**Airport**

Malek Mahmoud, Marwan Abudaif, and Mazin Bersy

900222057, 900221725, 900221972

The American University of Cairo

The School of Sciences and Engineering

CSCE110102

Dr. Howaida Ismail

May 13, 2023

**Outline**

1. Title Page…………………………………………………………………….1
2. Outline…………………………………..……………………………………2
3. Abstract…………………………………….………………………………....3
4. Introduction…….……………………………………………………………..4

V. Project………………………………………………………………………….5

VI. UML…………………………………………………………………………6

VI. Output………………………………………………………………………….7

VII. Analysis and Critique…………………………………………………………9

IX. Analysis of code………………………………………………………………10

X. Conclusion…………………………………………………………………….13

**Abstract**

The goal of this project was to handle a randomly generated number of airplanes to land at an airport. The project was also asked to make the airport have a capacity of only one airplane at a time. The main objective of this program is to calculate the average wait time an airplane has to wait in the landing queue before it dequeues and lands. In order to implement this, the use of a double-linked list queue was recommended to be used. To make sure this whole program runs with randomly generated numbers, we ran some functions that would generate random numbers for arrival times for each plane, the number of planes, and the wait time for each plane. Moreover, we created a double-linked list queue template class in order for it to be compatible with our plane class. We went a step further also and made sure that within 10 minutes of every plane’s arrival, there would be no clashes. In the end, we achieved the average wait time by adding the wait time of every plane, then dividing it by the number of planes.

Keywords: Airport, Airplane, Queue

**Introduction**

Airports are visited daily by millions of people. They play a critical role in today’s world. A simple shutdown for a day at an airport could cause significant losses in business and transportation. This project represents an airport simulation. It is a project that is built with the foundation of C++ programming. A variety of concepts were implemented in order to run this program, which will be studied and analysed in depth throughout this paper. This paper will demonstrate and illustrate how this project operates, in addition, discuss it in detail

the foundation of it.

**Project**

As mentioned earlier, this project is a simulation of an airport. In specific, it focuses on the landing of airplanes at the airport. We allowed only one airplane present at a time at the airport assuming the airport can only hold one. In this simulation, we put into consideration wait time in order to make it as close as possible to reality. The airport operates for six hours. The base time starts from when the first plane arrives, and along with it the clock. We ran a real-time clock to increment along. This project operates on randomly generated input by the code. Such as number of planes, arrival time, and waiting time. So that by every run we receive different output and numbers. Furthermore, to implement the line of airplanes landing for the six-hour interval we used a template queue class. The queue class contained airplanes at each index. The queue was sorted according to their arrival time by the use of the method of quick sorting.

UML

Here is the UML structure for the project. It displays the classes and the relations they have. In addition to their multiplicity factor. Plus, member variables and functions used in each class.

A screenshot of a computer program

Description automatically generated with low confidence

**Output**

Below are three fully randomized runs.

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

**Analysis and Critique**

The project did not go smoothly, we ran into some issues as we worked. For example, our output used to have some planes landing at the same time, therefore, we had to create a function to deal with any arrival times colliding within 10 mins. Another issue was when we tried sorting with hours and minutes, it was difficult because if a plane lands at 11 pm and the next one at 1 am it read the 11 pm plane as the second one. Hence, we compared with minutes only then afterward turned the minutes into hours when printing. Furthermore, another issue we faced is storing the number of planes created in each plane. In order to overcome this, we created a static int member variable in the class. That is because it saves that value in all plane objects. So, if we increment every time the constructor is called, by the fourth plane the number of planes created will be saved as four in the fourth plane. This helped us tell if the plane created is the first one or not. That is due to planes coming after the first one need to have base time added to their arrival time. The arrival of the first plane is what sets the base time and is where the real-time clock starts from.

**Analysis of code**

The project contains six files. The Queue.h file, which is a template queue class responsible to enqueue and dequeue airplanes in a first in first out manner. A plane.h file that contains the declaration of variables and list of functions in the plane class. Along with its cpp file that possesses definitions of the functions in the .h file. Also, an Airport.h file that holds the member variables and functions used in order to represent an airport. Ofcourse, that goes also along with its cpp file. Lastly, is the Airport system.cpp that acts as our main file of the program and combines everything together. Below are some of the essential functions and variables and their demonstrated use.

**Queue.h:**

It uses a template double-linked list, which means it has pointers pointing backward and forward for all nodes.

isEmpty(): checks if the array is empty

insertRear(T n): inserts a node at the end

getcounter(): returns how many nodes exist in the array

insertFront(T n): inserts node at the front

insertAfter(T v, Node<T>\* curr): inserts node after a specified node given in the parameter

removeRear(): removes node at the end of the array

removeFront(): removes node at the begginign of the array

viewFront(): returns value at the first node

viewRear(): returns value at the end of the node

length(): returns current size of queue

print(): prints arrival time for the plans in the queue

isFull(int t): checks if the queue is full

sort(): calls quicksort to sort the array by comparing arrival times

dealWithClashes(): deals with any clashes between arrival times within 10 mins

**Plane.h:**

static int created;

static int baseTime;

static because we need the value to be carried on to the latest made objects, rather than always starting from zero every time the constructor is called

bool operator>(const Plane& other) const {

return this->arrivalTime > other.arrivalTime;

}

Overloading operator to be used to compare arrival times in planes.

**Plane.cpp:**

Plane(): If it is the first plane, it sets its arrival time at any random time and saves it as base time. However, if it is not then it sets an arrival time within a specified range not too far away from the base time and adds that number to the base time

PrintArrivalTime(): Prints arrival time for the plane

getWaitTime(): randomizes a wait time from 5 to 30 minutes

**Airport.cpp:**

service(): is used in the run sim in order to print the time landed for each plane, its wait time, and what time service starts at

averageWait(): total weight time divide by the number of planes

PopulateAirQueue(): makes a randomized-sized dynamic array of planes and enters them in a queue. Then it sorts the queue of airplanes by their arrival time in minutes. Afterward, it deals with any clashes within 10 minutes of arrival time. Lastly, it sets time and base time equal to the first arrival time

runsim(): calls the service function then displays average wait time

**Conclusion**

Airports are quite essential, as without them, the world cannot connect and come together. They are used on a daily basis and are a key role in the functioning of the world. We were able to achieve a simulation representing a queue of airplanes landing in an airport by order. Another goal we reached was to return the average wait time in every run. Following the instructions, we were able to make this simulation as close as possible to a real-life model. We faced some challenges as we proceeded, but thankfully we overcame them and were able to achieve the simulation needed.