**CSCE 1101-01 Spring 2023 Term Project Report**

**Airport Simulation**

Habiba El-Sayed, Youssef Badawy, Yasmina Mahdy

CSCE 1101, Section 1

Department of Computer Science and Engineering, AUC

**Abstract:**

This project aims to simulate an airplane queue in an airports management system by writing a program that includes two classes, an Airplane class and a DEQ class, in addition to the main function titled Simulation Program. In the simulation, only one runway exists where airplanes have to wait for permission to land until the airplane ahead of them in the linked list created by the DEQ class is done with servicing. To make the program more realistic, the urgency of the landing of a specific plane was taken into consideration so that an airplane that is marked as urgent would be prioritized in the linked list as if it is a priority queue. Most of the attributes of the plane and the probability of having a plane arriving at a specific time were computed by random functions which makes the topic of randomized variables the main investigation of this study. Moreover, it is observed how differently a randomized variable is generated when the rand and srand functions are changed. It is also discussed how different values of randomized variables affect other variables and the simulation itself during runtime.

**Keywords: Airport simulation; Airplane; Queue; DEQ; Singly Linked Lists; Random Functions; Priority Queues; UML; OOP**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1. Introduction**

Queues are essential to the workings of multiple institutions and especially airports. As a plane is about to descend into the runway, it needs to be given permission to land based on the capacity of the runway. It then needs to be added to a queue as it descends so that it can wait to be serviced by a crew according to its position in the queue. Therefore, this project aims to create an airplane queue simulation program for an airport that replicates the intricate and precise workings of a queue regulated by airport management. To create this simulation, reference to object oriented programming, random generating functions, writing and accessing files will be presented. Since an original approach was taken to object oriented programming, an investigation of randomized probabilities and its effect on other randomized functions will also be discussed.

**2. Problem Definition**

* There exists only one runway to handle all airplanes.
* The simulation operates as if according to a ticking clock.
* Arrival time is randomized, meaning that there is a random chance of arrival for each plane.
* Airplanes placed in a DEQ upon arrival.
* Airplane will have to wait based on its order in the queue and the time it takes to service all the planes before it.
* The time taken to service the plane will depend on the number of passengers it has and the current weather in the simulation.
* If a plane is urgent, it should be prioritized in the DEQ. In the case of multiple urgent planes, they are processed according to the order of their arrival
* The program should calculate the overall average wait time, the total number of arrivals and the number of jobs that have not been processed before the simulation ended (Still in the deque)
* There is an option to run the simulation in a manner that replicates real time but with the minutes converted to seconds
* The log of arrivals and processing should be output to a file

**3. Methodology**

To tackle the problem definition of a simulation of a queue of airplanes, an object oriented program was implemented with two classes and a main function, namely DEQ, airplane and simulation program (the main function). The implementation of the functions and the relationships between the classes can be found in the appendices in code and UML formats.

Each of the two classes, Airplane and DEQ resolve a different issue in the problem definition. The Airplane class includes the attributes or the variables that define the plane (such as the ID, number of passengers, etc..) and the functions that set and get these attributes so that they can be accessed by other classes in addition to a function that prints the information of airplanes. Meanwhile, the DEQ class creates pointers of type Airplane in order to create a linked list. These pointers include a pointer to the front of the linked list and its rear. The functions in DEQ therefore implement the linked list by including functions such as “add” to front and rear and “remove” from front and rear. The nodes of the linked list are moreover of type airplane so that each node includes all the attributes associated with the airplane.

To replicate a queue, airplanes are added from the rear when they land and removed from the front when they are serviced. Yet, some airplanes were made to be added from the front if they are labeled as urgent, which is a boolean variable in the Airplane class. A random generating function was also applied here in class Airplane to set the urgent variable where the variable is set to be true if the randomized variable was equal to one, out of 10 possible numbers (10%).

As for the main function of the program, “Simulation Program”, it pieces the previous two classes together by calling each of the functions in those classes when suitable,and calls new airplanes when they arrive and adds them to the queue. It calculates the wait time for each airplane in the queue, and generates random weather conditions 4 times every day, calculating the service time of each plane based on the weather and the number of passengers that it has. Additionally, it simulates the time for the program to make it run as if it is a shorter replica of a real-time program by including functions that read and write the simulation time input by the user.

To make our program more comparable with the ones used inside airports, we had to find a function to have a delay after every loop new release to make our application simulate the program as if in real time. After some research, we discovered the sleep\_for() feature in C++ , which lets in a thread to stop its execution for a time frame. This characteristic is essential to make certain that our application will now maintain the output without any pause. It takes a length parameter, which may be specified in one-of-a-kind time gadgets along with seconds, milliseconds, and microseconds. Sleep\_for() is generally used in situations where the program have to wait a positive amount of time before executing the next event; For example, in a time-based totally utility which include an Airplane Queue line, sleep\_for may be used to simulate the ready time earlier than a plane may be served.\

**4. Data Specifications**

While the simulation is running, some variables associated with duration and some attributes of the airplane were input into the functions either by the user or by a random generated function. The duration of the simulation program (in hh:mm or h:mm format) was input by the user at the beginning of the simulation so that the program can replicate the workings of an airport system as if it is running for this certain amount of time, and the user is asked to input the character ‘Y’ for a real time simulation, or anything else otherwise.

As for other variables, they were generated by random functions. First, two separate variables were generated, an R variable using a random function, and the probability of arrival of an airplanes. Both these variables were either passed or returned to a function that first checks if the R value is less than the probability of arrival. If the condition is true, a plane is arriving; else, no plane is arriving currently. This method of inputting the data regarding the arrival of the airplane was implemented this way so that the arrival of airplanes can be more spaced out so that each airplane would not have to wait a long time to be serviced. After comparing the two variables during the if statement, other variables were computed if the probability indicates that a plane will land. Other attributes that define an airplane in the Airplane class, such as the number of passengers, the weather and the plane ID were also generated by randomized functions. These randomized variables additionally helped create another variable which is the service time that depended on the number of passengers and the weather which changes every six hours.

**5. Experimental Results**

During the generation of the randomized variables,it is important to note here how the srand functions were seeded as this influenced the probability of an airplane arriving at certain times and thus other variables such as the waiting time. The seeding of the srand function was experimented with to observe which seeding presents the most realistic probability of planes arriving. This means that we searched for the most appropriate seeding so that not all planes arrive right after each other and then have a lot of low time in between. The most realistic seeding for this was srand(time(NULL + timeofDay)) where timeofDay is equal to the clockTime, or the time since the simulation started running, added to the time the simulation was started. Nullifying the seed directly would guarantee that each set of random numbers would be different as this makes the function vary according to the internal clock of the compiler. However, as our program runs in a very short time, this variation is not enough to produce a range of different variables. Thus, srand(time(NULL)) was not the best seeding method to generate different random numbers during runtime. Another seeding method was experimented with and that was srand(time(timeofDay)). Although this method provided different results during runtime, the same results would be repeated when the simulation was closed and another one was started. This was because the variable clockTime would reset to 0 after the end of each simulation. After experimenting a last time with srand(time(NULL+timeofDay)), it was concluded that each set of random numbers generated after this seeding would be unique. The addition of the NULL to the timeofDay would guarantee that each randomized variable is unique within the runtime program and when we start a new simulation.

We have also attempted to control the probability of arrival, which is achieved by generating a random variable R and comparing it to another variable (probabilityofArrival), which we sometimes tried to vary and sometimes kept constant. This was to control the chances of arrival so that airplanes would not arrive every single time the function was called, but only if (R<probability). However the numbers generated were completely random with too many outliers, and after much experimentation (we did not document most of it due to time constraints), we decided that this is an advanced issue that needed more time and focus.

**Some of the experimentations**

float R = rand() / float(rand()%32767);

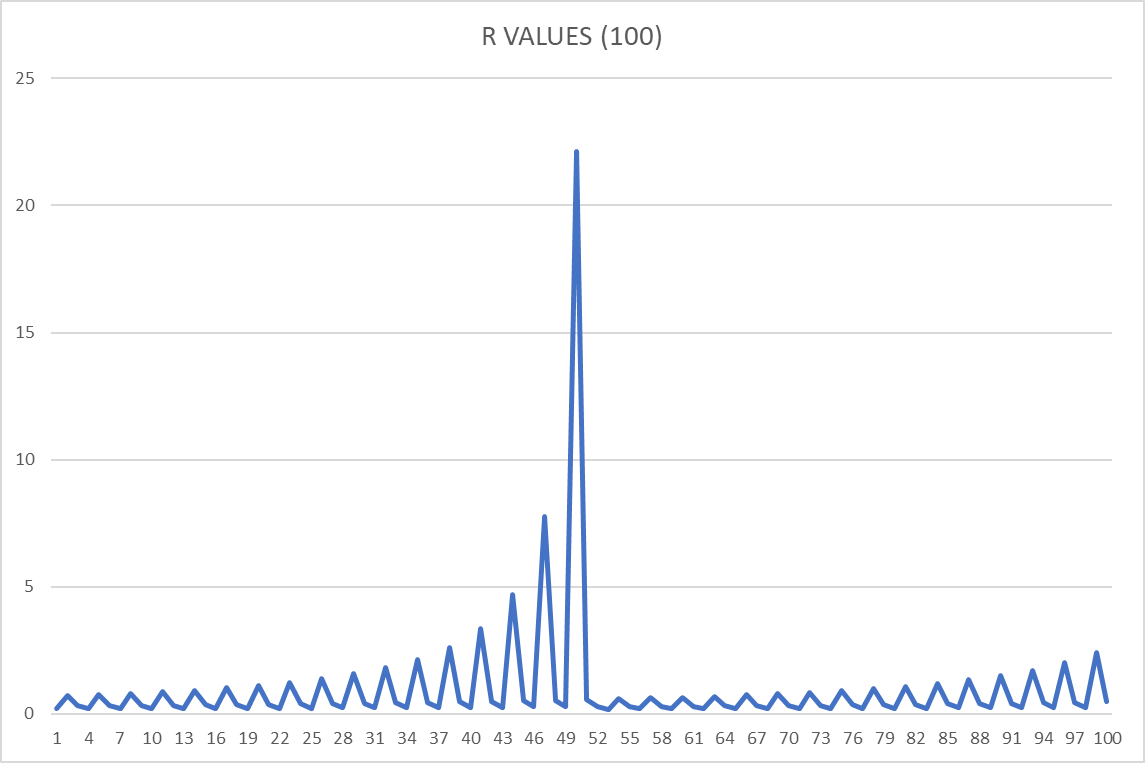
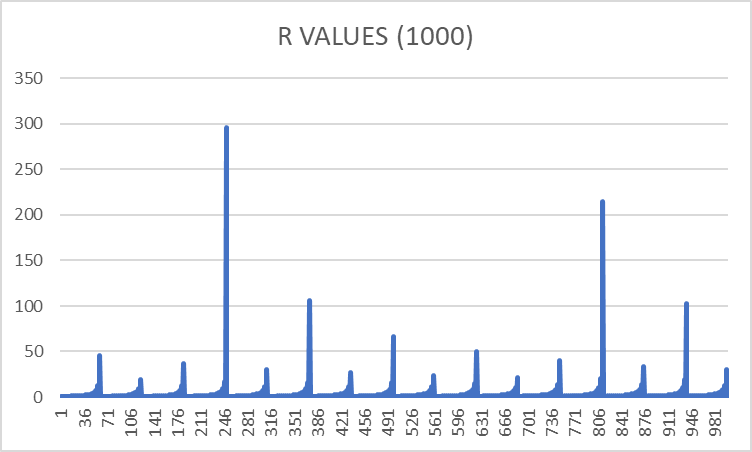


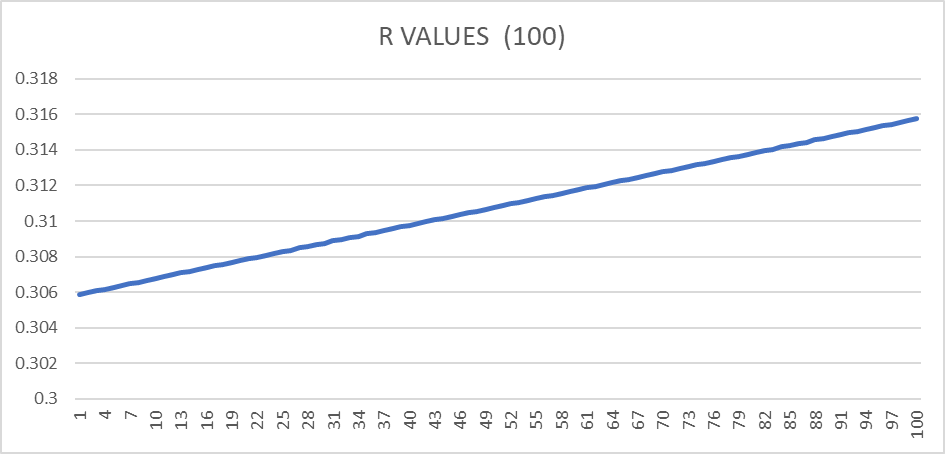
Fig. 1: when

| Mean | 0.339045 |
| --- | --- |
| Median | 0.339045 |

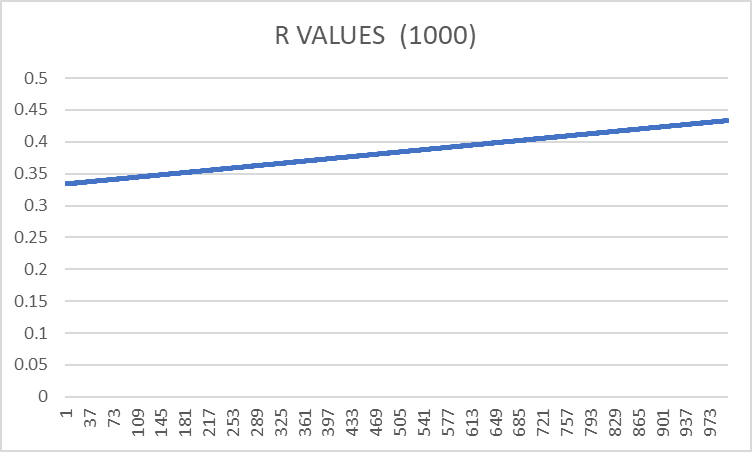


| Mean | 0.7555325 |
| --- | --- |
| Median | 0.7555325 |

float R = rand() / float(32767);

****

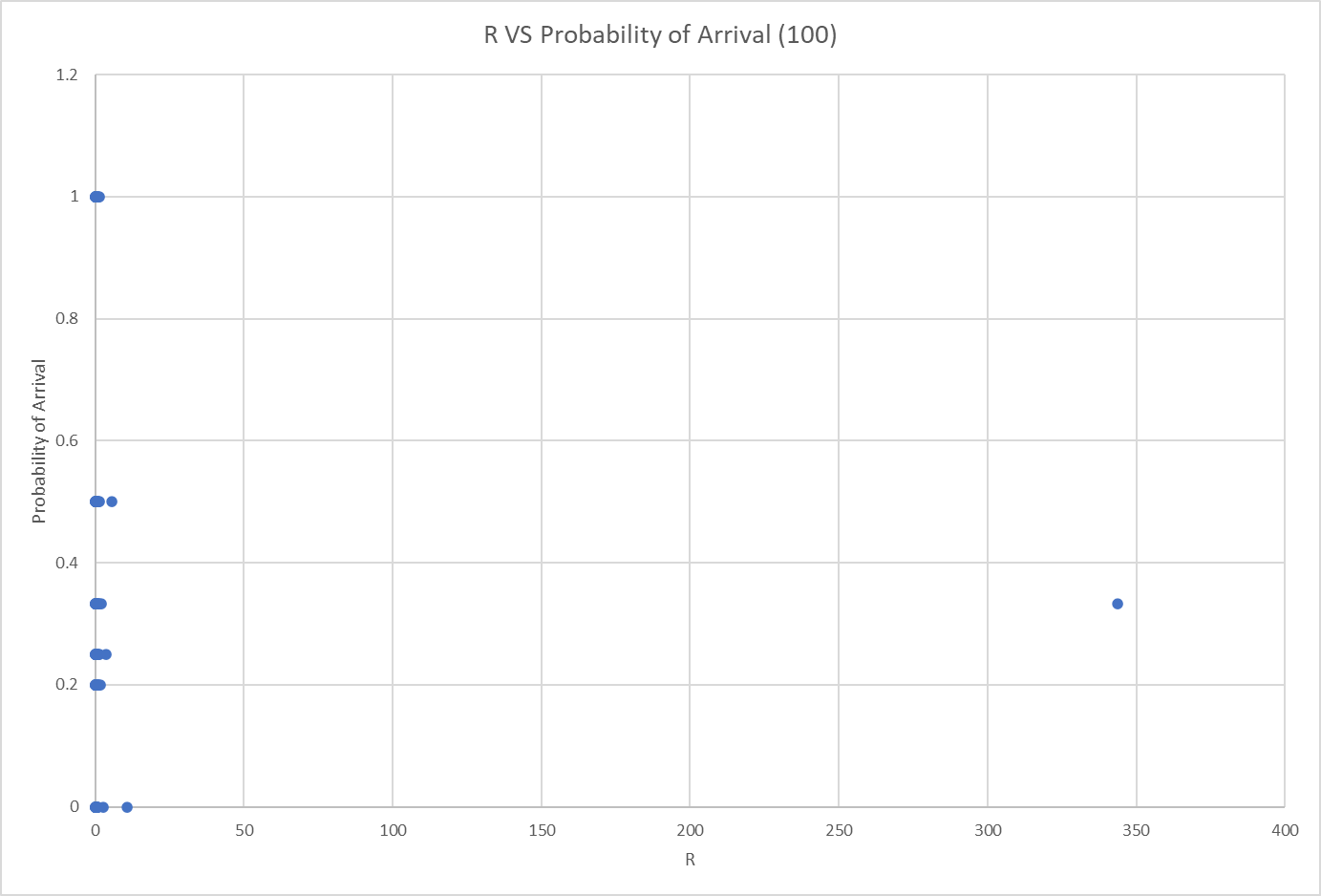
| Max | 0.315744 |
| --- | --- |
| Min | 0.305887 |

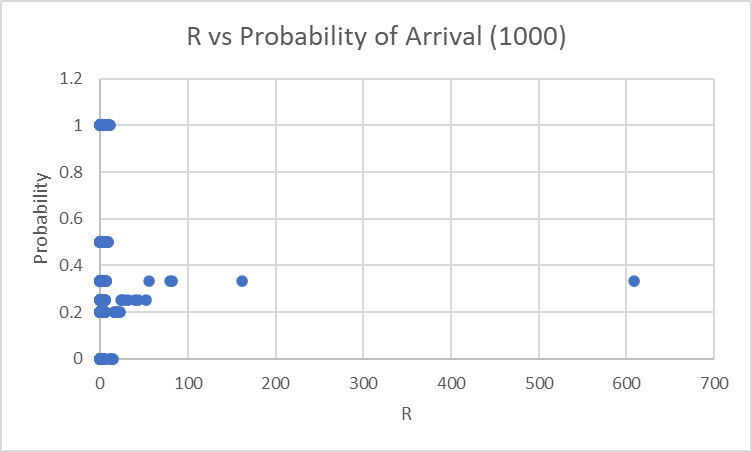
****

| Max | 0.433332 |
| --- | --- |
| Min | 0.333781 |

**float R = rand() / float(rand() % 35000);**

**Probability= 1/rand() % 6;**

****

****

**6. Analysis and Critique**

Although the randomized R value that predicts if an airplane will be arriving at a specific point in time was experimented with in order to determine when it would give the most randomly distributed range of values, the variables generated were often not varied enough. This meant that sometimes a lot of airplanes arrived right after each other and sometimes a long time was spent without a plane arriving. This was an issue that we were unable to resolve, and also meant that we were unable to implement the function timeStatusChange, which was supposed to simulate the prime and low times of the program during preset periods: Prime (12-4pm), Low (2-6am,), and change the probability of arrival accordingly, yet it required a high degree of being able to maneuver the probabilities, which we did not have. Therefore we have decided to output the number of total airplanes and the number of airplanes that were serviced at the end of the simulation run in order to be able to get a general idea of the problem, hopefully to be able to fix it later on.

**7. Conclusion**

Overall, the problem definition was implemented effectively using two classes and a main function. However, the range of the randomized variable needed further experimentation in order to receive more accurate results during the simulation which will enhance the implementation.

**References**

1. <https://stackoverflow.com/questions/12885356/random-numbers-with-different-probabilities#:~:text=You%20can%20easily%20implement%20this,for%20any%20probability%20you%20want>.
2. <https://stackoverflow.com/questions/27295655/generating-different-random-number-in-second-call>
3. <https://stackoverflow.com/questions/7343833/srand-why-call-it-only-once#:~:text=srand%20seeds%20the%20pseudorandom%20number,will%20restart%20the%20same%20sequence>.
4. <https://stackoverflow.com/questions/19921601/random-function-each-time-called-in-c>
5. <https://www.geeksforgeeks.org/csv-file-management-using-c/>

**Appendix: Listing of all Implementation Codes**

**Simulation Program.cpp**

#include <iostream>

#include <cstring>

#include <string>

#include <cstdlib>

#include <ctime>

#include <random>

#include <fstream>

#include <chrono>

#include <thread>

#include "DEQ.h"

#include "DEQ.cpp"

using namespace std;

//global variables

string stime, DayBegin = "6:00"; DEQ line; //bool Prime, Low;

int simTime, timeofDay, clockTime = 0, timeTillService = 0, jobCount = 0, jobTotal = 0, waitTotal = 0;

enum Weather { Sunny, Rainy, Windy, Stormy }; Weather currentWeather; int weatherFactor;

ofstream write("Log.txt", ios::app);

ofstream test("result.csv", std::ios::out | std::ios::app);

//Prototypes

//returns a random value for arrival time, range varies depending on time of the day

int generateArrivalAverage();

//

////generates a random double

float generateRandFloat();

//generates random probability of plane arrival range varies depending on time of the day

//bool generateProbability();

//reads a string time in hh:mm and returns minutes

int readTime(string);

//reads minutes and returns time -> will write later then implement in other functions

string writeTime(int);

//validates that the entered time is correct -> will write later then implement in other functions

void validateTime(string&);

//tests if a plane has arrived and can be entered into the landing

void Arrived(float);

//test if a plane can be serviced

void canService();

//returns true if there is a plane to dequeue, sets its wait time and updates waitTotal and jobCount. Returns false if DEQ is empty

// Calculates the landing time of an airplane based on weather and number of passengers

int calculateLandingTime(Airplane\*);

// Displays the time and weather of the airport

void displayWeather();

// Generates a random weather condition

void generateWeather();

//Prime or Low Time based on current time of day

//bool changeTimeStatus();

void writeToLog(Airplane);

int main()

{

cout << "Welcome to Airport Simulator\n\nDay Begins at " << DayBegin << "\n\nPlease Enter Simulation Duration in hh:mm or h:mm format: ";

cin >> stime;

validateTime(stime);

char real = 'N'; cout << "For Real Time Simulation, Enter the Character 'Y': "; cin >> real;

ofstream clear("Log.txt");

cout << "\n\nSimulation Start\n";

for (int i = 0;i < 30;i++) cout << "---";

clear << "Simulation Start\n"; clear.close();

for (int i = 0;i < 30;i++) write << "---";

write << "\n\n";

//convert to minutes

simTime = readTime(stime);

int averageTime;

float probability = float(1) / generateArrivalAverage();

timeofDay = readTime(DayBegin);

generateWeather();

//if (changeTimeStatus()) probability = float(1) / generateArrivalAverage();

if (real=='Y') {

for (clockTime; clockTime < simTime;clockTime++)

{

timeofDay++;

//if we reached 24:00

if (timeofDay == 1440)

{

timeofDay = 0;

simTime -= clockTime;

clockTime = 0;

}

//if (changeTimeStatus()) probability = float(1) / generateArrivalAverage();

if (timeofDay % 360 == 0) generateWeather();

Arrived(probability);

canService();

if (timeTillService > 0) timeTillService--;

cout << "Current Time: " << writeTime(timeofDay) << "\t Runway Status: ";

if (timeTillService == 0) cout << "Free \r"; else cout << "Occupied\r";

std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

}

}

else {

for (clockTime; clockTime < simTime;clockTime++)

{

timeofDay++;

//if we reached 24:00

if (timeofDay == 1440)

{

timeofDay = 0;

simTime -= clockTime;

clockTime = 0;

}

//if (changeTimeStatus()) probability = float(1) / generateArrivalAverage();

if (timeofDay % 360 == 0) generateWeather();

Arrived(probability);

canService();

if (timeTillService > 0) timeTillService--;

}

}

cout << endl;

for (int i = 0;i < 30;i++) cout << "---";

cout << "\nSimulation complete\n";

for (int i = 0;i < 30;i++) write << "---";

write << "\nsimulation complete\n";

if (jobCount != 0)

{

averageTime = waitTotal / jobCount;

cout << "\nAverage wait time is " << writeTime(averageTime) << ". Total Airplanes: " << jobTotal << ". Airplanes not processed: " << jobTotal - jobCount << "\n\n";

write << "\nAverage wait time is " << writeTime(averageTime) << ". Total Airplanes: " << jobTotal << ". Airplanes not processed: " << jobTotal - jobCount << "\n\n";

}

else

{

cout << "\nNo jobs have been processed\n\n";

write << "\nNo jobs have been processed\n\n";

}

write.close();

test.close();

return 0;

}

int generateArrivalAverage()

{

int T; srand(time(NULL) + timeofDay);

//only works properly if we call srand in the function itself

/\*if (Prime) T = 5;

else if (Low) T = 20;

else T = 10;\*/

T = rand() % 6;

return T;

}

//Y! fix this, or not

float generateRandFloat()

{

//only works properly if we call srand in the function itself

srand(time(NULL) + timeofDay);

//Y! fix probability based on simTime

float T = rand() / float(rand() % 35000);

/\*float T = rand() / float(10000);\*/

/\*(rand() % 40000 + 10000)\*/

//int T = rand() % 5 + 1;

return T;

}

//bool generateProbability()

//{

// //vary based on time of day

// srand(clockTime);

// if (rand() % 100 < 10) return true;

// else return false;

//}

int readTime(string time)

{

int t;

string shr = "", smin = "";

bool colon = false;

for (int i = 0;i < time.length();i++) {

if (time.at(i) != ':' && colon == false) shr += time.at(i);

else if (time.at(i) == ':') colon = true;

else smin += time.at(i);

}

t = stoi(shr) \* 60 + stoi(smin);

return t;

}

string writeTime(int t)

{

string time = "";

if (t / 60 < 10) time = time + '0';

time += to\_string(t / 60) + ':';

if (t % 60 < 10) time = time + '0';

time += to\_string(t % 60);

return time;

}

void validateTime(string& time)

{

//to make sure minutes are not more than 59

if (time.length() == 4 && time.at(2) > (5 +'0'))

{

cout << "\nInvalid Minutes. Please re-enter: "; cin >> time;

}

else if (time.length() == 5 && time.at(3) > (5 + '0'))

{

cout << "\nInvalid Minutes. Please re-enter: "; cin >> time;

}

//to make sure string entered is in h:mm or hh:mm format

while (!(time.length() == 4 && time.at(1) == ':') && time.length() != 5)

{

cout << "\nInvalid time format. Please re-enter: "; cin >> time;

}

}

void Arrived(float probability)

{

//if the probability allows for plane arrival

/\* if (generateProbability())\*/

//srand(clockTime);

float R = generateRandFloat();

probability = float(1) / generateArrivalAverage();

test << R <<", "<<probability << endl;

if ((R < probability))

{

//generates a new plane if arrived;

Airplane\* plane = new Airplane(timeofDay);

//sets arrival time with the current time: DayBegin+ClockTime

plane->setArrivalTime(writeTime(timeofDay));

plane->setServiceTime(calculateLandingTime(plane));

//adds plane to deque

line.addRear(plane);

jobTotal++;

//cout << "\nyes at " << writeTime(clockTime);

cout << "-->Airplane " << plane->getId() << " arrived. ";

if (plane->getUrgent()) cout << "Alert: Urgent Airplane->Moved to priority queue\n\n"; else cout << " \n\n";

}

return;

}

void canService()

{

//if the conditions are met for carrying out the service

if (timeTillService == 0 && !line.DEQisEmpty())

{

Airplane plane;

plane = line.removeFront();

int wt;

if (timeofDay < readTime(plane.getArrivalTime())) wt = 1440 + timeofDay - readTime(plane.getArrivalTime());

else wt = timeofDay - readTime(plane.getArrivalTime());

plane.setWaitTime(writeTime(wt));

waitTotal += wt;

jobCount++;

cout << "Start service: \n";

plane.print();

cout << "\nStarted Service at " << writeTime(timeofDay) << "\nWait Time Before Landing: " << plane.getWaitTime() << "\nEstimated Service Duration: " << writeTime(plane.getServiceTime()) << "\n\n";

writeToLog(plane);

timeTillService = plane.getServiceTime();

}

return;

}

int calculateLandingTime(Airplane\* plane) {

int passNum = plane->getPassNum();

return 10 \* (passNum / float(100)) \* weatherFactor + 2;

}

void generateWeather() {

srand(time(NULL) + timeofDay);

int r = rand() % 6;

switch (r) {

case 0:

case 1:

case 2: currentWeather = Sunny; weatherFactor = 1; break;

case 3: currentWeather = Rainy; weatherFactor = 1.25; break;

case 4: currentWeather = Windy; weatherFactor = 1.75; break;

case 5: currentWeather = Stormy; weatherFactor = 2;

}

displayWeather();

return;

}

void displayWeather() {

switch (currentWeather) {

case 0: cout << "\n\n[Weather Forecast Update] Sunny Conditions.\nEstimated Service Time is Normal\n\n"; return;

case 1: cout << "\n\n[Weather Forecast Update] Rainy Conditions.\n\*Warning: Estimated Service Time is Above Normal\*\n\n"; return;

case 2: cout << "\n\n[Weather Forecast Update] Windy Conditions.\n\*Warning: Estimated Service Time is Above Normal\*\n\n"; return;

case 3: cout << "\n\n[Weather Forecast Update] Stormy Conditions.\n\*Warning: Estimated Service Time is Above Normal\*\n\n"; return;

}

}

void writeToLog(Airplane plane)

{

if (!write.fail()) {

write << "\nAirplane ID: " << plane.getId() << "\nDeparture: " << plane.getDeparture() << "\nNumber of Passengers: " << plane.getPassNum() << "\nArrival Time; " << plane.getArrivalTime();

write << "\nStatus: ";

if (plane.getUrgent()) write << "Urgent\n"; else write << "Not Urgent\n";

write << "Started Service at " << writeTime(timeofDay) << "\nWait Time Before Landing: " << plane.getWaitTime() << "\nEstimated Service Duration: " << writeTime(plane.getServiceTime());

write << "\n\n\n";

}

else cout << "Failed to Open Log File to Write";

}

//bool changeTimeStatus() {

//

// //at 12 pm

// if (timeofDay == 720)

// {

// Prime = true;

// cout << "\n\n\*Warning: Entering Prime Time.\*\nFrequency of Airplane Arrival is Expected to Increase\n\n";

// return true;

// }

//

// //at 4 pm

// else if (timeofDay == 960)

// {

// Prime = false;

// cout << "\n\n\*Warning: Prime Time is Over.\*\nFrequency of Airplane Arrival is Expected to Go Back to Normal\n\n";

// return true;

// }

//

// //at 2 am

// else if (timeofDay == 120)

// {

// Low = true;

// cout << "\n\n\*Warning: Entering Low Time.\*\nFrequency of Airplane Arrival is Expected to Decrease\n\n";

// return true;

// }

//

// //at 6 am

// else if (timeofDay == 360)

// {

// Low = false;

// cout << "\n\*Warning: Low Time is Over. Frequency of Airplane Arrival is Expected to Go Back to Normal\n";

// return true;

// }

//

// return false;

//}

**Header: Airplane.h**

//Y: Added #pragma once to avoid class redefinition

#pragma once

#include <string>

#include <ctime>

#include <cstdlib>

#include <iostream>

#include <cstring>

using namespace std;

//Enumeration to represent the possible destinations

enum Country { Egypt, Kuwait, Saudia\_Arabia, UAE };

class Airplane {

private:

Country departure; // The departure country or airport

int passengerNum; // The number of passengers on the airplane

string id = ""; // The unique identifier for the airplane

string waittime; // The estimated wait time before landing

string arrivaltime; // The estimated arrival time

int serviceTime;

int t;

Airplane\* next = NULL; // A pointer to the next airplane in a linked list

bool urgent = false; // Whether the airplane has an urgent status

static int totalcount;

public:

// Default constructor

Airplane(int);

// Destructor

~Airplane();

// Setter function for the airplane's ID

void setId(string id);

// Getter function for the airplane's ID

string getId();

// Setter function for the airplane's departure country or airport

void setDeparture(string country);

// Getter function for the airplane's departure country or airport

string getDeparture();

// Setter function for the number of passengers on the airplane

void setPassNum(int w);

// Getter function for the number of passengers on the airplane

int getPassNum();

// Setter function for the pointer to the next airplane in a linked list

void setNext(Airplane\*);

// Getter function for the pointer to the next airplane in a linked list

Airplane\* getNext();

// Setter for WaitTime

void setWaitTime(string waittime);

//Getter for WaitTime

string getWaitTime();

// Setter for ArrivalTime

void setArrivalTime(string arrivaltime);

//Getter ArrivalTime

string getArrivalTime();

void setServiceTime(int);

int getServiceTime();

// Getter function for the airplane's urgent status

bool getUrgent();

static int getTotalAirplanes();

//Prints Airplane data

void print();

};

**Airplane.cpp**

#include "Airplane.h"

int Airplane::totalcount = 0;

Airplane::Airplane(int t=0) {

// Default constructor

//generate and set id here with ASCII

//generate and set departure;

/\*

string departure;

int passengerNum;

string id;

\*/

//play with ascii

srand(time(0)+t);

//id+= "B" + char(rand() % 25 + 65) + to\_string(rand() % 899 + 100);

id += "B";

id += char(rand() % 25 + 65);

id += to\_string(rand() % 899 + 100);

this->t = t;

passengerNum = rand() % 200 + 1;

int urg = rand() % 10; if (urg == 1) urgent = true;

totalcount++;

int r = rand() % 4;

switch (r)

{

case 0: departure = Egypt; break;

case 1: departure = Kuwait; break;

case 2: departure = Saudia\_Arabia; break;

case 3: departure = UAE ; break;

}

}

Airplane::~Airplane() {

// Destructor

}

//void Airplane::setId(std::string id) {

// this->id = id;

//}

std::string Airplane::getId() {

return this->id;

}

//void Airplane::setDeparture(std::string country) {

// this->departure = country;

//}

std::string Airplane::getDeparture() {

switch (departure)

{

case 0: return "Egypt";

case 1: return "Kuwait";

case 2: return "Saudia\_Arabia"; break;

case 3: return "UAE";

}

}

//void Airplane::setPassNum(int w) {

// this->passengerNum = w;

//}

int Airplane::getPassNum() {

return this->passengerNum;

}

void Airplane::setNext(Airplane\* n) {

this->next = n;

}

Airplane\* Airplane::getNext() {

return this->next;

}

void Airplane::setWaitTime(string waittime)

{

this->waittime = waittime;

}

string Airplane::getWaitTime() {

return waittime;

}

void Airplane::setArrivalTime(string arrivaltime) {

this->arrivaltime = arrivaltime;

}

string Airplane::getArrivalTime() {

return arrivaltime;

}

bool Airplane::getUrgent() {

//added unsigned int

srand(time(NULL)+t);

//it's gonna be an equal probability of urgent or non-urgent

//urgent shouldn't be that many

//so maybe one in every ten

return /\*this->\*/ urgent;

}

void Airplane::setServiceTime(int t)

{

serviceTime = t;

}

int Airplane::getServiceTime()

{

return serviceTime;

}

int Airplane::getTotalAirplanes() {

return totalcount;

}

void Airplane::print() {

// cout << "\nAirplane ID: " << getId() << endl;

// cout << "Departure: " << getDeparture() << endl;

// cout << "Number of Passengers: " << getPassNum() << endl;

// //cout << "Wait time before landing: " << getWaitTime() << endl;

//cout << "Arrival time: " << getArrivalTime() << endl;

cout << "Airplane ID: " << getId() << "\nDeparture: " << getDeparture() << "\nNumber of Passengers: " << getPassNum() << "\nArrival Time: " << getArrivalTime();

cout << "\nStatus: ";

if (urgent) cout << "Urgent"; else cout << "Not Urgent";

}

**Header :DEQ.h**

//Y: Added #pragma once to avoid class redefinition

#pragma once

#include <iostream>

#include "Airplane.h"

#include "Airplane.cpp"

using namespace std;

class DEQ{

private:

Airplane\* front;

Airplane\* rear;

int length;

public:

DEQ(){front=rear=NULL; length=0;} //

~DEQ() {

if (!DEQisEmpty())

{

Airplane\* prev = front;

while (front != NULL)

{

prev = front;

front = front->getNext();

delete prev;

}

}

}

//please add definition for the getlength function

int getDEQLength();

bool DEQisEmpty();

void addRear(Airplane\* node);//anyone of front/rear

void addFront(Airplane\* node);//

Airplane removeFront();//

Airplane removeRear();//

Airplane\* viewFront();

Airplane\* viewRear();

void viewQueue(Airplane\* node);

};

**DEQ.cpp**

#include "DEQ.h"

int DEQ::getDEQLength() { return length;}

bool DEQ::DEQisEmpty(){

if (length == 0)

return true;

else return false;

}

void DEQ::addFront(Airplane\* airplane){

if (DEQisEmpty()){

front=rear=airplane;

airplane->setNext(NULL);

}

else if (airplane->getUrgent()==true)

{

Airplane\* temp=front;

//temp->getNext() here is null. if the queue is all urgent or there is only 1 airplane

if (temp->getNext() == NULL)

{

if (temp->getUrgent() == true)

{

temp->setNext(airplane);

airplane->setNext(NULL);

rear = airplane;

}

else {

airplane->setNext(front);

front = airplane;

rear = temp;

}

}

else {

temp = front;

Airplane\* before = temp;

while (temp->getUrgent() == true)

{

before = temp;

temp = temp->getNext();

}

airplane->setNext(temp);

if (airplane->getNext() == NULL) rear = airplane;

if (temp != front)

before->setNext(airplane);

else front = airplane;

}

}

else {

airplane->setNext(front);

front=airplane;

}

length++;

}

Airplane DEQ::removeFront(){ //?

if(DEQisEmpty())

cout << "No airplanes currently" << endl;

else if (front->getNext()==NULL)

{

Airplane temp = \*front;

delete front;

front = NULL;

rear = NULL;

length--;

return temp; //return last item

}

else {

Airplane \*second, temp = \*front;

second = front->getNext();

delete front;

front = second;

length--;

return temp; //return last item

}

}

void DEQ::addRear(Airplane\* airplane){

if (airplane->getUrgent()==true)

{

addFront(airplane);

}

else if (DEQisEmpty()){

front=rear=airplane;

length++;

}

else {

rear->setNext(airplane);

rear=airplane;

length++;

}

}

Airplane DEQ::removeRear(){ //make a while loop until next is rear

Airplane\* temp;

Airplane\* temp1;

Airplane sendback;

if(DEQisEmpty())

cout << "No airplanes currently" << endl;

else if(front->getNext() == NULL) {

sendback = \*front;

delete front;

front = NULL;

length--;

return sendback;

}

else{

temp=front;

temp1 = temp;

while(temp->getNext() != NULL) {

temp1=temp;

temp=temp->getNext();

}

sendback = \*temp;

delete temp;

temp1->setNext(NULL);

length--;

return sendback;

}

}

Airplane\* DEQ::viewFront(){

if(DEQisEmpty())

cout << "No airplanes currently" << endl;

else

return front; //-> airplane name ?

}

Airplane\* DEQ::viewRear(){

if(DEQisEmpty())

cout << "No airplanes currently" << endl;

else

return rear; //-> airplane name ?

}

void DEQ::viewQueue(Airplane\* a){

cout << "\n\nBeginning of Queue" << endl;

while (a!=NULL){

a->print();

a=a->getNext();

}

cout << "End of Queue\n\n" << endl;

}