Airport Simulation

Date assigned: **Wednesday, October 31, 2018**
Date due: **Wednesday, November 14, 2018**
Points: **60**

Queue

You are to implement the Queue ADT in a file called **queue.c** using the header file **queue.h**. You can find this header file on zeus in **/home/CS300Public/2018/06Files**. All of the data structures and function prototypes are defined in the header file. Further, each function prototype has been described to the point that you should be able to implement each function. The error codes that can be produced are listed for each function. Higher precedence error codes are listed first.

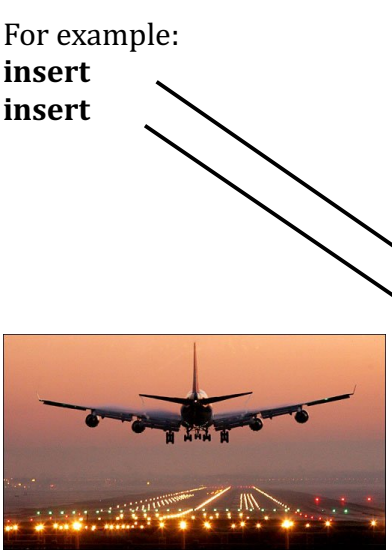
The Queue must be implemented using the Priority Queue from the previous assignment as the base data structure. No new pqueue.h or list.h files are necessary for this assignment. In addition to implementing the Queue data structure, you must provide a Makefile and test driver (queuedriver.c that produces an executable named queuedriver) in a project called **GenericDynamicQ** that thoroughly tests your Queue. The driver should display to the screen a series of SUCCESS or FAILURE messages with enough description that a user can quickly spot broken functionality.

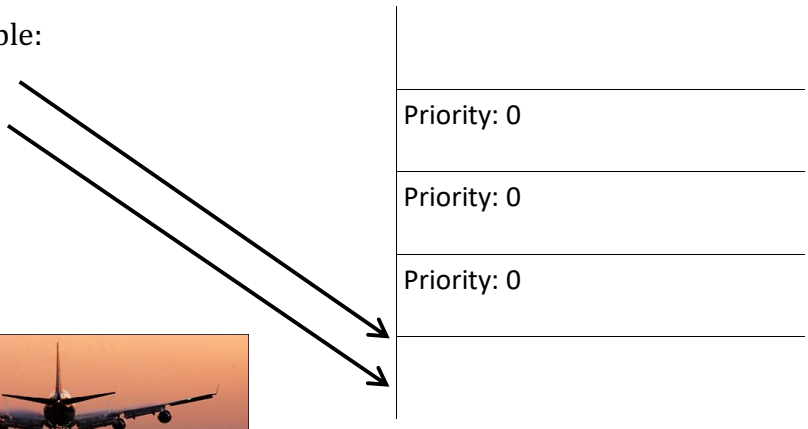
You may add any helper functions you need to queue.c. These helper functions must be marked **static** so they are not available outside the module. You may not alter queue.h in any way.

Masking Priority in the non-Priority Queue.

Your Queue enqueue function must call **pqueueEnqueue** and always provide a priority of zero. This will cause your Queue to default to non-priority queue behavior.

For example:
insert
insert





Airport Simulator

You are to use your Priority Queue and Queue modules to implement an airport simulator. You must simulate runway usage at an airport by determining which planes take off or land on each runway. Each plane that takes off flies to another airport and each plane that lands has taken off from a separate airport.

You will need to write airport.h.

You will need to implement a project named **Airport** that includes an airport module (airport.h, airport.c) that provides all the necessary functionality of the airport and an airportdriver (airportdriver.c) that runs your airport simulation. You may also include a driver that thoroughly tests the functionality of airport.c (airporttestdriver.c). I strongly encourage the creation of the test driver.

Airport Simulation

The airport simulator is a turn-based simulator. Within each turn, a number of events specified below, occur. A clock (an integer) is used to track the number of turns. Each turn takes one clock tick. Your simulator needs to determine which planes land and take off at each turn.

A plane may be sitting on the ground waiting to take off or a plane may be in the air waiting to land. The planes in the air have a non-negative integer amount of fuel. During each turn, the fuel of each in-air plane is reduced by one. Once a plane in the air reaches zero fuel that plane must land before the next turn or that plane will crash. A priority queue, using fuel as priority, must be used to track planes waiting to land. This allows planes to land in priority order (zero fuel has highest priority, 1 unit has the next priority, etc.).

A non-priority queue must be used to track planes waiting to take off. Planes take off in the order in which they entered the system (FIFO). Both queues must store the clock tick (an integer) in which the given plane enters the system.

The airport has three runways. During each turn, each runway may either land exactly one plane or allow exactly one plane to take off. A runway may not both land and launch a plane in the same turn. A runway may also sit idle for a turn if no plane needs to land or take off.

The clock starts at 1.

A turn includes the following events in the following order:

1. Read a line of data from the file “data/airport.txt”. Each line describes airplanes that are joining the takeoff queue and airplanes that have arrived and need to land. Further, the amount of fuel on board for each newly arrived “need to land” plane is provided. Each plane arrives with a positive, non-zero integer amount of fuel. No fuel is assigned to planes that need to take off. At most, 3 planes may join the takeoff queue and an additional 3 planes may arrive needing to land (for a total of 6 new planes in the system per turn). It is also possible that zero planes enter the system at a given turn.
2. Enter the new planes into their appropriate data structures.
3. Decrement each “need to land” plane's fuel by 1 (including those that just arrived).
4. Those planes that “need to land” with a fuel value of zero must be assigned runways for landing. When all three runways are full any remaining planes in the air with zero fuel crash.
5. If step 4 did not use all three runways, the remaining runways are used. Service (land or take off) the plane at the head of the larger queue and remove that plane from its queue. If the queues are the same size, land a plane. Repeat step 5 until all runways are used or both queues are empty.
6. Print the results for the events of this turn.
7. Increment the clock by 1.
8. Return to step 1. Stop the simulation when both the file is exhausted and both queues are empty.
9. Print the summary statistics.

Airport Output

You must output the following table to the screen. Before the first clock tick and after each 20th clock tick reprint the table header (after clock tick 20, 40, 60, etc). The first line of numbers 12345... is for your reference and is not to be printed. Your output must look exactly as below, down to the spaces. There are 0 spaces after the final digit on each line in the table. There are no tabs. Each digit is right aligned. The data files I will run will not produce any digits that overflow too far to the left to disrupt the formatting (at most we will have 9999 clock ticks, for example). The summary statistics are printed with %g or as integers as appropriate. The data file that produced this output is given at the end of the document.

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
     |            Planes Added                |      Runways      |   Queue  Lengths
Time | Takeoff  Landing (Fuel Remaining)  |  1   2   3  Crash | Takeoff  Landing
---- | -------  ------------------------ | --- --- --- ----- | -------  -------
   1 |       3        3 |    1    1    1 |  E   E   E       0 |       3        0
   2 |       3        3 |    2    2    2 |  T   T   T       0 |       3        3
   3 |       0        0 |    -    -    - |  E   E   E       0 |       3        0
   4 |       2        3 |    7    5    9 |  T   T   L       0 |       3        2
   5 |       2        3 |    6    7    5 |  L   T   L       0 |       4        3
   6 |       2        3 |    2    9    4 |  L   T   L       0 |       5        4
   7 |       3        2 |    1    5    - |  E   T   T       0 |       6        5
   8 |       2        2 |    6    2    - |  T   L   T       0 |       6        6
   9 |       1        0 |    -    -    - |  T   L   T       0 |       5        5
  10 |       0        0 |    -    -    - |  L   T   L       0 |       4        3
  11 |       0        0 |    -    -    - |  T   L   T       0 |       2        2
  12 |       0        0 |    -    -    - |  L   T   L       0 |       1        0
  13 |       0        0 |    -    -    - |  T   -   -       0 |       0        0

Average takeoff waiting time: 3.5
Average landing waiting time: 2.73684
Average flying time remaining on landing: 1.31579
Number of planes landing with zero fuel: 7
Number of crashes: 0
```

Key

A – in the Fuel Remaining column means there was no plane added to the system in that position. Time 3 above shows zero planes being added to the landing queue and time 7 shows only two planes being added to the landing queue. Therefore, at time 7 there are two values in the Fuel Remaining columns followed by a –. All dashes must be in the farthest right column as possible.

Runways are marked as L, T, E, or –. A dash means the runway was unused. T means a plane used that runway to take off. E means a plane used that runway to land and the plane had zero fuel remaining (emergency landing). L means a plane used that runway to land and the fuel remaining was greater than zero.

Summary Statistics

You must track the necessary data to produce the above summary statistics. Average takeoff waiting time, average landing waiting time, average flying time remaining on landing (average amount of fuel remaining), and number of planes landing with zero (does not include planes that crashed). The summary statistics include those planes that crash.

When calculating the average waiting and takeoff time, assume that planes have a minimum waiting time of 1. So, even if a plane has just entered the queue and takes off right away, they still have waited 1 turn. The same applies to landing.

Data File
The data file is guaranteed to not be corrupt or invalid. Zeros in various spots are valid. Fuel values may be up to four digits. Airplane crashes do not mean your simulation is not working. Some data files I provide may contain airplane crashes. If your airport simulator crashes that is an entirely different story.

Each **line** in the data file contains the following integers, separated by a single space, in this order:

Number Of New Planes That Want To Takeoff
Number Of New Planes That Want To Land
The amount of fuel on board for new landing plane 1
The amount of fuel on board for new landing plane 2
The amount of fuel on board for new landing plane 3

The amount of fuel is zero if there is no plane in that position.

```
3 3 1 1 1
3 3 2 2 2
0 0 0 0 0
2 3 7 5 9
2 3 6 7 5
2 3 2 9 4
3 2 1 5 0
2 2 6 2 0
1 0 0 0 0
```

- Relying on three previously constructed projects can complicate your life as bugs in other projects are found and fixed. **Start early!**
- For scale, I wrote about 500 (non-commented) lines of code for this project (airport.c, airportdriver.c).
- Your List driver, PriorityQueue driver, and Queue driver must still compile and run.

For each project:

1. Your code is to be written in C using Eclipse. Programs written in other environments will not be graded.
2. The Makefile must contain the necessary targets to build each driver as well as a clean, tarball, and valgrind targets similar to the identically named targets in your previous assignments. Typing **make** on the command line must build each driver.
3. Test one function at a time. This will lessen your level of frustration greatly.
4. You are to use the coding guidelines from the coding standards.
5. The only changes to existing projects is to fix bugs.

Queue

1. **<u>IMPORTANT:</u>** When implementing your queue module, you are to use the functions from the GenericDynamicPriorityQ module and nothing from GenericDynamicList

Airport Simulator

1. Submit a file called **cs300_6_PUNetID.tar.gz** by 9:15am on the due date. This file must include your **Airport** module as well as **GenericDynamicQ**, **GenericDynamicPriorityQ**, and **GenericDynamicList** projects. Each project is to be complete such that I can type make in any of the projects and execute any driver I so desire. Make sure you have all dependencies set correctly and that each Makefile builds the appropriate object files before building the executable. Also, each module's Makefile must have a target called **valgrind** such that typing **make valgrind** executes the valgrind command. I will clean all projects BEFORE typing make in the Airport project.
2. Running valgrind from the command line requires the following type of line in your Makefile

```
valgrind:

   valgrind -v --leak-check=full --show-leak-kinds=all
bin/airportdriver data/airport.txt
```

   Running valgrind from Eclipse requires that you right click on the executable and then go to Profiling Tools Configurations and add data/airport.txt to the arguments tab.

3. Data file input is to be from the command line.
4. Turn in a color, double sided, stapled packet of code by the same deadline in 1. The packet must be in the following order:
    1. airportDriver.c (.h then .c if you have both, otherwise just .c)
    2. airport.c (.h then .c if you have both)
    3. Any extra .h/.c pairs you have. (do not include any code from previous projects)
    4. Makefile
    5. Do NOT print airportTestDriver.c if you write it.