Data Structure and Algorithm (CSE2003)

Simulation of Airport using Queues
FINAL REPORT

By;

NAME: P.SANDEEP

REG NO: 18BIS0153

SLOT :B2

Submitted to;

- **Faculty Name: Gopinadth sir**
- **Slot: B2**
- **Class Number: SJT403**

Contents

Abstract

Our aim is simulating an airport in landing and taking off of planes happens simultaneously. In each unit of time, one plane can land or one plane can take off, but not both. That's reason we need to simulate for plane which plane is ready to landing or ready to take-off. We define a queue for landing and also for taking off. When the queue for landing is full, planes are asked to proceed to next runway and the same happens in case of taking off also. If the queue for taking off is full, the plane is asked to proceed to next runway for its takeoff. Apart from this, other data is also calculated using some simple algorithms.

Why we need simulation?

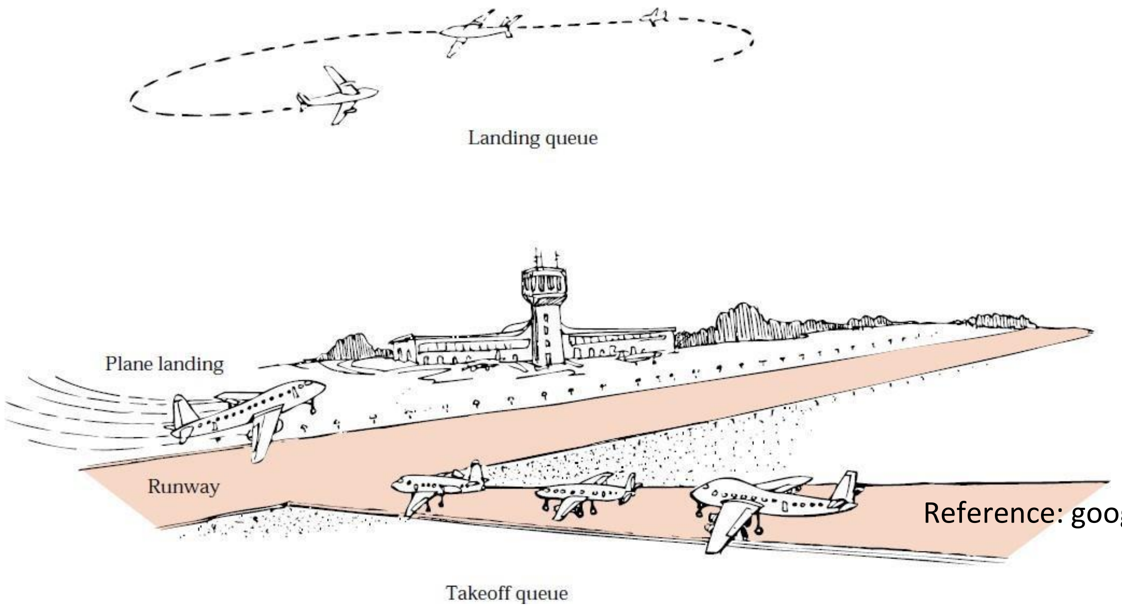
Simulation is the use of one system to imitate the behaviour of another system. Simulations are often used when it would be too expensive or dangerous to experiment with the real system. There are physical simulations, such as wind tunnels used to experiment with designs for car bodies and flight simulators used to train airline pilots. Mathematical simulations are systems of equations used to describe some system, and computer simulations use the steps of a program to imitate the behaviour of the system under study.

Queue:

A **Queue** is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of a **queue** is any **queue** of consumers for a resource where the consumer that came first is served first.

Introduction to our project:

In our Project consider a small but busy airport.



In each unit of time, one plane can land or one plane can take off, but not both. Planes arrive ready to land or to take off at random times, so at any given moment of time, the runway may be idle or a plane may be landing or taking off, and there may be several planes waiting either to land or take off. This is where our program comes into use. It helps the planes to take off and land without any confusion.

When 2 or more planes are waiting to land, and some planes are waiting to take-off it becomes difficult to give instructions for the planes. With this program, we can give proper instructions for planes. This program is made with a priority that, landing occurs before take-off.

RELATED WORKS:

A queue of people at ticket-window: The person who comes first gets the ticket first. The person who is coming last is getting the tickets in last. Therefore, it follows first-in-first-out (FIFO) strategy of queue.

In this Simulation also, we use the same strategy(QUEUES).

MOTIVATION:

Most of the airports are busy. Simultaneously, lot of landing and take-offs take place. So, it becomes difficult to manage airplanes, especially if there are many planes waiting to land or take-off. It becomes difficult for us to find a way by ourselves, without using computer. So, we came up with an idea to design a program which will give proper instructions for the planes, when to take-off and land without any collision. With this program, it will easy to manage an Airport.

PROPOSED METHODOLOGY:

As mention problem we need two queues. To solve problem.
1st queue for arrivals plane (To land).
2nd queue for departure plane (To Take off).

1.) The Runway Class Specification
2.) The Plane Class Specification
3.) Functions and Methods of the Simulation

a. Simulation Initialization
b. Runway Initialization
c. Accepting a New Plane into a Runway Queue
d. Handling Runway Access
e. Plane Initialization
f. Asking a plane to land in next runway
g. Processing an Arriving Plane
h. Processing a Departing Plane
i. Communicating a Plane's Arrival Data
j. Marking an Idle Time Unit
k. Finishing the Simulation

Points to be noted:

1. The same runway is used for both landings and take-offs initially. After the runway gets too busy for either landing or taking off, the planes are asked to use the 2nd runway.
2. One plane can land or take off in a unit of time, but not both.

3. A random number of planes arrive in each time unit.
4. A plane waiting to land goes before one waiting to take off.
5. The planes that are waiting are kept in queues landing and take- off, both of which have a strictly limited size.

Ask to user for input:

1. Up to what number of planes can be waiting to land or take off at any time?
2. How many units of time will the simulation run?
3. Expected number of arrivals per unit time?
4. Expected number of departures per unit time?

Excepted output:

We want to run programme until in given unit of time. **Example:** user give input like,

1. Up to what number of planes can be waiting to land or take off at any time? 5
2. How many units of time will the simulation run? 1000
3. Expected number of arrivals per unit time? .48
4. Expected number of departures per unit time? .48

   In Example simulation time 1000 unit and expected number of arrivals and departures 1 plane per .48-time unit.so we need to do all planes land and take off in one runway to maximum 5 in queues If queues are full show the message like Plane Number N told to try to take-off again later. If plane want to land but in runway plane ready to take-off so in this time give message Plane Number N landed after sometime (approximant base on plane in queue) time unit in the take-off queue. when both queues are empty the runway is idle and simulation continue as a beginning (just for imagine).

Details of Functions and Classes:

1.utility.h

This header file contains:

#include<iostream> for standard iostream operations, #include<limits> for numeric limits,
#include<cmath> for mathematical functions, #include<cstdlib> for Cstring functions, #include<cstddef> for C library language support, #include<fstream> for file input and output, #include<cctype> for character classification, and #include<ctime> for date and time functions.

bool user_says_yes();
And Error exception code Error_code which contain value like
{success,fail,range_error,underflow,overflow,fatal,not_present,duplicate_error, entry_inserted,entry_found,internal_error}

2.random.h

The header files <cstdlib> provide random number generation routines in C++ systems.

Generate random numbers is to start with one number and apply a series of arithmetic operations that will produce another number with no obvious connection to the first. Hence the numbers we produce are not truly random at all, as each one depends in a definite way on its predecessor, and we should more properly speak of *pseudo-random* numbers. The number we use (and simultaneously change) is called the *seed*.

EXAMPLE: To introduce the idea, let us note that saying that an average family has 2.6 children does not mean that each family has 2 children and 0.6 of a third. Instead, it means that, averaged over many families, the mean number of children is 2.6. Hence, for five families with 4, 1, 0, 3, 5 children the mean number is 2.6. Similarly, if the number of planes arriving to land in ten-time units is 2, 0, 0, 1, 4, 1, 0, 0, 0, 1, then the mean number of planes arriving in one unit is 0.9.

Now start with a fixed number called the *expected value v* of the random numbers. Then to say that a sequence of nonnegative integers satisfies a *Poisson distribution* with expected value *v* implies that, over long sub-sequences, the mean value of the integers in the sequence approaches *v*.

Poisson value example: If, for example, we start with an expected value of 1.5, then we might have a sequence reading 1, 0, 2, 2, 1, 1, 3,

0, 1, 2, 0, 0, 2, 1, 3, 4, 2, 3, 1, 1,.... If you calculate the average value for sub-sequences of this sequence, you will find that sometimes the average is less than 1.5 and sometimes it is more, but slowly it becomes more likely to be close to 1.5.

3. plane.h

The class Plane needs to maintain data about particular Plane objects. This data must include a flight number, a time of arrival at the airport system, and a Plane status as either arriving or departing.

Since we do not wish a client to be able to change this information, we shall keep it in private data members. When we declare a Plane object in the main program, we shall wish to initialize these three pieces of information as the object is constructed. Hence, we need a Plane class constructor that has three parameters. Other times, however, we shall wish to construct a Plane object without initializing this information, because either its values are irrelevant or will otherwise be determined. Hence, we really need two constructors for the Plane class, one with three parameters and one with none.

In class Random Public members are:
1.Random(bool pseudo = true); for Declare random-number generation methods. And
2.int poisson(double mean); And private members are:
1.int reseed( );
2.double random_real();
3.int seed, multiplier, add_on for constants for use in arithmetic operations;

Functions of random class
1.int Random :: poisson(double mean) after run this function /* A random integer, reflecting a Poisson distribution with parameter mean, is returned. */

2.int Random :: reseed( ) after run this function /* The seed is replaced by a pseudorandom successor. */

3.Random :: Random(bool pseudo) after run this function /*The values of seed, add_on, and multiplier are initialized. The seed is initialized randomly only if pseudo == false. */
4. double Random :: random_real( ) after run this function /*A random real number between 0 and 1 is returned. */

enum Plane_status {null, arriving, departing};
//for initialize status of plane
Declaration of plane class and its function.
In class Plane
The public member(variable and functions, constructer) are:

1. Plane( );
2. Plane(int flt, int time, Plane_status status);
3. void refuse( ) const;
4. void land(int time) const;
5. void fly(int time) const;
6. int started( ) const; The private members are:

1. int flt_num;
2. int clock_start;
3. Plane_status state;

Function of plane class we define in main file…

4. queue.h

Declare queue data type for handling landing and take-off queue.

Here define queue class and Extended_queue class.

Here, the names *append* and *serve* are used for the fundamental operations on a queue to indicate clearly what actions are performed and to avoid confusion with the terms we shall use for other data types. Other names, however, are also often used for these operations, terms such as *insert* and *delete* or the coined words *enqueue* and *dequeue*.

Error codes are generated by any attempt to append an entry onto a full Queue or to serve an entry from an empty Queue. Thus, our queues will use the same enumerated Error_code declaration as stacks, including the codes success, underflow, overflow.

The operations that we have called empty, append, serve, and retrieve are known in the standard library as empty, push, pop, and front.

We use Extended_queue class for three more operations that are very useful for queues. The first is, which takes a queue that has already been created and makes it empty. Second is the function size, which returns the number of entries in the queue. The third is the function serve_and_retrieve, which combines the effects of serve and retrieve.

The class Extended_queue is derived from the class Queue.

const int maxqueue = 10;
In class Queue
Public members are:
1.Queue()
2.bool empty() const
3.error code serve()
4.Error_code append(const Plane &item);
/*If there is space, x is added to the Queue as its rear. Otherwise an Error_code of overflow is returned.*/
5.Error_code retrieve(Plane &item) const;
/*If the Queue is not empty, the front of the Queue has been recorded as x. Otherwise an Error_code of underflow is returned.*/

Protected members are:
1.int count;
2.int front, rear;
3.Plane entry[maxqueue];

we inheritance queue class's member in Extended_queue ans add other member in Extended_queue class.

In class Extended_queue: public Queue //Inheritance queue class Public members are:
1.bool full() const;
//Return true if the Extended_queue is full; return false otherwise.
2.int size() const;
//Return the number of entries in the Extended_queue.
3.void clear();
4.Error_code serve_and_retrieve(Entry &item);
/*Return underflow if the Extended_queue is empty. Otherwise remove and copy the item at the front of the Extended_queue to item and return success.

5. FinalQueue.h

Define all function to make queue data-type. Which we use in our simulation.

In this header file we define functions of queue and Extended_queue We define function:

1. Queue::Queue()
2. bool Queue::empty() const
3. Error_code Queue::append(const Plane &item)
4. Error_code Queue::serve

5. Error_code Queue::retrieve(Plane &item) const
6. int Extended_queue::size() const

Final File (main file):

In main file define runway class and its function, function of initialize, and define different function for class that we make in different header file.

Runway class: The Runway class needs to maintain two queues of planes, which we shall call landing and take-off, to hold waiting planes. It is better to keep a plane waiting on the ground than in the air, so a small airport allows a plane to take off only if there are no planes waiting to land. Hence, our Runway method activity, which controls access to the Runway, will first service the head of the Queue of planes waiting to land, and only if the landing Queue is empty will it allow a Plane to take off.

One aim of our simulation is to gather data about likely airport use. It is natural to use the class Runway itself to keep statistics such the number of planes processed, the average time spent waiting, and the number of planes (if any) refused service. These details are reflected in the various data members of the following Runway class definition.

```
enum Runway_activity {idle, land, takeoff}; In Runway class Public members are:
1.Runway(int limit);
2.Error_code can_land(const Plane &current);
3.Error_code can_depart(const Plane &current);
4.Runway_activity activity(int time, Plane &moving);
5.void shut_down(int time) const; Private members are:
1.Extended_queue landing; //landing object of extended_queue
2.Extended_queue takeoff; //take-off object of extended_queue
3.int queue_limit;
4.int num_land_requests;       // number of planes asking to land
5.int num_takeoff_requests;    // number of planes asking to take off
6.int num_landings;  // number of planes that have landed
7.int num_takeoffs;  // number of planes that have taken off
8.int num_land_accepted;       // number of planes queued to land
9.int num_takeoff_accepted;    // number of planes queued to take off
10.int num_land_refused;       // number of landing planes refused
11.int num_takeoff_refused;    // number of departing planes refused
12.int land_wait;      // total time of planes waiting to land
13.int takeoff_wait;   // total time of planes waiting to take off
14.int idle_time;      // total time runway is idle
```

Functions and Methods of the Simulation

1. Simulation Initialization:
void initialize(int &end_time, int &queue_limit, double &arrival_rate, double &departure_rate)

Before run this function, we need /* *The user specifies the number of time units in the simulation, the maximal queue sizes permitted, and the expected arrival and departure rates for the airport.*

after run this function *The program prints instructions and initializes the parameters* end_time, queue_limit, arrival_rate, *and* departure_rate *to the specified values.*

We use in this function: *utility function* user_says_yes */

Functions of runway

2. Runway Initialization: **11** | P a g e
Runway :: Runway(int limit)

after run this function/ (usage of function) /* *The* Runway *data members are initialized to record no prior* Runway *use and to record the* limit *on queue sizes.* */

3. Accepting a New Plane into a Runway Queue:
1. Error_code Runway :: can_land(const Plane &current) for after run this function/ (usage of function) *If possible, the* Plane current *is added to the landing* Queue*; otherwise, an* Error_code *of* overflow *is returned. The* Runway *statistics are updated.*

In this function we use extended_queue's function.

2. Error_code Runway :: can_depart(const Plane &current)

after run this function/ (usage of function) *If possible, the* Plane current *is added to the takeoff* Queue*; otherwise, an*Error_code *of* overflow *is returned. The* Runway *statistics are updated.*

In this function we use extended_queue's function.

4. Handling Runway Access:

Runway_activity Runway :: activity(int time, Plane &moving) after run this function/ (usage of function) *If the landing* Queue *has entries, its front*
Plane *is copied to the parameter* moving *and a result* land *is returned. Otherwise, if the takeoff* Queue *has entries, its front* Plane *is copied to the parameter* moving *and a result* take-off *is returned. Otherwise,* idle *is returned.* Runway *statistics are updated.*

In this function we use extended_queue's function.

Functions of Plane class

5. Plane Initialization:

1. Plane :: Plane(int flt, int time, Plane_status status) after run this function/ (usage of function) *The* Plane *data members* flt_num*,* clock_start*, and* state *are set to thevalues of the parameters* flt*,* time *and* status*, respectively.* */

2. Plane :: Plane( ) after run this function/ (usage of function) *The* Plane *data members* flt_num*,* clock_start*,* state *are set to illegal default values.*

6. Refusing a Plane:

void Plane :: refuse( ) const

after run this function/ (usage of function) *Processes a* Plane *wanting to use* Runway*, when the* Queue *is full.*

7. Processing an Arriving Plane:

void Plane :: land(int time) const after run this function *Processes a* Plane *that is landing at the specified time.*

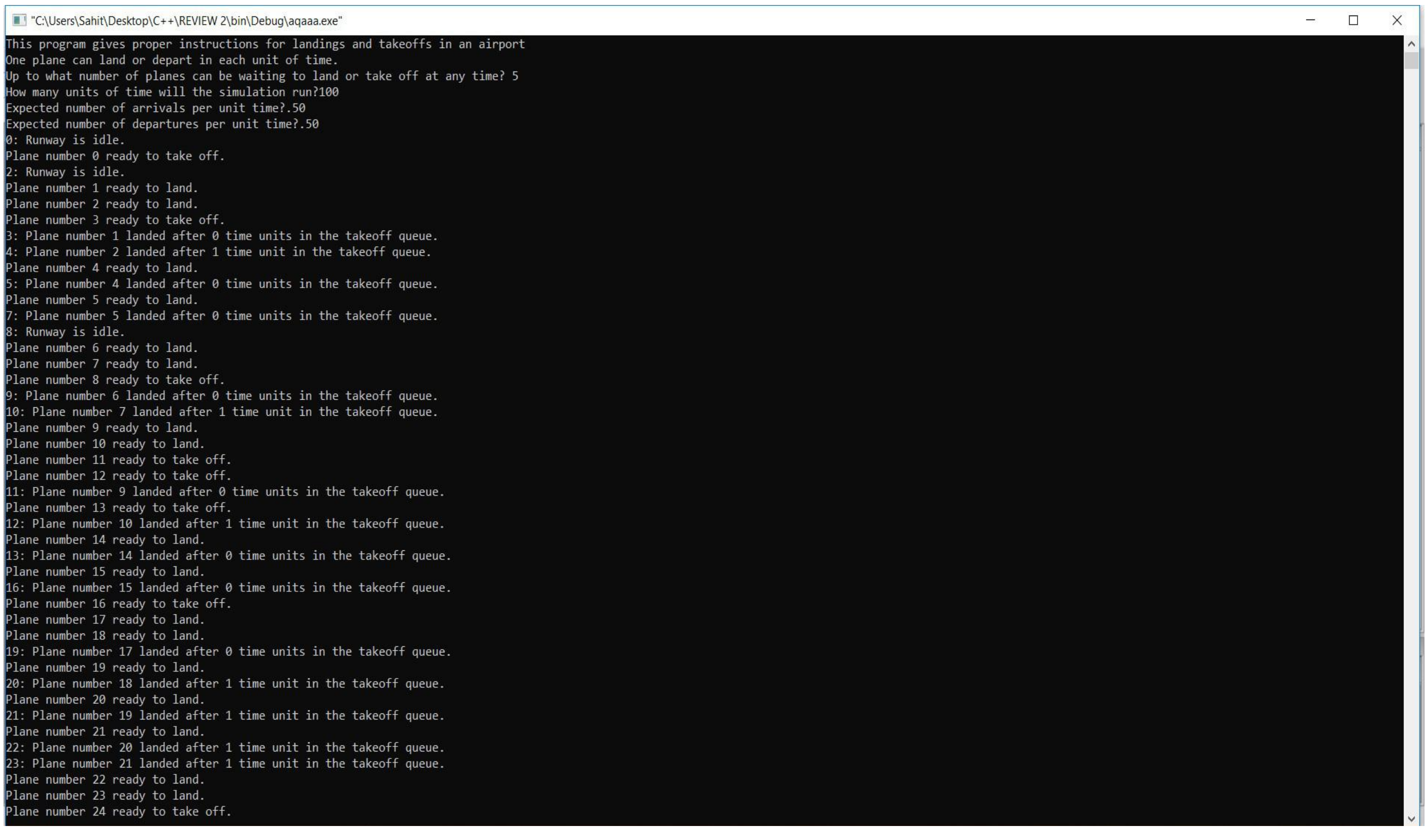
8. Processing a Departing Plane:

void Plane :: fly(int time) const after run this function *Process a* Plane *that is taking off at the specified time.*

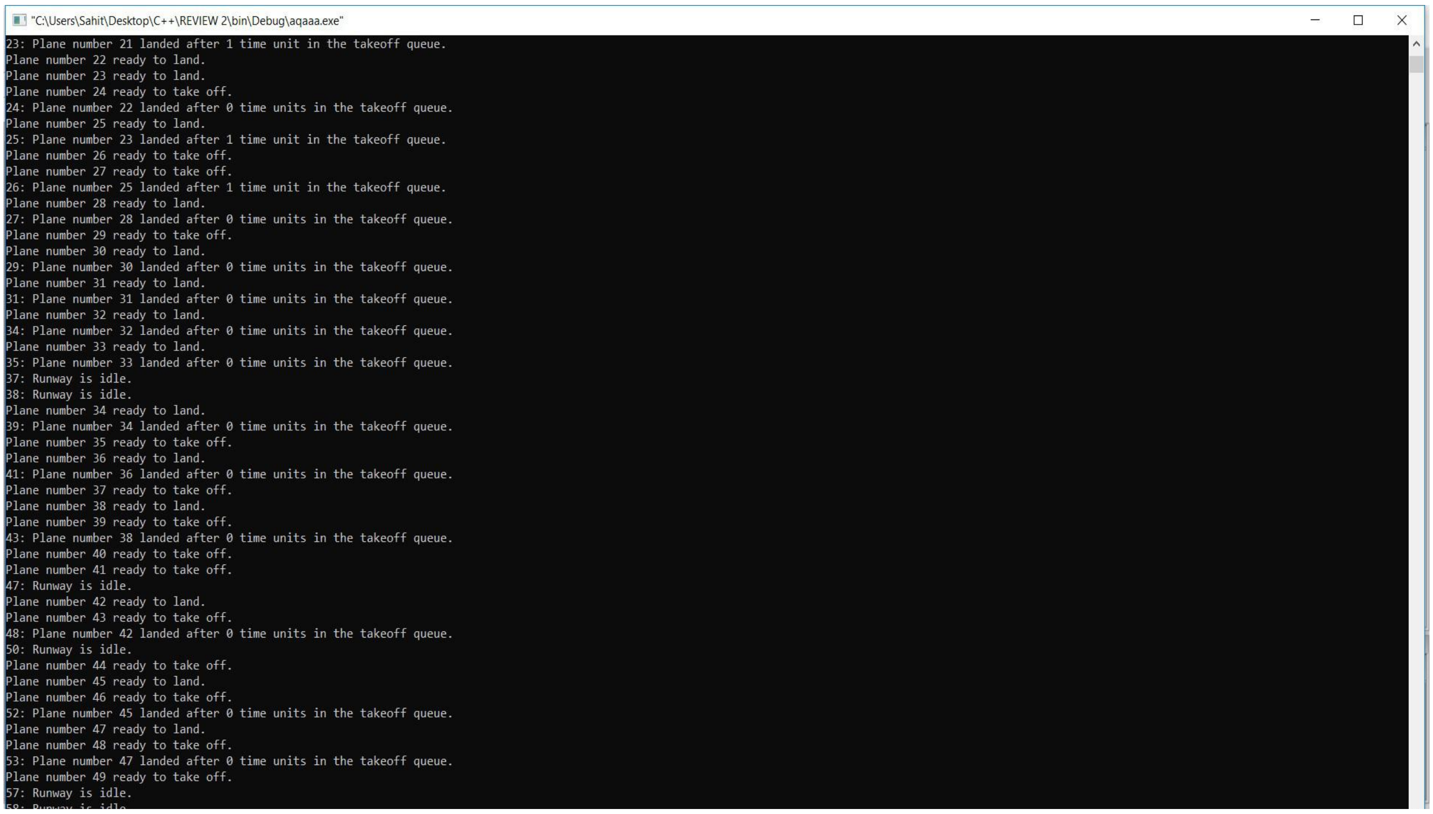
Main Function

Int main()
Before run this function, we need: *The user must supply the number of time intervals the simulation is to run, the expected number of planes arriving, the expected number of planes departing per time interval, and the maximum allowed size for runway queues.* after run this function: *The program performs a random simulation of the airport, showing the status of the runway at each time interval, and prints out a summary of airport operation at the conclusion.* We use in this function: *Classes* Runway, Plane, Random *and functions* run_idle, initialize.

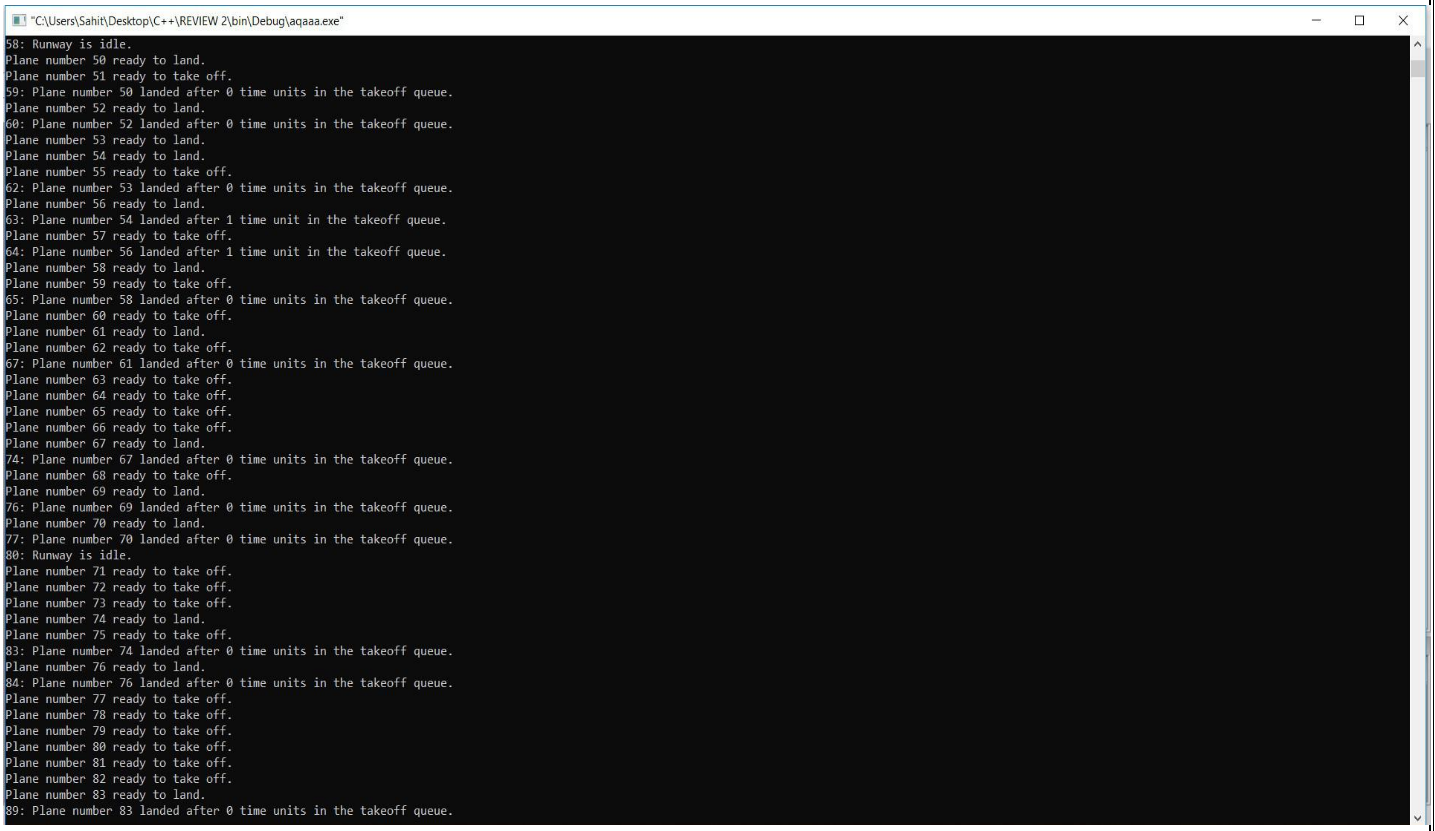
Output: (Images)


"C:\Users\Sahit\Desktop\C++\REVIEW 2\bin\Debug\aqaaa.exe"

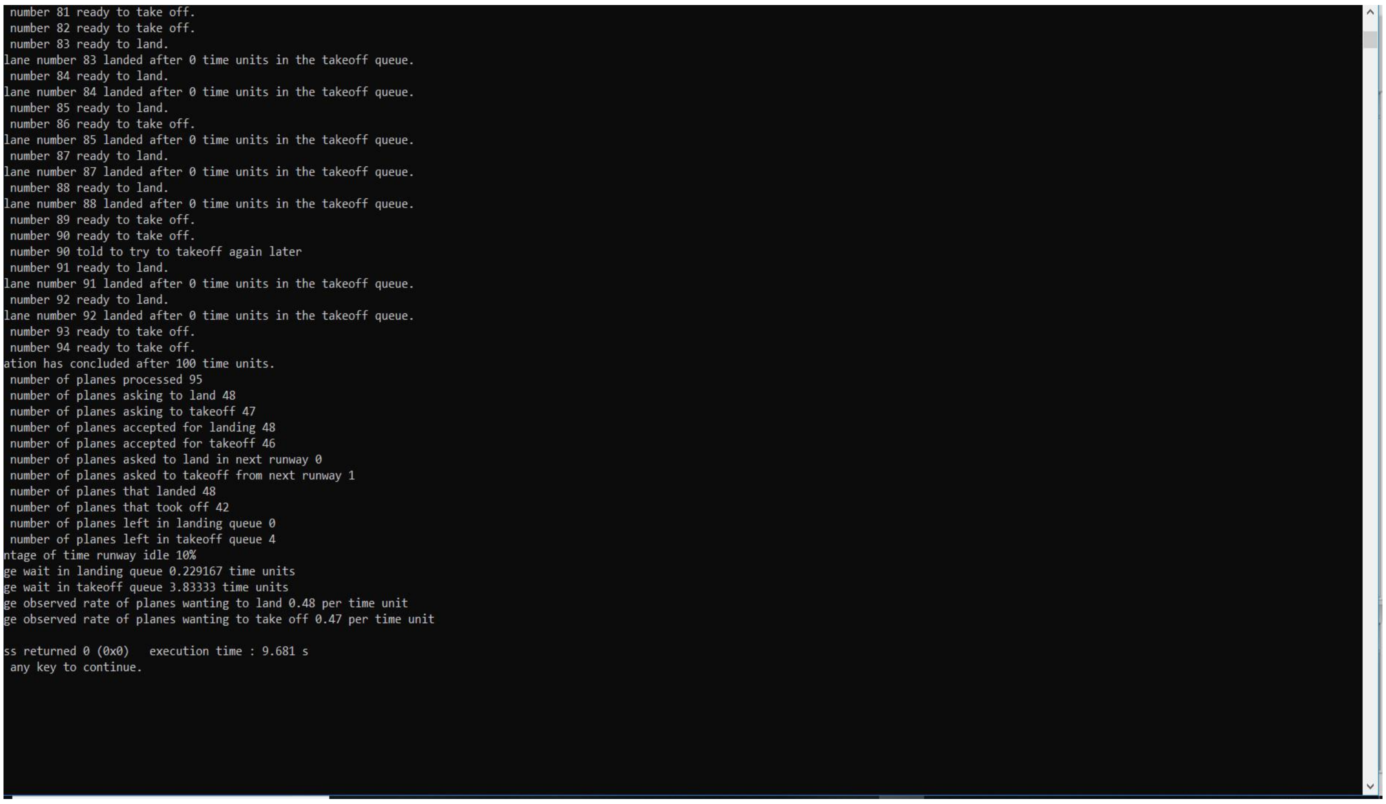
```
This program gives proper instructions for landings and takeoffs in an airport
One plane can land or depart in each unit of time.
Up to what number of planes can be waiting to land or take off at any time? 5
How many units of time will the simulation run?100
Expected number of arrivals per unit time?.50
Expected number of departures per unit time?.50
0: Runway is idle.
Plane number 0 ready to take off.
2: Runway is idle.
Plane number 1 ready to land.
Plane number 2 ready to land.
Plane number 3 ready to take off.
3: Plane number 1 landed after 0 time units in the takeoff queue.
4: Plane number 2 landed after 1 time unit in the takeoff queue.
Plane number 4 ready to land.
5: Plane number 4 landed after 0 time units in the takeoff queue.
Plane number 5 ready to land.
7: Plane number 5 landed after 0 time units in the takeoff queue.
8: Runway is idle.
Plane number 6 ready to land.
Plane number 7 ready to land.
Plane number 8 ready to take off.
9: Plane number 6 landed after 0 time units in the takeoff queue.
10: Plane number 7 landed after 1 time unit in the takeoff queue.
Plane number 9 ready to land.
Plane number 10 ready to land.
Plane number 11 ready to take off.
Plane number 12 ready to take off.
11: Plane number 9 landed after 0 time units in the takeoff queue.
Plane number 13 ready to take off.
12: Plane number 10 landed after 1 time unit in the takeoff queue.
Plane number 14 ready to land.
13: Plane number 14 landed after 0 time units in the takeoff queue.
Plane number 15 ready to land.
16: Plane number 15 landed after 0 time units in the takeoff queue.
Plane number 16 ready to take off.
Plane number 17 ready to land.
Plane number 18 ready to land.
19: Plane number 17 landed after 0 time units in the takeoff queue.
Plane number 19 ready to land.
20: Plane number 18 landed after 1 time unit in the takeoff queue.
Plane number 20 ready to land.
21: Plane number 19 landed after 1 time unit in the takeoff queue.
Plane number 21 ready to land.
22: Plane number 20 landed after 1 time unit in the takeoff queue.
23: Plane number 21 landed after 1 time unit in the takeoff queue.
Plane number 22 ready to land.
Plane number 23 ready to land.
Plane number 24 ready to take off.
```


"C:\Users\Sahit\Desktop\C++\REVIEW 2\bin\Debug\aqaaa.exe"

```
23: Plane number 21 landed after 1 time unit in the takeoff queue.
Plane number 22 ready to land.
Plane number 23 ready to land.
Plane number 24 ready to take off.
24: Plane number 22 landed after 0 time units in the takeoff queue.
Plane number 25 ready to land.
25: Plane number 23 landed after 1 time unit in the takeoff queue.
Plane number 26 ready to take off.
Plane number 27 ready to take off.
26: Plane number 25 landed after 1 time unit in the takeoff queue.
Plane number 28 ready to land.
27: Plane number 28 landed after 0 time units in the takeoff queue.
Plane number 29 ready to take off.
Plane number 30 ready to land.
29: Plane number 30 landed after 0 time units in the takeoff queue.
Plane number 31 ready to land.
31: Plane number 31 landed after 0 time units in the takeoff queue.
Plane number 32 ready to land.
34: Plane number 32 landed after 0 time units in the takeoff queue.
Plane number 33 ready to land.
35: Plane number 33 landed after 0 time units in the takeoff queue.
37: Runway is idle.
38: Runway is idle.
Plane number 34 ready to land.
39: Plane number 34 landed after 0 time units in the takeoff queue.
Plane number 35 ready to take off.
Plane number 36 ready to land.
41: Plane number 36 landed after 0 time units in the takeoff queue.
Plane number 37 ready to take off.
Plane number 38 ready to land.
Plane number 39 ready to take off.
43: Plane number 38 landed after 0 time units in the takeoff queue.
Plane number 40 ready to take off.
Plane number 41 ready to take off.
47: Runway is idle.
Plane number 42 ready to land.
Plane number 43 ready to take off.
48: Plane number 42 landed after 0 time units in the takeoff queue.
50: Runway is idle.
Plane number 44 ready to take off.
Plane number 45 ready to land.
Plane number 46 ready to take off.
52: Plane number 45 landed after 0 time units in the takeoff queue.
Plane number 47 ready to land.
Plane number 48 ready to take off.
53: Plane number 47 landed after 0 time units in the takeoff queue.
Plane number 49 ready to take off.
57: Runway is idle.
```



```
58: Runway is idle.
Plane number 50 ready to land.
Plane number 51 ready to take off.
59: Plane number 50 landed after 0 time units in the takeoff queue.
Plane number 52 ready to land.
60: Plane number 52 landed after 0 time units in the takeoff queue.
Plane number 53 ready to land.
Plane number 54 ready to land.
Plane number 55 ready to take off.
62: Plane number 53 landed after 0 time units in the takeoff queue.
Plane number 56 ready to land.
63: Plane number 54 landed after 1 time unit in the takeoff queue.
Plane number 57 ready to take off.
64: Plane number 56 landed after 1 time unit in the takeoff queue.
Plane number 58 ready to land.
Plane number 59 ready to take off.
65: Plane number 58 landed after 0 time units in the takeoff queue.
Plane number 60 ready to take off.
Plane number 61 ready to land.
Plane number 62 ready to take off.
67: Plane number 61 landed after 0 time units in the takeoff queue.
Plane number 63 ready to take off.
Plane number 64 ready to take off.
Plane number 65 ready to take off.
Plane number 66 ready to take off.
Plane number 67 ready to land.
74: Plane number 67 landed after 0 time units in the takeoff queue.
Plane number 68 ready to take off.
Plane number 69 ready to land.
76: Plane number 69 landed after 0 time units in the takeoff queue.
Plane number 70 ready to land.
77: Plane number 70 landed after 0 time units in the takeoff queue.
80: Runway is idle.
Plane number 71 ready to take off.
Plane number 72 ready to take off.
Plane number 73 ready to take off.
Plane number 74 ready to land.
Plane number 75 ready to take off.
83: Plane number 74 landed after 0 time units in the takeoff queue.
Plane number 76 ready to land.
84: Plane number 76 landed after 0 time units in the takeoff queue.
Plane number 77 ready to take off.
Plane number 78 ready to take off.
Plane number 79 ready to take off.
Plane number 80 ready to take off.
Plane number 81 ready to take off.
Plane number 82 ready to take off.
Plane number 83 ready to land.
89: Plane number 83 landed after 0 time units in the takeoff queue.
```



```
 number 81 ready to take off.
 number 82 ready to take off.
 number 83 ready to land.
lane number 83 landed after 0 time units in the takeoff queue.
 number 84 ready to land.
lane number 84 landed after 0 time units in the takeoff queue.
 number 85 ready to land.
 number 86 ready to take off.
lane number 85 landed after 0 time units in the takeoff queue.
 number 87 ready to land.
lane number 87 landed after 0 time units in the takeoff queue.
 number 88 ready to land.
lane number 88 landed after 0 time units in the takeoff queue.
 number 89 ready to take off.
 number 90 ready to take off.
 number 90 told to try to takeoff again later
 number 91 ready to land.
lane number 91 landed after 0 time units in the takeoff queue.
 number 92 ready to land.
lane number 92 landed after 0 time units in the takeoff queue.
 number 93 ready to take off.
 number 94 ready to take off.
ation has concluded after 100 time units.
 number of planes processed 95
 number of planes asking to land 48
 number of planes asking to takeoff 47
 number of planes accepted for landing 48
 number of planes accepted for takeoff 46
 number of planes asked to land in next runway 0
 number of planes asked to takeoff from next runway 1
 number of planes that landed 48
 number of planes that took off 42
 number of planes left in landing queue 0
 number of planes left in takeoff queue 4
ntage of time runway idle 10%
ge wait in landing queue 0.229167 time units
ge wait in takeoff queue 3.83333 time units
ge observed rate of planes wanting to land 0.48 per time unit
ge observed rate of planes wanting to take off 0.47 per time unit

ss returned 0 (0x0)   execution time : 9.681 s
 any key to continue.
```

Code:

1.utility.h

```
#include<iostream> // standard iostream operations
#include<limits> // numeric limits
#include<cmath> // mathematical functions
#include<cstdlib> // C-string functions
#include<cstddef> // C library language support
#include<fstream> // file input and output
#include<cctype> // character classification
#include<ctime> // date and time functions

bool user_says_yes();
enum Error_code {success,fail,range_error,underflow,overflow,fatal,not_present,duplicate_error,entry_inserted,entry_found,internal_error};
```

2.random.h

```
class Random
{
   public:
      Random(bool pseudo = true);
      // Declare random-number generation methods here.
      int poisson(double mean);
   private:
      int reseed( );
      double random_real();
      int seed,
         multiplier, add_on;
      // constants for use in arithmetic operations
};

int Random :: poisson(double mean)
/* Post: A random integer, reflecting a Poisson distribution with parameter mean,
is returned. */
{
   double limit = exp(-mean);
   double product = random_real( );
   int count = 0;
   while (product > limit)
   {
      count++;
      product *= random_real( );
   }
   return count;
}
int Random :: reseed( )
/* Post: The seed is replaced by a pseudorandom successor. */
{
   seed = seed * multiplier + add_on;
   return seed;
}
Random :: Random(bool pseudo)
/* Post: The values of seed, add_on, and multiplier are initialized. The seed is
initialized randomly only if pseudo == false. */
{
   if (pseudo) seed = 1;
   else seed = time(NULL)%INT_MAX;
   multiplier = 2743;
   add_on = 5923;
```

```
}
double Random :: random_real( )
/* Post: A random real number between 0 and 1 is returned. */
{
   double max = INT_MAX + 1.0;
   double temp = reseed( );
   if (temp < 0) temp = temp + max;
      return temp/max;
```

3. plane.h

```
enum Plane_status {null, arriving, departing}; class Plane
{
    public:
        Plane( );
        Plane(int flt, int time, Plane_status status); void refuse(
        ) const; void land(int time) const; void fly(int
        time) const; int started( ) const;
    private: int flt_num; int clock_start;
        Plane_status state;
};
```

4. queue.h

```
#include "utility.h" #include "entry.h"
#include "plane.h"
const int maxqueue = 10; // number of elements in queue

class Queue { public:
Queue();
bool empty() const; Error_code serve();
Error_code  append(const  Plane  &item);  Error_code  retrieve(Plane
&item) const; protected:
int count;
int front, rear;
Plane entry[maxqueue];
};

class Extended_queue: public Queue { public:
bool full() const; int size() const; void
clear();
Error_code serve_and_retrieve(Entry &item);
};
```

5.FinalQueue.h

```
#include "queue.h"
Queue::Queue()
{
 count = 0;
 rear = maxqueue - 1;
 front = 0;
}
bool Queue::empty() const
{
 return count == 0;
}
Error_code Queue::append(const Plane &item)
{
 if (count >= maxqueue)
 return overflow;
 count++;
rear = ((rear + 1) == maxqueue) ? 0 : (rear + 1);
entry[rear] = item;
return success;
}
Error_code Queue::serve()
{
if (count <= 0)
return underflow;
count--;
front = ((front + 1) == maxqueue) ? 0 : (front + 1);
return success;
}
Error_code Queue::retrieve(Plane &item) const
{
if (count <= 0)
return underflow;
item = entry[front];
return success;
}
int Extended_queue::size() const
{
return count;
}
```

Final File (main file):

```
#include "FinalQueue.h"
#include "random.h"
void initialize(int &end_time, int &queue_limit,double &arrival_rate, double &departure_rate)
{
   cout << "This program gives proper instructions for landings and takeoffs in an airport "<<endl<< "One plane can land or depart in each unit of time." << endl;
   cout << "Up to what number of planes can be waiting to land "<< "or take off at any time? " << flush;
   cin >> queue_limit;
   cout << "How many units of time will the simulation run?" << flush;
   cin >> end_time;
   bool acceptable;
   do
   {
      cout << "Expected number of arrivals per unit time?" << flush;
      cin >> arrival_rate;
      cout << "Expected number of departures per unit time?" << flush;
      cin >> departure_rate;
      if (arrival_rate < 0.0 || departure_rate < 0.0)
         cerr << "These rates must be nonnegative." << endl;
      else
         acceptable = true;
      if (acceptable && arrival_rate + departure_rate > 1.0)
         cerr << "Safety Warning: This airport will become saturated. " << endl;
   } while (!acceptable);
}

enum Runway_activity {idle, land, takeoff};
class Runway
{
   public:
      Runway(int limit);
      Error_code can_land(const Plane &current);
      Error_code can_depart(const Plane &current);
      Runway_activity activity(int time, Plane &moving);
      void shut_down(int time) const;
   private:
      Extended_queue landing;
      Extended_queue takeoff;
      int queue_limit;
      int num_land_requests; // number of planes asking to land
      int num_takeoff_requests; // number of planes asking to take off
      int num_landings; // number of planes that have landed
```

```
    int num_takeoffs; // number of planes that have taken off
    int num_land_accepted; // number of planes queued to land
    int num_takeoff_accepted; // number of planes queued to take off
    int num_land_refused; // number of landing planes refused
    int num_takeoff_refused; // number of departing planes refused
    int land_wait; // total time of planes waiting to land
    int takeoff_wait; // total time of planes waiting to take off
    int idle_time; // total time runway is idle
};

Runway :: Runway(int limit)
/* Post: The Runway data members are initialized to record no prior Runway use
and to record the limit on queue sizes. */
{
  queue_limit = limit;
  num_land_requests = num_takeoff_requests = 0;
  num_landings = num_takeoffs = 0;
  num_land_refused = num_takeoff_refused = 0;
  num_land_accepted = num_takeoff_accepted = 0;
  land_wait = takeoff_wait = idle_time = 0;
}

Error_code Runway :: can_land(const Plane &current)
/* Post: If possible, the Plane current is added to the landing Queue; otherwise,
an Error_code of overflow is returned. The Runway statistics are updated.
Uses: class Extended_queue. */
{
Error_code result;
  if (landing.size( ) < queue_limit)
    result = landing.append(current);
  else
    result = fail;
    num_land_requests++;
  if (result != success)
    num_land_refused++;
  else
    num_land_accepted++;
  return result;
}

Error_code Runway :: can_depart(const Plane &current)
/* Post: If possible, the Plane current is added to the takeoff Queue; otherwise, an
Error_code of overflow is returned. The Runway statistics are updated.
Uses: class Extended_queue. */
{
  Error_code result;
  if (takeoff.size( ) < queue_limit)
    result = takeoff.append(current);
  else
    result = fail;
    num_takeoff_requests++;
  if (result != success)
    num_takeoff_refused++;
  else
    num_takeoff_accepted++;
  return result;
}

Runway_activity Runway :: activity(int time, Plane &moving)
{
  Runway_activity in_progress;
  if (!landing.empty( ))
  {
    landing.retrieve(moving);
    land_wait += time - moving.started( );
    num_landings++;
    in_progress = land;
```

```
      landing.serve( );
   }
   else if (!takeoff.empty( ))
   {
      takeoff.retrieve(moving);
      takeoff_wait += time - moving.started( );
      num_takeoffs++;
      takeoff.serve( );
   }
   else
   {
      idle_time++;
      in_progress = idle;
   }
return in_progress;
}

Plane :: Plane(int flt, int time, Plane_status status)
/* Post: The Plane data members flt_num, clock_start, and state are set to the
values of the parameters flt, time and status, respectively. */
{
   flt_num = flt;
   clock_start = time;
   state = status;
   cout << "Plane number " << flt << " ready to ";
   if (status == arriving)
      cout << "land." << endl;
   else
      cout << "take off." << endl;
}

Plane :: Plane( )
/* Post: The Plane data members flt_num, clock_start, state are set to illegal default
values. */
{
   flt_num = -1;
   clock_start = -1;
   state = null;
}

void Plane :: refuse( ) const
/* Post: Processes a Plane wanting to use Runway, when the Queue is full. */
{
   int qwe;
   cout << "Plane number " << flt_num;
   if (state == arriving)
      cout << " directed to another airport" << endl;
   else
      cout << " told to try to takeoff again later" << endl;
}

void Plane :: land(int time) const
/* Post: Processes a Plane that is landing at the specified time. */
{
   int wait = time - clock_start;
   cout << time << ": Plane number " << flt_num << " landed after "<< wait << " time unit" << ((wait == 1) ? "" : "s")<< " in the takeoff
queue." << endl;
}

void Plane :: fly(int time) const
/* Post: Process a Plane that is taking off at the specified time. */
{
   int wait = time - clock_start;
   cout << time << ": Plane number " << flt_num << " took off after "<< wait << " time unit" << ((wait == 1) ? "" : "s")<< " in the takeoff
queue." << endl;
}
```

```
int Plane :: started( ) const
/* Post: Return the time that the Plane entered the airport system. */
{
   return clock_start;
}

void run_idle(int time)
/* Post: The specified time is printed with a message that the runway is idle. */
{
   cout << time << ": Runway is idle." << endl;
}
void Runway :: shut_down(int time) const
/* Post: Runway usage statistics are summarized and printed. */
{
cout << "Simulation has concluded after " << time << " time units." << endl;
cout<< "Total number of planes processed "<< (num_land_requests + num_takeoff_requests) << endl;
cout<< "Total number of planes asking to land "<< num_land_requests << endl;
cout<< "Total number of planes asking to takeoff "<<num_takeoff_requests << endl<< "Total number of planes accepted for landing "<<
num_land_accepted << endl;
cout<< "Total number of planes accepted for takeoff "<< num_takeoff_accepted << endl;
cout<< "Total number of planes asked to land in next runway "<< num_land_refused << endl;
cout<< "Total number of planes asked to takeoff from next runway "<< num_takeoff_refused << endl;
cout<< "Total number of planes that landed "<< num_landings << endl;
cout<< "Total number of planes that took off "<< num_takeoffs << endl;
cout<< "Total number of planes left in landing queue "<< landing.size( ) << endl;
cout<< "Total number of planes left in takeoff queue "<< takeoff.size( ) << endl;
cout << "Percentage of time runway idle "<< 100.0 * ((float) idle_time)/((float) time) << "%" << endl;
cout << "Average wait in landing queue "<< ((float) land_wait)/((float) num_landings) << " time units";
cout << endl << "Average wait in takeoff queue "<< ((float) takeoff_wait)/((float) num_takeoffs)<< " time units" << endl;
cout << "Average observed rate of planes wanting to land "<< ((float) num_land_requests)/((float) time)<< " per time unit" << endl;
cout << "Average observed rate of planes wanting to take off "<< ((float) num_takeoff_requests)/((float) time)<< " per time unit" << endl;
}

int main()
{
   repeat:
   int end_time; // time to run simulation
   int queue_limit; // size of Runway queues
   int flight_number = 0;
   double arrival_rate, departure_rate;
   initialize(end_time, queue_limit, arrival_rate, departure_rate);
   Random variable;
   Runway small_airport(queue_limit);
   for (int current_time = 0; current_time < end_time; current_time++)
   {
      // loop over time intervals
      int number_arrivals = variable.poisson(arrival_rate);
      // current arrival requests
      for (int i = 0; i < number_arrivals; i++)
      {
         Plane current_plane(flight_number++, current_time, arriving);
         if (small_airport.can_land(current_plane) != success)
            current_plane.refuse( );
      }
      int number_departures = variable.poisson(departure_rate);
      // current departure requests
      for (int j = 0; j < number_departures; j++)
      {
         Plane current_plane(flight_number++, current_time, departing);
         if (small_airport.can_depart(current_plane) != success)
            current_plane.refuse( );
      }
      Plane moving_plane;
      switch (small_airport.activity(current_time, moving_plane))
      {
         // Let at most one Plane onto the Runway at current_time.
         case land:
            moving_plane.land(current_time);
            break;
         case takeoff:
            moving_plane.fly(current_time);
            break;
         case idle:
            run_idle(current_time);
      }
   }
   small_airport.shut_down(end_time);
   return 0;
}
```

Referred Papers: (FOR HELP AND CODING)

- https://practice.geeksforgeeks.org/problems/give-real-life-example-of-stack-and-queue
- https://www.flexsim.com/airport-simulation/
- https://www.google.com/search?q=works+related+to+airport+simulation&rlz=1C1CHBF_enIN819IN819&oq=works+related+to+airport+simulation&aqs=chrome..69i57.9594j0j4&sourceid=chrome&ie=UTF-8
- http://www.cs.utsa.edu/~javalab/cs1723/recitations/SimulationCaseStudy/SimulationCaseStudy.html
- http://ww1.sersc.org/journals/IJAST/vol115/12.pdf
- https://www.researchgate.net/publication/221527178_Simulation_of_passenger_check-in_at_a_medium-sized_US_airport