Sabanci University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Summer 2020-2021

Take-Home Exam 3
Due: 20 August 2020 11.55pm (Sharp Deadline)

DISCLAIMER:

Your program should be a robust one such that you have to consider all relevant user mistakes and extreme cases; you are expected to take actions accordingly!

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 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 take-home exam is to practice on queue data structure, class templates and operator overloading. You are asked to overload some operators (=, +=, *=, +, <<, >>) as member/free functions. The details will be explained in subsequent sections of this assignment document.

Main is Given

We have given you the *main.cpp* file that you will use in your project. You **are NOT allowed to change** this file (we will replace it while grading anyways :)). What you rather need to do is to implement your class in a way that it will work harmoniously with the given main function. Since we will test your solution with a different *main.cpp* file, you should implement the methods and operators in the way they should be, rather than finding workarounds.

Please do NOT forget to change the name of the *main* file to ***"SUCourseUserName_the3.cpp"***. **This is very important; otherwise we cannot grade your submission**.

What does Main do?

The main function starts by reading a file name from the console and opening it. Then, by using the extraction operator (>>) that you will overload, it reads the contents of this file into a DynQueue object. Later, it does several operations with an integer typed queue and displays the contents of the queue on the console after each operation, using the insertion operator (<<) that you will overload. Afterwards, it will take another file name from the console and perform similar operations with a string typed queue. You can inspect the main file for details, as it is quite short.

Class to Implement

You can use the DynIntQueue class covered during the lectures as a basis. This basis class:

1) Is not templated,
2) Works for only integers,
3) Does not implement deep copy constructor,
4) Does not implement destructor, and
5) Lacks several other operators that the main function uses.

The ***definitive tasks*** of the assignment can be ordered like this:

- Convert the basis class into a templated version,
- Implement a deep copy constructor (shallow copy might result in failure with another main program),
- Implement the destructor (deallocate the dynamic queue held within the object),
- Implement extraction operator >> (you may assume that this function will only be used with ifstream objects, but not with cin or istringstream.),
- Implement insertion operator (<<),
- Implement =, +=, *=, and + operators for the class:
    - = operator to perform a deep copy,
    - += operator to add a new object to the queue,
    - *= operator to multiply the value field of each node of the queue with the right hand side operand, which is to be an integer,
        - In case that the value type is a string or char, this operator should work as a repeater (like in Python).
    - + operator to add the value fields of each no to one another.

As you all know, templated class function definitions should be in the same translation unit with the source files that they are used within. In order for you not to have any problems with this, and also for us to construct the projects for grading easier, please do all the implementation in the **header file** as given and do **not** create a *.cpp* file for the DynQueue class.

Format of the Input Files

The input file will contain the values that are going to be the elements of the queue to be created. Your program should read the file and create the DynQueue object. Each line of this file contains a single value so as to be considered as a single queue item. You may assume that there will be at least one line in the given file.

Rules

In this assignment, the existence of *main.cpp* already narrows down the kind of implementation that you can follow. On top of this, there are other limitations as well.

First of all, you are **not** allowed to use vectors or other data structures inside your class to hold the data. It **must** be a dynamic queue as it is in the basis class. As a result, you are going to implement a node that can store a value of any given type. These restrictions follow the fact that implementing the assignment operator, destructor and copy constructor would be meaningless with a vector field in the class (as these would be done automatically). We want you to get experience on implementing these methods, hence we restrict such usages.

Second of all, the definitive tasks given in the previous section **must** be all there. Although one or two of them may seem irrelevant to you, **you must** implement all of them or you will risk your partial credits from the homework. We always inspect your codes and see what you lack or do not lack in your implementation.

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

Please enter a filename: ***inFile.txt***

1 2 3 4 5 2 3 4 5 6 3 4 5

1 2 3 4 5 2 3 4 5 6 3 4 5 2

1 2 3 4 5 2 3 4 5 6 3 4 5 2

2 4 6 8 10 4 6 8 10 12 6 8 10 4

3 6 9 12 15 6 9 12 15 18 9 12 15 6

6 12 18 24 30 12 18 24 30 36 18 24 30 12

Please enter a filename: ***inFile.txt***

1 2 3 4 5 2 3 4 5 6 3 4 5

1 2 3 4 5 2 3 4 5 6 3 4 5 2

1 2 3 4 5 2 3 4 5 6 3 4 5 2
11 22 33 44 55 22 33 44 55 66 33 44 55 22
111 222 333 444 555 222 333 444 555 666 333 444 555 222
111111 222222 333333 444444 555555 222222 333333 444444 555555 666666 333333 444444 555555 222222

Some Important Rules

Although some of the information is given below, please also read THE submission and grading policies from the lecture notes of the first week. In order to get a full credit, your program must be efficient, modular (with the use of functions), well commented and 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, in order to observe the real performance of your codes, we are going to run your programs in Release mode and **we may test your programs with very large test cases**. Hence, take into consideration the efficiency of your algorithms other than correctness.

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+.

YOU CAN USE GRADE CHECKER FOR THIS THE!

You can use GradeChecker (https://learnt.sabanciuniv.edu/GradeChecker/) to check your expected grade. Just a reminder, you will see a character ¶ which refers to a newline in your expected output.

Make sure you upload any supporting files that you uses, too.

Grade Checker and the automated grading system use a different compiler than MS Visual Studio does. Hence, you should check the “Common Errors” page to see some extra situations to consider while doing your assignment. If you do not consider these situations, you may get a lower score (even zero) even if your program works correctly with MS Visual Studio.

Common Errors Page: https://learnt.sabanciuniv.edu/GradeChecker/commonerrors.jsp

Grade Checker can be pretty busy and unresponsive during the last day of the submission. Due to this fact, leaving the assignment for the last day generally is not a good idea. You may wait for hours to test your assignment or make an untested submission, sadly..

Grade Checker and Sample Runs together give a good estimate of how correct your implementation is, however we may test your programs with different test cases and **your final grade may conflict with what you have seen on Grade Checker.** We will also **manually** check your code, indentations and so on, hence do not object to your grade based on the Grade Checker 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 Grade Checker or Sample Runs**. The cases that you *do not need* to consider are also given throughout this documentation.

Submit via SUCourse ONLY! **Grade Checker is not considered as a submission**. Paper, e-mail or any other methods are not acceptable, either.

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 assignment on time:

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

What and where to submit (PLEASE READ, IMPORTANT)

You should test your program using Grade Checker. We will use the same UNIX based C++ compiler that Grade Checker uses for grading your THE.

It'd be a good idea to write your name and lastname 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

Submission guidelines are below. Since the grading process will be semi-automatic, you are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be zero. The lack of even one space character in the output will result in your grade being zero, so please test your programs yourself.

- Name the main.cpp into:

  "SUCourseUserName_the3.cpp"

  Your SUCourse user name is actually your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SU e-mail address is **atam@sabanciuniv.edu**, then the file name must be: **"atam_the3.cpp"**.

- Please make sure that the files are the latest versions of your THE program.

- You should upload the sample txt files, the header you implemented and the main cpp file with its new name.

- Do not zip any of the documents but upload them as separate files only.

- Submit your work **through SUCourse+ only**!

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

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!
Elif Pınar Ön