CS 235 Lab 4 Specifications

The Railroad Station

Introduction

In this lab we hope to help you achieve a deeper understanding of stacks, queues, and deques, and how to think through and write test cases for your own code. In order to facilitate this a test driver will not be provided for Lab 4. Rather you will have to think about how you might test and verify your code using a main function. Because there will be no test driver, submitting this lab will be different from the other labs. We will outline specific requirements for the main function to help guide you and to keep pass-offs simple and uniform. As in other labs, you will likely know before you submit whether your code is working properly, but the burden will be on you to think through and explore different test cases as you write and test your code. Note that although there will not be a test driver provided, you must still inherit from the given interface files.

The Main Function

As part of the lab, you will write a main function in a separate .cpp file that uses the code you write for Lab 4. The main function should display the following 6 options for the user:

1. Add a car to the station
2. Remove a car from the station
3. Add a car from the station to a storage facility
4. Remove a car from a storage facility and add it to the station
5. Display the cars currently accessible in the station and storage facilities.
6. Quit.

The user should then be prompted to select one of the options. For example, if the user enters a '5' the currently accessible cars from the station and storage facilities should be displayed. As with other labs, **you may not use any predefined data structures** (except for your linked list from Lab 2.)
All operations other than Operation 6 should return the user to the main menu upon completion.

Operation 1: Add a Car to the Station

This option should allow the user to enter a train car represented by a non-negative integer and attempt to add that car to the station. Remember that negative integers are not valid train car IDs and train car IDs are unique integers. You should reject any invalid input (if it's not an int, display a message stating that the car could not be added and return to the main menu.) If the train car cannot be added to the station the user should be alerted with a message explaining that the car could not be added.

Operation 2: Remove a Car from the Station

This option should remove a car from the station. If the station is empty, this option does nothing.

Operation 3: Add a Car from the Station to a Storage Facility

This option should allow the user to choose from among the data structures (stack, queue, or deque) and attempt to add the current car in the station to that data structure. If the car cannot be added to the data structure or there is no car in the station, the user should be alerted. If the car can be added, it should be moved from the station to the appropriate data structure.

Operation 4: Remove a Car from a Storage Facility and Add it to the Station

This option should prompt the user for a car number. Then, if that car is available at an accessible point in one of the data structures, this should attempt to remove the car from the data structure and add it to the station. If the station is already occupied or if the car cannot be accessed, this does nothing and alerts the user.

Operation 5: Display the Cars Currently Accessible in the Station and Storage Facilities

This option should display all of the available cars in the station and the storage facilities. When a user selects this option, a list of the data structures and the cars accessible from those data structures should be displayed. For example, you might display:

Stack: 7
Deque: 6, 5
Queue: 4
Station: empty

Operation 6: Quit

The program displays “Choo choo ciao!” and exits.
(ASCII train art optional but awesome =)

Test Cases

In order to verify that your code works, you should test a variety of cases where cars are added to the station, added to or removed from the different storage facilities, and then removed from the station. It should be straightforward to test these cases once your main function is up and running. This list gives some ideas of possible tests, but you should thoroughly test your code to make sure that it works properly in other situations as well.

Some ideas:

-Add a car to the station, remove the car from the station
-Attempt to remove a car from an empty station

-Attempt to add an invalid car to the station (negative int, non-int, etc.)
-Add a car to the stack, the queue, and both sides of the deque
-Add several cars to the stack, queue, and deque, then remove several cars from each. Check the available cars after each operation to verify that your data structures are functioning properly.
-Attempt to add a car to a full stack, queue, or deque
-Attempt to add a duplicate car to the station
-Attempt to remove a non-existent or inaccessible car from a data structure
-Fill the station and the storage facilities with cars, then remove them one at a time until the station and storage facilities are empty, checking the contents of your data structures after each insertion and removal.

Implementation Notes

For your main function to be graded, it must compile on the CS Lab machines using the g++ compiler. Therefore, it is in your best interest to ensure your main function compiles as described.

You can run your main function using the g++ compiler through the Terminal. Common commands for navigation in the Terminal:

Command	**What it does**
`pwd`	"print working directory" – prints the absolute file path to the directory you are currently in.
`cd filepath`	"change directory" – changes to the directory specified by the given file path. If a directory or file name has spaces, put the file path in quotes.
`cd ..`	Goes up one level in the directory.
`ls`	"listing" – displays the list of files and folders of the current directory.

How to manually compile C++ code with g++:

```
g++ *.h *.cpp –o NameOfPogramToCreate
```

"g++" is the command to compile, "*.h" and "*.cpp" mean to collect all .h's and .cpp's of the current directory, and "-o" is a flag that means the following word will be the name of your executable.
A simple run of your program, after compilation, is done with "./". For example if you named your program "nameOfProgram," you can run it by navigating to its directory, typing the following and hitting enter:

```
./nameOfProgram
```

If you have questions about compiling or running your code through the terminal, feel free to come to the TA office for a demonstration. It's really pretty easy!

Submission Procedures

Because this lab is implemented differently from the others, when you are ready to submit your lab solution make sure that the following activities have been completed:

1. You have verified that your solution compiles on the Lab machines with the g++ compiler.
2. You have tested your own main function against the provided test cases AND OTHERS to ensure that each of your data structures inserts and removes properly.
3. You have a hard copy of your UML diagram on hand.
4. You have compressed your code, interface files, factory, and main function all into preferably a ".zip". When you compress a file, there is a drop down menu of common archive types. Most lab computers default to ".tar.gz" but you can select ".zip" explicitly.

You will be responsible for testing your code and verifying that it functions properly before attempting to pass off. When you come in to pass off, the TA will run a series of tests to test the functionality of your Station class. If the program does not meet the lab requirements, the TA will explain which parts of the lab need to be changed and you will be responsible for fixing any bugs and submitting the lab again.

As a final caution, please be sure to read the entirety of the Lab 4 specifications as well as all of the comments given in the StationInterface. **You are responsible for ensuring your program meets all requirements and complies with all restrictions**, both there and in this document.

Extra Credit

This lab has extra credit. In order to obtain the extra credit, the stack, deque, and queue must all function properly and you must implement the input-restricted deque and

output-restricted deque as outlined in the lab requirements and the StationInterfaceExtra.h file. If your stack, deque, and queue do not pass the tests, you will need to fix them before receiving the extra credit.