Name: Jacob Isbell
Date: 3/29/2012
Assignment: Program 3.CSC2110

I implemented three class, Airport, Queue and Plane. These three classes handle the majority of the program's processing, except for the main.cpp file which handles basic program looping and creates the instance of class "Airport".

The airport class handles the majority of the program's logic. After being instantiated by the main file, the airport class sets most of its private data members to 0. The airport class then creates the random seed time that will be used later in the program. It then calls its userInput function which asks the user for necessary data to run the simulation. The class then calls the simulationLogic function which handles all of the conditions the program may run into and provides the main logic as to whether to add/remove a plane as well as perform other tasks. After finishing the simulationLogic the Airport class calls printResults which will, as it says, output the results of the simulation.

The queue class is what handles the manipulation of the plane objects. It manages the addition of planes, the dequeueing of planes and can also return a boolean value if it is empty. The queue is in the linked list configuration for storing planes.

Lastly, we have the plane class. The plane class is quite simple. It has an enqueueTime variable so we can later tell when it was created as well an insertEnqueueArrivalTime function to assist in the manipulation of said data.

Pseudo-code of Algorithm:

```
Airport Class:

        Ask the user for the different needed values
    AverageTimeInTakeoff
      Add each plane's TimeInTakeoffQueue and divide by total number of planes.
    AverageTimeInLanding
      Add each plane's TimeInLandingQueue and divide by total number of planes.
    SimulationLogic
      While CurrentTime is less than TotalSimulationTime:
        Landing Probability & Enqueue:
          Generate a random number between 1 and 100
          If the random number is less than or equal to the LandingProb:
            Add a new plane to the landing list.
            Set that plane's EnqueueArrivalTime to CurrentTime.

        Takeoff Probability & Enqueue:
          Generate a random number between 1 and 100.
          If the random number is less than or equal to the LandingProb:
            Add a new plane to the arrival queue.
            Set that plane's EnqueueLeavingTime to CurrentTime.

        If the landing queue is not empty:
          Dequeue the landing queue.
```

If TimeInLandingQueue is greater than FuelTime:
Plane has crashed (hasCrashed == true)
Increment crashed counter.
Otherwise, increment landing counter and add MinutesToLand to CurrentTime.
Else if landing queue is empty an takeoff queue is not empty:
Dequeue takeoff queue.
Increment takeoff counter.
Else:
Increment CurrentTime by one.

Queue Class:
Public:
Constructor
Create a vector of maximum size

Plane Class:

Private:
ArrivalTime is set to CurrentTime when the plane is dequeued from landing queue.
EnqueueArrivalTime is set to CurrentTime when the plane is added to the landing queue.
EnqueueLeavingTime is set to CurrentTime when the plane is added to the takeoff queue.
LeavingTime is set to CurrentTime when the plane is dequeued from takeoff queue.
TimeInLandingQueue is ArrivalTime - EnqueueArrivalTime.
TimeInTakeoffQueue is LeavingTime - EnqueueLeavingTime.
RemainingFuel

Summary:

Design Decisions

I decided to create a plane class 1. because I believe it made handling the plane objects simpler, and 2. I want a stab at the bonus points. I also decided to seperate out some of the functions in the airport class such as the user output and printing results sections. I believe this makes the code much more manageable and readable to both current and future programmers.

Issues

At first I was going to use a vector to store the plane objects but later decided it would be easier (and more efficient) to implement a linked list queue instead due to the fact that a vector does not have a FIFO structure. Other than that, the project design by the team was nearly flawless.

Notes

This program was much more fun and much more reasonable an assignment (and manageable), in my opinion, than the blackjack program. I think this is mainly due to the fact that this is a good example of how to use classes and such whereas the blackjack program didn't need all the required specifications to run efficiently.