (with the use of functions), well commented and properly indented. Besides, you also have to use understandable identifier names. Presence of any redundant computation, bad indentation, meaningless identifiers or missing/irrelevant comments may decrease your grade in case that we detect them. When we grade your THEs, we pay attention to these issues. Moreover, **we may test your programs with very large test cases**. Hence, take into consideration the efficiency of your algorithms other than correctness.

Sample runs give a good estimate of how correct your implementation is, however, we will test your programs with different test cases and **your final grade may conflict with what you have seen on CodeRunner**. We will also **manually**

check your code, indentations and so on, hence do not object to your grade based on the **CodeRunner** results, but rather, consider every detail on this documentation. **So please make sure that you have read this documentation carefully and covered all possible cases, even some other cases you may not have seen on CodeRunner or the sample runs**. The cases that you *do not need* to consider are also given throughout this documentation.

Submit via CodeRunner on SUCourse ONLY! Paper, e-mail or any other methods are not acceptable.

The internal clock of SUCourse might be a couple of minutes skewed, so make sure you do not leave the submission to the last minute. In the case of failing to submit your THE on time:

"No successful submission on SUCourse on time = A grade of zero (0) directly."

You will submit three files: *DynStack.cpp*, *DynStack.h* files and *main.cpp*. DynStack.cpp file's content will be copied and pasted into the "Answer" area as in other assignments. However, you will upload DynStack.h file and main.cpp file as attachments to your submission in the relevant assignment submission page on SUCourse.

How to get help?

You may ask your questions to TAs or to the instructor. Information regarding the office hours of the TAs and the instructor are available at SUCourse.

What and where to submit (PLEASE READ, IMPORTANT)

It'd be a good idea to write your name and last name in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). If your full name is "Duygu Karaoğlan Altop", and if you want to write it as comment; then you must type it as follows:

// Duygu Karaoglan Altop

You should copy the full content of the .cpp file and paste it into the specified "Answer" area in the relevant assignment submission page on SUCourse. ***Please note that the warnings may be considered as errors on CodeRunner, which means that you should have a compiling and warning-free program***.

Since the grading process will be automatic, you are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be zero (0). Any tiny change in the output format will result in your grade being zero (0), so please test your programs yourself, and against the sample runs that are available at the relevant assignment submission page on SUCourse.

In the CodeRunner, there are some visible and invisible (hidden) test cases. You will know whether your code has successfully passed all the test cases or not before submitting your code. There is no re-submission. You don't have to complete your task in one time, you can continue from where you left last time but you should not press submit before finalizing it. Therefore, you should make sure that it's your final solution version before you submit it. Also, we still do not suggest that you develop your solution on CodeRunner but rather on your IDE on your computer.

You may visit the office hours if you have any questions regarding submissions.

How to get help?

You may ask your questions to TAs or to the instructor. Information regarding the office hours of the TAs and the instructor are available at SUCourse.

Plagiarism

Plagiarism is checked by automated tools, and we are very capable of detecting such cases. Be careful with that. Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do NOT send any part of your code to your friends by any means or you might be charged as well, although you have done your THE by yourself. THEs are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**
In case of plagiarism, the rules on the Syllabus apply.

Good Luck!
Ahmed Salem, Duygu Karaoğlan Altop