CSCI 311 - Algorithms and Data Structures
Project 2

Assignment Date: Oct 13, 2023
Due Date: 11:59 pm on Nov 6, 2023
Grace Date: 11:59 pm on Nov 9, 2023

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 structure 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 200 points.

C++ code for this project should be submitted on Canvas and turnin. More details on the turnin submission are included below. All code and tests should be included in a **.zip file** called Project2. Your main function should be written in a file called AirportDriver.cpp. You may, and I recommend that you do, include additional files to help organize your code. Remember, coding style and comments matter. Poor style or a lack of comments may cost you points!

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. Also, please include a list of the people you work with in a comment at the top of your submission.

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. To start the project, your team has tasked you with reverse engineering (without using a decompiler) this executable.

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 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 aircraft 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

As part of this project, you will need to probe the given executable to determine how the original simulation works under different conditions and test your own code to make sure that the simulation is working as expected. Along with .cpp and .h files, you should submit at least 10 tests with expected outputs and short descriptions of what they test. For example, you might include a test file "test_1.in" that looks like this:

```
4
0 1 departing 1
0 2 departing 1
0 3 departing 1
0 4 arriving 3
```

Along with expected output in "test_1.out":

```
Time step 0
    Entering simulation
        0 1 departing 1
        0 2 departing 1
        0 3 departing 1
        0 4 arriving 3
    Runway A
        0 1 departing 1
    Runway B
        0 4 arriving 3
Time step 1
    Entering simulation
    Runway A
        0 2 departing 1
    Runway B
        0 3 departing 1
```

A potential description of this test might be "Testing to see when arriving and departing aircraft use each of the runways. Note that aircraft 3 departs from runway *B* after aircraft 4, the only arriving aircraft, has landed even though aircraft 4 has a higher priority value". Descriptions of test cases can be included in comments at the end your AirportDriver.cpp file.

Think carefully about important scenarios that are worth testing! Approach the problem of reverse engineering the given executable systematically.

4 turnin

Your code should also be submitted to turnin for testing. The tests that you create **should not** be taken from turnin. Note that turnin expects a single file called AirportDriver.cpp to be submitted. Any classes you implement, including airplane and priority queue classes, should be included in this single file. This is to accommodate submissions with different class structures. Your code submitted to Canvas **should not** be organized in this way.

5 Requirements and Grading

Your main must be written in a file called AirportDriver.cpp. You must include a binary heap 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.
- 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:

- (75 pts) A priority queue implementation with the functionality described above.
- (75 pts) A working simulation of a two-runway airport as described above.
- (30 pts) At least 10 test cases with expected outputs and brief descriptions as described above. These tests should be well thought out and should cover important aspects of the simulation.
- (10 pts) Comments describing functions and important chunks of code.
- (10 pts) Organization. (coding style)

6 Submissions Reminder

- Your AirportDriver.cpp on turnin ("project2").
- Your AiportDrive.cpp and 10 test cases (one ".in" and one ".out" for each case) in a zip file on Canvas.