CSCI 311 - Algorithms and Data Structures
Project 2

Assignment Date: Oct 18, 2024
Due Date: 11:59 pm on Nov 18, 2024
Grace Date: 11:59 pm on Nov 21, 2024

In this project, we will simulate takeoffs and landings at a small airport. As in the first project, there is very little direction regarding how to design or write your code. These decisions are mostly left to you. Make sure to **start early** and to **stay organized**. This project is worth a total of 50 points.

C++ code for this project should be submitted both on Canvas and on inginious. More details on the inginious submission are included below. **In addition, you can only submit once your project2 code file on Canvas, and you can only submit at most 3 times your project2 code file on inginious.** Make sure you finish testing your code using the provided tests cases on the server and **only submit your final code on inginious for grading**. All your code must be written in a file called "AirportDriver.cpp". Remember, coding style and comments matter. Poor style or a lack of comments may cost you points!

Plagiarism, or use of ai, once discovered, results into failing this assignment with a zero point and a referral.

There will be time in lab to discuss these problems in small groups and I highly encourage you to collaborate with one another outside of class. However, you must write up your own solutions **independently** of one another. Do not post solutions in any ways.

Good luck!

1 The Simulation

You are working as part of a consulting team to simulate takeoffs and landings at a small airport so that they can better understand the limitations of their current procedures and, eventually, improve efficiency. The airport has an executable file that allows them to run simulations using their current procedures but has lost the associated cpp file. Unfortunately, no one you are able to contact can fully describe these procedures. However, you have a number of test files (ins and outs) for you to derive the algorithm for the scheduling.

The airport has two runways, *A* and *B*. Aircraft will arrive at or leave from the airport at discrete time steps. More than one aircraft may enter the simulation at each step but only one aircraft may use each of the runways at a time. Each aircraft will have a time at which it enters the simulation, an ID, an indication as to whether it is departing from or arriving at the airport, and a priority.

At each time step,

1. New aircraft(s) is (are) added to the simulation

2. Actions are performed for runways *A* and *B*
3. Time is incremented

It is possible for an aircraft to enter the simulation and leave (takeoff or land) in the same time step.

2 Input and Output

Your program should read input from `cin`. This is intended to allow you to enter test cases by hand or by providing a file as input directly from the terminal. For example, suppose you compile your program to a.out and that you have a file named "test 1.in". Then running `./a.out < test_1.in` will run your program using the lines of "test 1.in" as input. I highly recommend that you use this approach to help save tests and time.

The first line of input contains a single integer *n* indicating the number of aircraft that will arrive at or depart from the airport in the simulation. The following *n* lines each contain information for a single aircraft entering the simulation. In particular, each aircraft will include

1. A time at which it enters the simulation (int, non-negative)
2. An ID (int, unique)
3. An indication as to whether it is departing or arriving at the airport (string, either "departing" or "arriving")
4. A priority (int)

separated by spaces. You can assume that this information is in sorted order with respect to the times that the aircraft enter the simulation. If two or more aircrafts enter the simulation at the same time, they should be considered in the order in which they are given.

At each time step where something happens, your program should print information to the screen. Specifically, output should be organized as follows:

```
Time step <time>
    Entering simulation
        <aircraft information>
        <...>
        <aircraft information>
    Runway A
        <aircraft information>
    Runway B
        <aircraft information>
```

Aircraft information should include everything about an aircraft in the same order as the input separated by spaces. Each indentation is a tab (use the escape sequence "\t"). If nothing happens in a particular section, that section should be left empty but the heading should remain. For example, suppose one aircraft enters the simulation and one takes off from runway *A* at time step 42. Then the output should be

```
Time step 42
    Entering simulation
        42 10 departing 5
    Runway A
```

23 4 departing 2
Runway B

If nothing happens in a given time step, nothing should be printed to the screen.

3 Testing

The test cases are provided in the attached tar file. Your code file must generate the same outputs on the same inputs in the test cases.

4 Code File and Inginious

You must use the attached blank code file "AirportDriver.cpp" and put your solution codes into it according to the directions given in that file. Your final code file "AirportDriver.cpp" should be submitted to inginious for grading in addition to Canvas. Any classes you implement, including airplane and priority queue classes, should be included in this single file.

5 Requirements and Grading

In your file AirportDriver.cpp, you must include a priority queue implementation with the following methods:

- empty() - returns true if the priority queue is empty and false otherwise.
- size() - returns the number of elements in the priority queue.
- push(Object o) - a void method which adds the object o to the priority queue. (Object can be replaced with your own class.)
- pop() - removes and returns the first element of the priority queue.
- peek() - returns, but does not remove, the first element of the priority queue.

You may use the C++ standard library implementation of a queue in your project if you would like, however, you must implement priority queues from scratch. It is probably a good idea to make an Airplane class of some kind. Your priority queue can be written to work specifically with objects of this type.

Grading for this project will be broken down as follows:

- (15 pts) A priority queue implementation with the functionality described above.
- (30 pts) A working simulation of a two-runway airport as described above. (passing the test case on ingenious)
- (5 pts) Coding style and comments describing functions and important chunks of code.

6 Submissions Reminder

- Your AirportDriver.cpp file on inginious.
- Your AiportDrive.cpp file on Canvas.