Project 3: Student Valet Parking

CPSC 131 Spring 2019

### Introduction

A very prestigious university in the land of forever sunshine, golden sunsets, and flipflops has a wonderful problem to be solved. You see, once the word got out that this university’s incredible graduates are in high demand, students from all around the world ascended on its serene campus seeking this profound knowledge – and they all drove! Now every one of these new arrivals circle the parking lots for that coveted, sometimes mythical open parking space. Fear not, as this astute university is about to pilot a new *Student Valet Parking* initiative, and you get to help. Here’s how it works: Once all the marked parking spaces are occupied, parking lot attendants allow students to drop off their vehicles in exchange for a claim check. At the end of the knowledge-fulling day, students simply present their claim check to an attendant and pick up their vehicle. It’s even rumored it comes with free car washing … hmmm.

Here’s the catch. There’s no room for more cars, so these parking lot attendants squeeze the cars into the long, skinny aisles of existing parking lots. Some of these aisles are so skinny you can’t see past the first parked car! Others have spikes to enforce one-way traffic. 

When a student drops off her car, the parking lot attendant gives her a claim check with a unique claim number, places a copy of the claim number in the car so they can be matched later, and then squeezes the car into one of those skinny aisles. When she returns to pick up her car, the car in front of hers is driven around the lot and placed at the back of the line while another attendant moves all the other cars up one spot. At some point the car at the front will match the claim check and is returned to the student.

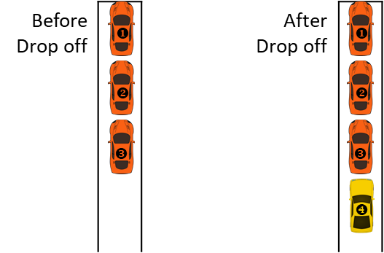
### Objective

The goal of this assignment is to increase queue concepts and proficiency by making a class that wraps around the standard queue ([std::queue](http://www.cplusplus.com/reference/queue/queue/?kw=queue)) class. Do not implement the queue class, just use the standard library’s implementation. Refer to [std::queue](http://www.cplusplus.com/reference/queue/queue/?kw=queue) for a list of class methods.

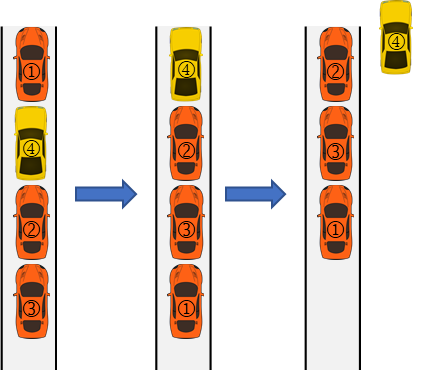
There are three major domain objects:

* **Automobile** – Encapsulates the color, brand, model, and license plate number of the cars parked in the parking lot. Automobile objects are stable, meaning their attributes are specified at construction and they don’t change thereafter. There is no default constructor. You are provided with a complete header file *Automobile.hpp* and a skeleton source file *Automobile.cpp*. You are to implement the sections in the source file marked with “To be completed”.
* **ClaimCheck** – Claim checks have two attributes: a unique claim check number and the automobile associated with the claim. Students are issued a claim check when dropping off their vehicle, and redeem their vehicle with the claim check when picking up. Two claim checks are equal if they have equal claim check numbers. An automobile is associated with a claim check and a unique claim check number is generated at construction. There is no default constructor. Once a claim check is created it cannot be altered. The claim check number and the associated automobile may be queried from a claim check. You are provided with a complete header file *ClaimCheck.hpp* and a skeleton source file *ClaimCheck.cpp*. You are to implement the sections in the source file marked with “To be completed”.
* **ParkingLot** – Parking lots support dropping off and picking up automobiles. A student drops off an automobile and receives a claim check. A student provides a claim check and picks up an automobile. Traffic flow is enforced with one-way traffic spikes, so you cannot back up. Once you enter the aisle you must pull all the way through, if nobody is parked in front of you. You are provided with a complete header file *ParkingLot.hpp* and a skeleton source file *ParkingLot.cpp*. You are to implement the sections in the source file marked with “To be completed”.

Logically, when dropping off a vehicle, the vehicle is parked behind any cars already in the parking lot, as illustrated below.

**

When picking up, vehicles are removed one at a time from the front and added to the back until the desired car is found as illustrated below. If the claimed vehicle is not found in the parking lot, you can either throw an std::invalid\_argument (strongly recommended but not required) or print an error message.

**

Complete the implementation of these classes, and make sure all tests pass. Your code is tested in the provided main.cpp.

Initially the given code may not compile. As you complete the code, the tests should start to pass in main.cpp.

### Source Code Files

You are given complete header and “skeleton” source files with declarations that may be incomplete and without any implementation. Implement the code marked with “To be completed” in the following files and ensure that all tests in main.cpp pass successfully.

* *Automobile.hpp / Automobile.cpp*
* *ClaimCheck.hpp / ClaimCheck.cpp*
* *ParkingLot.hpp / ParkingLot.cpp*
* *main.cpp*: The main function tests your functions. You are encouraged to add additional tests. During grading your main.cpp file will be replaced with the one you were provided with.
* *README.md*: You must edit this file to include your name and CSUF email. This information will be used so that we can enter your grades into Titanium.

### Hints

Make sure your code compiles, and then try and solve the logic. Focus on solving one test at a time. It’s recommended that you start with *Automobile* first, then *ClaimCheck*, followed by *ParkingLot*.

Additional hints are included within sections marked with “To be completed”.

You are encouraged to practice good defensive programming and error checking, but as time grows short focus on the positive path.

### Obtaining and submitting code

We will be using GitHub Classroom to distribute the skeleton code and collect your submissions. This requires you to have an account on github.com. If you are new to GitHub, do the following to get started:

1. Create an account at github.com. You may want to use this account to show a portfolio of your work to prospective employers in the future, so choose something professional.
2. Read Understanding the GitHub Flow and Hello World at GitHub Guides.
3. Read the instructions below for instructions on how to test.

Once you understand the basic operation of git, click the assignment link to fork your own copy of the skeleton code to your PC.

Do not fork your repository to your personal github account (instructors have admin access to private repositories under https://github.com/CSUF-CPSC-131-Spring2019/). Your code should have a URL like https://github.com/CSUF-CPSC-131-Spring2019/project1-brians, NOT https://github.com/brian/project1-brians.

<https://classroom.github.com/a/fa7jFR2A>

Then edit your code locally as you develop it. As you make progress, commit and push your changes to your repository regularly. This ensures that your work is backed up, and that you will receive credit for making a submission. Don’t wait until the deadline to learn how to push code!

### Testing

Unless otherwise directed, use the following command to compile your program:  
**clang++ -g -Wall -std=c++14 main.cpp -o test**

If you included separate implementation files your command will need to include the separate implementation files as well (but never the header files):

**clang++ -g -Wall -std=c++14 main.cpp Automobile.cpp ClaimCheck.cpp ParkingLot.cpp -o test**

To attempt to run the compiled test program, use the following command:  
**./test**

### Grading rubric

Your grade will be comprised of two parts, *Form* and *Function*.

*Function* refers to whether your code works properly as tested by the main function (80%).

*Form* refers to the design, organization, and presentation of your code. An instructor will read your code and evaluate these aspects of your submission (20%).

### Deadline

The project deadline is Monday, March 18 before midnight.

You will be graded based on what you have pushed to the main branch of your GitHub repository as of the deadline. Commits made after the deadline will not be considered. Late submissions will not be accepted.

**Your code must compile/build for it to be tested and graded. If you only complete part of the project, make sure that it compiles before submitting.**