



**PES UNIVERSITY EC Campus**  
**1 KM before Electronic City, Hosur Road,**  
**Bangalore-100**

A Project Report on

## **Airport Simulator using Structure and Queues**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

For the Academic year 2020

by

**Ria Singh(PES2UG19CS326)**

**Siddhi Patil (PES2UG19CS389)**

**Shatakshi Mohan(PES2UG19CS379)**



**PES UNIVERSITY EC Campus**

**1 KM before Electronic City, Hosur Road,  
Bangalore-100**

**Department of Computer Science Engineering**

## **CERTIFICATE**

Certified that the project work entitled **Airport Simulation** carried out by **Siddhi Patil** bearing SRN **PES2UG19CS389** , **Shatakshi Mohan** bearing SRN **PES2UG19CS379** and bearing **Ria Singh** SRN **PES2UG19CS326** ,bonafide students of **IV semester** in partial fulfillment for the award of **Bachelor of Engineering** in PES University during the year **2021**.

The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the course **Secured Programming with C**.

.....  
**GUIDE**

**Mr. Vishwachethan D**

.....  
**Dr. Sandesh**  
**HOD, CSE**

## **Declaration**

I hereby declare that the project entitled “**Airport simulator**” submitted for Bachelor of Computer Science Engineering degree is my original work and the project has not formed the basis of the awards of any degree, associate ship, fellowship or any other similar titles.

**Signature of the Student:** Ria Singh

**Place:** Bengaluru

**Date:** 13/04/2021

**Signature of the Student:** Siddhi Patil

**Place:** Bengaluru

**Date:** 13/04/2021

**Signature of the Student:** Shatakshi Mohan

**Place:** Bengaluru

**Date:** 13/04/2021

# **Airport Simulation**

# **Authors**

Siddhi Patil bearing SRN PES2UG19CS389 in  
PES University

Ria Singh bearing SRN PES2UG19CS326 in  
PES University

Shatakshi Mohan bearing SRN PES2UG19CS379 in  
PES University

# 1.0 Abstract: Problem Definition

---

There is a small busy airport with only one runway. 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 unit of time, the runway may be idle or a plane may be landing or taking off.

There may be several planes waiting either to land or to take off. Number of planes waiting to take off or land should not exceed a certain fixed limit.

Landing planes should have higher priority than taking-off planes (since it is better to make a plane wait on land rather than in air).

After running for some period, the program should print statistics such as: number of planes processed, landed, and took off, refused; average landing and take-off waiting times, and average run-way idle time. Number of planes waiting to take off or land should not exceed a certain fixed limit.

# Table of Contents

---

1.0 Abstract: Problem Definition

2.0 Introduction

3.0 Requirement Specification

3.1 Establishment

3.2 Risk Assessment

4.0 Software Requirement Specification

4.1 Influence of C Programming Language

4.2 Influence of Compiler

4.3 Influence static analysis tool

5.0 Security Requirement specification

5.1 Safety features.(data type used,functions used)

5.2 List the safety recommendation required for your software (CERT rules and recommendations)

5.3 Justify the recommendation.

## 6.0 Design

### 6.1 Data Flow Diagram

### 6.2 Modularity

### 6.3 Readability

### 6.4 Abstraction

## 7.0 Implementation (Entire Code)

## 8.0 Testing and Verification

### 8.1 Non-Compliant Code Output

### 8.2 Compliant Code Output

## 9.0 Maintenance

### 9.1 Modification

### 9.2 Creating Multiple Versions

### 9.3 Scalability

## 10.0 Conclusion

## 11.0 References



# 2.0 Introduction

---

## *Problem Definition and Need*

### **2.1 Problem Definition:**

To create a realistic airport simulation for a single runway with the use of safety recommendations provided by CERT C.

We have used various data structures such as queues and structures, along with dynamic memory allocations.

### **2.2 Need of safe programming:**

Software vulnerabilities are unfortunately an ever-present risk, which is why secure coding is essential. For that reason, it's important that you ensure that your code is secure and protected. Here, we explain what is secure coding and provide best practices for secure coding. An insecure application lets hackers in. They can take direct control of a device — or provide an access path to another device.

This can result in:

- Denial of service to a single user
- Compromised secrets.
- Loss of service.
- Damage to the systems of thousands of users.
- Loss of life.

Secure code protects your software. That's why it's important to incorporate secure coding practices throughout the planning and development of your product. Using secure coding standards — such as CERT C and CWE — is key.

The preferences and recommendations prescribed by the CERT has helped us in writing a safe and secure code. Starting from variable description to the complicated memory allocation, the CERT rules have played a major role in minimizing the syntax and run-time errors and also provided a solution to encounter them. Our application needs a safe program and CERT recommendations are of high technical importance to us.

## 3.0 Requirement Specification

---

### *3.1 Establishment*

The process of simulation would run for many units of time, hence run a loop in main( ) that would run from <curtime> to <endtime> where <curtime> would be 1 and <endtime> would be the maximum number of units the program has to run. Generate a random number. Depending on the value of the random number generated, perform the following tasks.

1. If the random number is less than or equal to 1 then get data for the plane ready to land. Check whether or not the queue for landing of planes is full. If the queue is full then refuse the plane to land. If the queue is not empty then add the data to the queue maintained for planes landing.
2. If the random number generated is zero, then generate a random number again. Check if this number is less than or equal to 1. If it is , then get data for the plane ready to take off. Check whether or not the queue for taking a plane off is full. If the queue is

full then refuse the plane to take off otherwise add the data to the queue maintained for planes taking off.

3. It is better to keep a plane waiting on the ground than in the air, hence allow a plane to take off only, if there are no planes waiting to land.
4. After receiving a request from a new plane to land or take off, check the queue of planes waiting to land, and only if the landing queue is empty, allow a plane to take off.
5. If the queue for planes landing is not empty then remove the data of the plane in the queue else run the procedure to land the plane.
6. Similarly, if the queue for planes taking off is not empty then remove the data of the plane in the queue else run the procedure to take off the plane.
7. If both the queues are empty then the runway would be idle.
8. Finally, display the statistical data.

### *3.2 Risk Assessment*

Any recommendation of high priority risk is developed with extra care. Below is a risk assessment analysis for some of our core functions which relate to deep security and privacy systems in our software:

**Function name:** randomnumber()

**Description:** CWE-327: Use of a Broken or Risky Cryptographic Algorithm

**Recommendation:** MSC30-C. Do not use the rand() function for generating pseudorandom numbers

<b>Severity</b>	Medium
-----------------	--------

<b>Likelihood</b>	Unlikely
-------------------	----------

<b>Remediation Cost</b>	Low
-------------------------	-----

<b>Priority</b>	P6
-----------------	----

<b>Level</b>	L2
--------------	----

**Function name:** `clean_stdin()`, `clrscr()`

**Description:** CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

**Recommendation:** MSC24-C. Do not use deprecated or obsolescent functions. `gets()` is being referred to here.

<b>Severity</b>	High
<b>Likelihood</b>	Probable
<b>Remediation Cost</b>	Medium
<b>Priority</b>	P12
<b>Level</b>	L1

**Function name:** clean\_stdin(), clrscr()

**Description:** CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

**Recommendation:** ENV03-C.Sanitize the environment when invoking external programs.

**Severity** High

**Likelihood** Likely

**Remediation Cost** High

**Priority** P9

**Level** L1

These were a few important risk assessment analyses we had to make as they form a core part of our deep security analysis.

## 4.0 Software Requirement

---

### *4.1 Influence of C programming language*

One of the main reasons for the versatility of the C programming languages is the ability to directly access memory manipulating functions (by means of pointers to memory locations). However, this ability is what uncovers vulnerability loopholes in the languages, and exposes C programs to buffer overflow and format string attacks.

Most vulnerabilities in C programs arise from poor or careless access of memory. The memory problems normally arise because these languages do not have any mechanism for carrying out checks whether accessed memory has been allocated or initialized.

Thus it's always a beneficial practice to use auditing tools to check for potentially dangerous functions in the C source codes, so that we can easily identify security flaws and correct them before you see a red line.

Furthermore, one can use specialized compilers that



can assist in checking for memory boundary violations, memory leaks, and perform further auditing. Besides being prone to hacking attacks, the C and the C++ are indispensable. Therefore, knowing how to design a secure, solid code in these languages is important to ensure programs function optimally as desired while providing integrity and privacy of data.

## *4.2 Influence of Compiler*

We used gcc compiler which stands for GNU Compiler Collection, on the terminal provided by Microsoft Visual Studio Code. VS code has options to add various extensions, so we used third party extensions such as error lens to automatically detect syntactical errors. The command `gcc` also means GNU Compiler Collection, and it essentially serves as a front-end for several compilers and the linker.

Optimization can occur during any phase of compilation; however, the bulk of optimizations are performed after the syntax and semantic analysis of the front end and before the code generation of the back end; thus a common, even though somewhat contradictory, name for this part of the compiler is the

"middle end." The exact set of GCC optimizations includes the standard algorithms, such as loop optimization, jump threading, common subexpression elimination, instruction scheduling, and so forth.

Therefore by using gcc on VS Code we found optimal ways to make the safest possible code powered by CERT recommendations and rules.

### *4.3 Influence of Memory*

Memory issues arise because C language does not have any mechanism for carrying out checks whether accessed memory has been allocated or initialized. To put plainly, no implicit memory bounds checking is carried out. Allocation and freeing of memory is an activity that is likely to result in memory leaks or even crash a program.

For example, Buffer overflow and other related forms of attacks normally take place when a user enters more data than the program was designed to hold, consequently leading to arbitrary memory modifications. Based on how the stack is arranged, an

intruder can inject arbitrary code into memory, and hence overwrite the target buffer.

Thus we have used queues for our planes' order of landing and taking off and used pointers for dynamic memory allocation. Having allocated and copied structures containing flexible array members dynamically (MEM33-C). We have defined and used valid pointer functions (MEM10-C). We have avoided large stack allocations (MEM05-C).

# 5.0 Security Requirement

---

## 5.1 Safety Features

We have integrated three main safety features in our project. Following are the screenshots from flawfinder.

```
siddhi: ~
This causes a new program to execute and is difficult to use safely
(CWE-78). try using a library call that implements the same functionality
if available.
/home/siddhi/Downloads/server.c:38: [4] (shell) system:
This causes a new program to execute and is difficult to use safely
(CWE-78). try using a library call that implements the same functionality
if available.
/home/siddhi/Downloads/server.c:42: [4] (shell) system:
This causes a new program to execute and is difficult to use safely
(CWE-78). try using a library call that implements the same functionality
if available.
/home/siddhi/Downloads/Alrport.C:349: [3] (random) srand:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:276: [3] (random) random:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:280: [3] (random) random:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:283: [3] (random) random:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:284: [3] (random) random:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:384: [3] (random) srand:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:22: [1] (buffer) getchar:
Check buffer boundaries if used in a loop including recursive loops
(CWE-120, CWE-20).
/home/siddhi/Downloads/server.c:33: [1] (buffer) getchar:
Check buffer boundaries if used in a loop including recursive loops
(CWE-120, CWE-20).
/home/siddhi/Downloads/server.c:37: [1] (buffer) getchar:
Check buffer boundaries if used in a loop including recursive loops
(CWE-120, CWE-20).
/home/siddhi/Downloads/server.c:41: [1] (buffer) getchar:
Check buffer boundaries if used in a loop including recursive loops
(CWE-120, CWE-20).

ANALYSIS SUMMARY:

Hits = 14
Lines analyzed = 950 in approximately 0.15 seconds (6271 lines/second)
Physical Source Lines of Code (SLOC) = 735
Hits@level = [0] 0 [1] 4 [2] 0 [3] 6 [4] 4 [5] 0
Hits@level+ = [0+] 14 [1+] 14 [2+] 10 [3+] 10 [4+] 4 [5+] 0
Hits/KSLOC@level+ = [0+] 19.0476 [1+] 19.0476 [2+] 13.6054 [3+] 13.6054 [4+] 5.44218 [5+] 0
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See 'Secure Programming for Linux and Unix HOWTO'
(http://www.dwheeler.com/secure-programs) for more information.
siddhi@siddhi:~$ flawfinder /home/siddhi/Downloads --html
```

```
siddhi@siddhi: ~
Flawfinder version 1.31, (C) 2001-2014 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 169
Examining /home/siddhi/Downloads/server.c
Examining /home/siddhi/Downloads/Airport.C
Examining /home/siddhi/Downloads/server.h
Examining /home/siddhi/Downloads/client.c
Warning: Skipping non-existent file --html

FINAL RESULTS:

/home/siddhi/Downloads/Airport.C:359: [4] (shell) system:
This causes a new program to execute and is difficult to use safely
(CWE-78). try using a library call that implements the same functionality
if available.
/home/siddhi/Downloads/server.c:34: [4] (shell) system:
This causes a new program to execute and is difficult to use safely
(CWE-78). try using a library call that implements the same functionality
if available.
/home/siddhi/Downloads/server.c:38: [4] (shell) system:
This causes a new program to execute and is difficult to use safely
(CWE-78). try using a library call that implements the same functionality
if available.
/home/siddhi/Downloads/server.c:42: [4] (shell) system:
This causes a new program to execute and is difficult to use safely
(CWE-78). try using a library call that implements the same functionality
if available.
/home/siddhi/Downloads/Airport.C:349: [3] (random) srand:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:276: [3] (random) random:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:280: [3] (random) random:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:283: [3] (random) random:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:284: [3] (random) random:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:384: [3] (random) srand:
This function is not sufficiently random for security-related functions
such as key and nonce creation (CWE-327). use a more secure technique for
acquiring random values.
/home/siddhi/Downloads/server.c:22: [1] (buffer) getchar:
Check buffer boundaries if used in a loop including recursive loops
(CWE-120, CWE-20).
/home/siddhi/Downloads/server.c:33: [1] (buffer) getchar:
Check buffer boundaries if used in a loop including recursive loops
(CWE-120, CWE-20).
/home/siddhi/Downloads/server.c:37: [1] (buffer) getchar:
Check buffer boundaries if used in a loop including recursive loops
(CWE-120, CWE-20).
/home/siddhi/Downloads/server.c:41: [1] (buffer) getchar:
Check buffer boundaries if used in a loop including recursive loops
(CWE-120, CWE-20).
```

We observe that we have three main vulnerabilities CWE-78 (Shell), CWE-327 (random), CWE-120/20 (Buffer overflow). We dealt with the vulnerability caused by the inbuilt gets() function, the rand() function of random.c library and the classic buffer overflow as discussed in detail below.



## dealing with gets()

The `gets()` function reads a line from standard input into a buffer until a terminating newline or end-of-file (EOF) is found. No check for buffer overflow is performed.

It is impossible to tell without knowing the data in advance how many characters `gets()` will read, and because `gets()` will continue to store characters past the end of the buffer, it is extremely dangerous to use. In the past it has been used to break computer security.

Also the `gets()` function has been deprecated in the International Organization for Standardization /IEC 9899:TC3 and removed from C11.

A strictly C99-conforming application is to use the `getchar()` function. The `getchar()` function returns the next character from the input stream. If the stream is at EOF, the EOF indicator for the stream is set and `getchar()` returns EOF.

## dealing with classic buffer overflow

A portable way to clear up to the end of a line that you've already tried to read partially is:

```
int c;  
while ((c = getchar()) != '\n' && c != EOF){}
```

This reads and discards characters until it gets `\n` which signals the end of the file. It also checks against EOF in case the input stream gets closed before the end of the line. The type of char must be int (or larger) in order to be able to hold the value EOF.

There is no portable way to find out if there are any more lines after the current line (if there aren't, then `getchar` will block for input).

`clean_stdin()` along with `system clr)` is invoked in the `clrscr()` to tackle the buffer overflow

Also in `clrscr()` `#ifdef` directive allows for conditional compilation. The preprocessor determines if the provided macro exists before including the subsequent

code in the compilation process. So basically it decides early on if our system is windows, linux or macOS.

## dealing with rand()

rand() function makes no guarantees as to the quality of the random sequence produced. The numbers generated by some implementations of rand() have a comparatively short cycle and the numbers can be predictable.

POSIX random() function is a better pseudorandom number generator. Although on some platforms the low dozen bits generated by rand() go through a cyclic pattern, all the bits generated by random() are usable.

Applications that have strong pseudorandom number requirements must use a generator that is known to be sufficient for their needs.

Our second method was to introduce a user generated seeding using srand().



## *5.2 List the safety recommendation required for your software*

### 1) Rule 01. Pre-processor

1.1 RECOMMENDATION: PRE30-C. Do not create a universal character name through concatenation

1.2 RECOMMENDATION: PRE32-C. Do not use pre-processor directives in invocations of function-like macros(The arguments to a macro must not include pre-processor directives, such as #define, #ifdef, and #include. Doing so results in undefined behaviour)

### 2) Rule 02. Declarations and Initialization (DCL)

2.1 DCL31-C. Declare identifiers before using them.

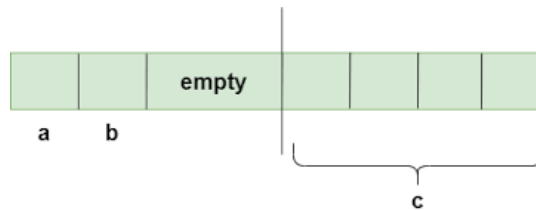
2.2 DCL36-C. Do not declare an identifier with conflicting linkage classifications(static, extern)

2.3 DCL37-C. Do not declare or define a reserved identifier

2.4 DCL40-C. Do not create incompatible declarations of the same function or object

### 3) Rule 03. Expressions (EXP)

3.1 RECOMMENDATION: EXP42-C Do not compare padding data. (usually present in structures, because these padding values are unspecified, attempting a byte-by-byte comparison between structures can lead to incorrect results.)



3.2 EXP30-C. Do not depend on the order of evaluation for side effects

3.3 EXP32-C. Do not access a volatile object through a nonvolatile reference

3.4 EXP35-C. Do not modify objects with temporary lifetime

3.5 EXP37-C. Call functions with the correct number and type of arguments

3.6 EXP39-C. Do not access a variable through a pointer of an incompatible type

3.7 EXP45-C. Do not perform assignments in selection statements( like in a for/while loop or if statement; avoid assignments like  $b=a$  )

3.10 EXP46-C. Do not use a bitwise operator with a Boolean-like operand

## 4) Rule 04. Integers (INT)

4.1 INT30-C. Ensure that unsigned integer operations do not wrap (we've used unsigned for seeding in random number generation)

Unsigned integer operations can wrap if the resulting value cannot be represented by the underlying representation of the integer.

4.2 INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors

4.3 INT34-C. Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand

4.4 INT35-C. Use correct integer precisions

4.5 INT36-C. Converting a pointer to integer or integer to pointer

Conversions between integers and pointers can have undesired consequences depending on the implementation

## 5) Rule 05. Floating Point (FLP)

5.1 FLP30-C. Do not use floating-point variables as loop counters (may loop for undeterminable number of times)

## 6) Rule 08. Memory Management (MEM)

6.1 MEM34-C. Only free memory allocated dynamically

## 7) Rule 09. Input Output (FIO)

7.1 FIO41-C. Do not call `getc()`, `putc()`, `getwc()`, or `putwc()` with a stream argument that has side effects

## 8) Rule 10. Environment

8.1 ENV30-C. Do not modify the object referenced by the return value of certain functions

8.2 ENV34-C. Do not store pointers returned by certain functions

## 9) Rule 48. Miscellaneous (MSC)

9.1 MSC30-C. Do not use the `rand()` function for generating pseudorandom numbers

9.2 MSC32-C. Properly seed pseudorandom number generators.

### *5.3 Justify the recommendations*

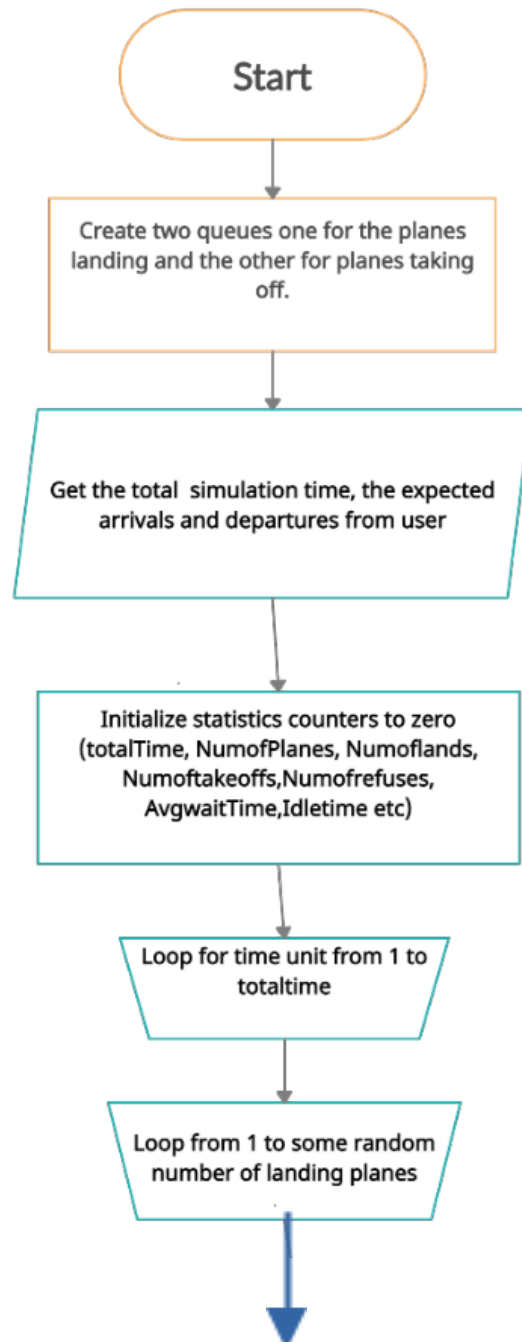
We wrote a non compliant code following just the logic required in our program, then upon successful completion we ran it through splint and flawfinder. We got CWE (Common Weakness Enumeration) warnings per se.

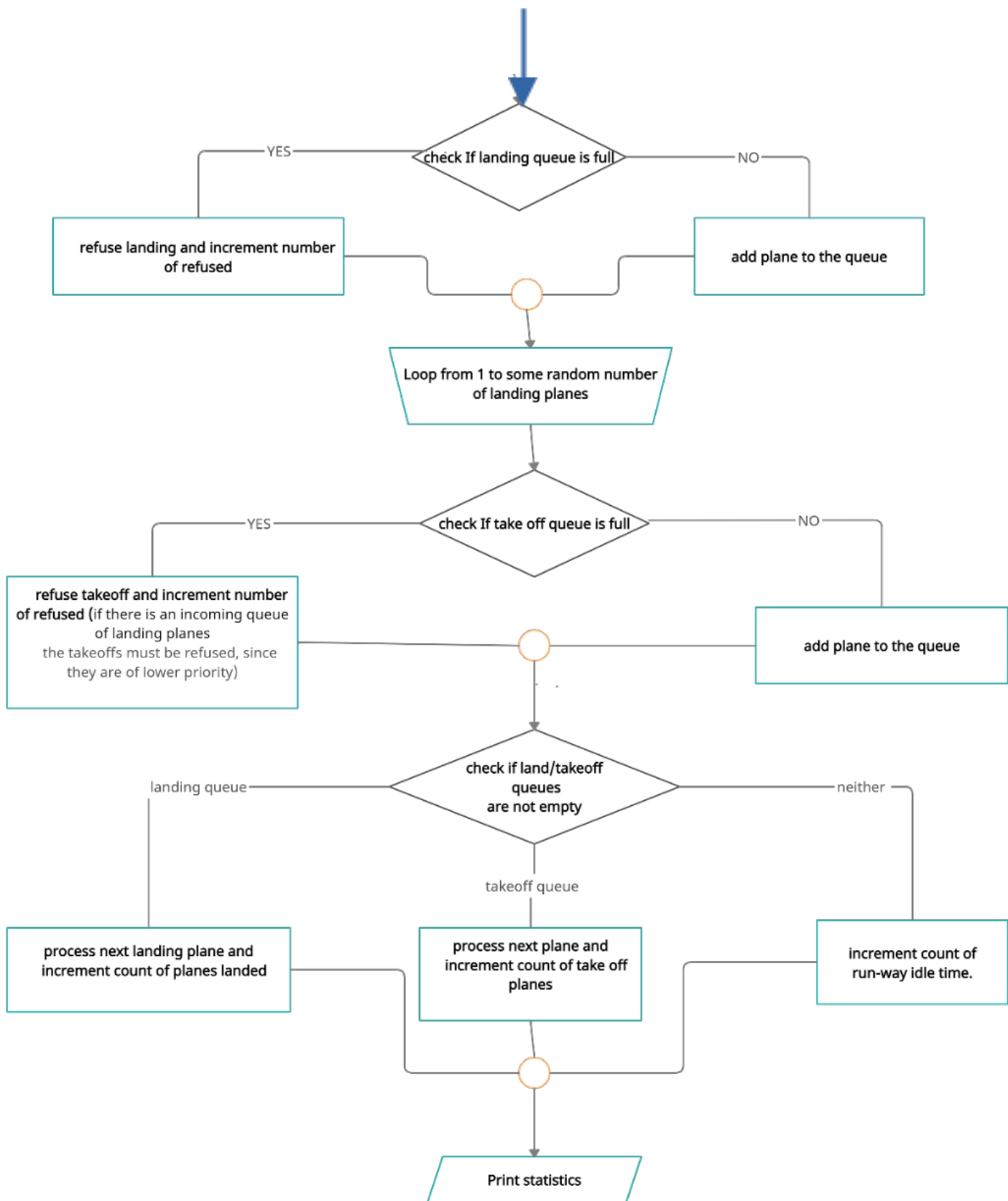
Looking up the above mentioned on mitre.org and confluence, we corrected these vulnerabilities. Simultaneously we noted down the recommendations and rules we can follow as required in our code and made the necessary changes.

# 6.0 Design

---

## 6.1 Data Flow Diagram





## *6.2 Modularity*

Modularity is the degree to which a system's components may be separated and recombined, often with the benefit of flexibility and variety in use. The concept of modularity is used primarily to reduce complexity by breaking a system into varying degrees of interdependence and independence across and "hide the complexity of each part behind an abstraction and interface".

For this reason we have used the makefile method. We defined the headers in `server.h`. We have extern declaration of global variables. This file contains all the data types, primary and reference (user-defined) such as the structures and implemented the queues we have used in our code. We have defined the structure plane which is basically the builder. In `server.h` we have called the functions which are defined in the `server.c` file.

The source code where all the functions are essentially defined in the `server.c`. It uses the data types declared in `server.h` for computation in the functions defined here.

The functions and data types are all finally called and used in `client.c`. Here the complete calculation of the



airport statistics takes place, using the data provided by the server.h and server.c.

### 6.3 Readability

It is often said that readability is perhaps the most important quality-defining measure of a given piece of code for reasons concerning maintainability, ease of understanding and use.

1. Standards of indentation and formatting are followed, so that the code and its structure are clearly visible.
2. Variables are named meaningfully, so that they communicate intent. For example- idle time indicates the time for which the airport runway remains idle
3. Comments, which are present only where needed, are concise and adhere to standard formats.
4. Facilities of the language are used skillfully, leveraging iteration and recursion rather than copy and paste coding.
5. Functions are short and to the point, *and do one thing*.
6. Limited Line Length
7. Indirection is minimized as much as possible, while still maintaining flexibility.

## 6.4 Abstraction

Abstraction is a representation of a computation entity. It is a way to conceal its particular information and only give the most relevant information to the programmer.

We've used data structures like queues and structures , wherein these are given a user-defined name with similar guidelines used in the functions' naming thereby increasing the abstraction levels. An example of which is:

```
void initializeQueue(queue* q){  
    q->count = 0;  
    q->front = 0;  
    q->back = -1;  
}
```

Here and in multiple instances we've also followed the Cert recommendation DCL20-C, explicitly specifying void when a function accepts no arguments.

In all of our functions and we've avoided the use of a primitive data type, a user defined data type is built for every occurrence of a data type.

# 7.0 Implementation

---

## *server.h (header file)*

//defining global constants

#define MAX\_POSSIBLE 3

#define ARRIVE 0

#define DEPART 1

//Defining the structure plane

typedef struct Plane {

int planeID;

int fuelTime ;

}plane;

//Implementation of queue

typedef struct Queue {

int count;

int front;

int back;

plane arrayOfPlanes[MAX\_POSSIBLE];

} queue;

//Defining the structure plane, which is basically the builder

typedef struct airPort {

queue landingInit;

queue takeoffInit;

queue \*landingQueue, \*takeoffQueue;

```
int waitForLanding, waitForTakeoff;
int idletime, NumOfLand, NumOfPlanes, NumOfRefuses, NumOfTakeOffs;
plane p;
}airport;
```

```
//function declarations
```

```
void delay(float);
void initializeQueue(queue*);
void addToQueue(queue*, plane);
plane popQueue(queue*);
int sizeOfQueue(queue);
int isEmptyQueue(queue);
int isFullQueue(queue);
void initializeAirport(airport*);
void start(int*);
void createNewPlane(airport*, int, int);
void refuseLand(airport*);
void land(airport*, plane, int);
void fly(airport*, plane, int);
void idle(airport*, int);
void Result(airport*, int);
int randomnumber(void);
void airportAddToQueue(airport*, int);
plane airportPopQueue(airport*, int);
int sizeOfAirport(airport, int);
int isFullAirport(airport, int);
int isEmptyAirport(airport, int);
void seed(void);
```

```
void clrscr(void);  
void clean_stdin(void);
```

## *client.c*

```
//All the includes  
#include<stdio.h>  
#include<stdlib.h>  
#include<ctype.h>  
#include<time.h>  
  
#include"server.h"  
  
int main(void){  
    //The main function  
    //Creating instances for all the structures  
    //And defining the variables  
    airport AirPort;  
    int i, numOfPlanes, presentTime, TotalTime ;  
  
    plane temp;  
    //Initializing the airport  
    initializeAirport(&AirPort);  
    //Starting  
    start(&TotalTime);  
    //Outer for loop manages the number of iterations, which is a count of  
time  
    //Effectively, 1 iteration is 1 unit of time
```

```

for(presentTime = 1 ; presentTime <= TotalTime ; presentTime++){
    numOfPlanes = randomnumber();
    //The num of planes for every iteration is
    //randomly generated by using a slightly complex random function
    for(i = 0 ; i < numOfPlanes ; i++){
        //Generation of planes randomly
        createNewPlane(&AirPort, presentTime, DEPART);
        //For every iteration a new plane will be generated
        //and added to the landing queue if and only if
        //the airport is empty and no plane needs to land
        if(isFullAirport(AirPort, DEPART))
            refuseLand(&AirPort);
            //The plane will be refused to be allowed to
            //take off if the airport is not empty
        else
            airportAddToQueue(&AirPort, DEPART);
            //Will be allowed to take off if the airport is empty
    }
}

```

```

numOfPlanes = randomnumber();

```

```

for(i = 0 ; i < numOfPlanes ; i++){
    //For every iteration a new plane will be generated
    //and added to the landing queue
    createNewPlane(&AirPort, presentTime, ARRIVE);
    airportAddToQueue(&AirPort, ARRIVE);
}

```

```

//Here, the generation of planes is complete
//Its time to either land or take off
if(!(isEmptyAirport(AirPort, ARRIVE))){
    //If landing is possible
    while(!isEmptyAirport(AirPort, ARRIVE)){
        //All the planes that need to land will land
        temp = airportPopQueue(&AirPort, ARRIVE);
        land(&AirPort, temp, presentTime);
    }
} else if(!(isEmptyAirport(AirPort, DEPART))) {
    //Take off will not be allowed for all at the same time
    temp = airportPopQueue(&AirPort, DEPART);
    fly(&AirPort, temp, presentTime);
} else {
    //If there is no plane to land or takeoff,
    //the runway is idle
    idle(&AirPort, presentTime);
}
delay(0.25);
//mimicking the sleep function

}

//After each iteration, a unit of time passes

//Calling the result function
Result(&AirPort, TotalTime);
//clearing the screen
clrscr();

```

```
    return 0;  
}
```

*servern.c (non-compliant code)*

```
#include<stdio.h>  
#include<stdlib.h>  
#include<ctype.h>  
#include<math.h>  
#include<time.h>  
#include<limits.h>
```

```
#include"server.h"
```

```
void clrscr(){  
  
    printf("\nPress Enter key to exit\n");  
    getchar();  
    system("cls");  
}
```

```
void delay(float seconds)  
{  
    int milli=1000*seconds;  
    clock_t start=clock();  
    while(clock()<start+milli);  
}
```



```

}
void initializeQueue(queue* q){
    q->count = 0;
    q->front = 0;
    q->back = -1;
}
void addToQueue(queue* q, plane next){
    if(q->count >= MAX_POSSIBLE){
        printf("\nQueue is isFull\n");
        return ;
    }
    (q->count)++;
    q->back++;
    q->back %= MAX_POSSIBLE;
    q->arrayOfPlanes[q->back] = next ;
}

plane popQueue(queue *q) {
    plane Plane ;

    if(q->count <= 0) {
        printf("\nQueue is isEmptyQueue.\n");
        Plane.planeID = 0 ;
        Plane.fuelTime = 0 ;
    }else {
        (q->count)--;
        Plane=q->arrayOfPlanes[q->front] ;
        q->front=(q->front+ 1) % MAX_POSSIBLE ;
    }
}

```

```
    }  
    return Plane ;  
}
```

```
int sizeOfQueue(queue q) {  
    return q.count ;  
}
```

```
int isEmptyQueue(queue q){  
    return(sizeOfQueue(q) < 1);  
}
```

```
int isFullQueue(queue q){  
    return(sizeOfQueue(q) >= MAX_POSSIBLE);  
}
```

```
void initializeAirport(airport *airPort){  
    initializeQueue(&(airPort->landingInit));  
    initializeQueue(&(airPort->takeoffInit));  
    airPort->landingQueue = &(airPort->landingInit);  
    airPort->takeoffQueue = &(airPort->takeoffInit);  
    airPort->NumOfPlanes = airPort->NumOfLand = airPort->NumOfTakeOffs  
=0;  
    airPort->NumOfRefuses = airPort->idletime = 0 ;  
    airPort->waitForLanding = airPort->waitForTakeoff = 0 ;  
}
```

```
void start(int *totalTime){  
    int flag = 1;
```

```

double temp;

printf("\nProgram that simulates an airport with only one runway.\n");
printf("One plane can land or depart in each unit of time.\n");
printf("Up to %d planes can be waiting to land or take off at any time.\n",
MAX_POSSIBLE);
printf("How many units of time will the simulation run?");
scanf("%lf", &temp);
*totalTime = temp/1;
printf("\n%d\n", *totalTime);
printf("\n\n\n");
return;
}

```

```

void createNewPlane(airport *airPort, int curtime, int type) {
    (airPort->NumOfPlanes)++;
    airPort->p.planeID = airPort->NumOfPlanes ;
    airPort->p.fuelTime = curtime ;
    switch(type) {
        case ARRIVE :
            printf("\nPlane %d ready to land.\n", airPort->NumOfPlanes);
            return;

        case DEPART :
            printf("\nPlane %d ready to take off.\n", airPort->NumOfPlanes);

```

```

        return ;

    default:
        printf("Error");
        exit(0);
        break;
    }
}

void refuseLand(airport *airPort) {
    printf("\nPlane %d told to try later.\n", airPort->p.planeID);
    (airPort->NumOfRefuses)++;
}

void land(airport *airPort, plane Plane, int currentTime) {
    int wait;
    wait = currentTime - Plane.fuelTime;
    printf("\n%d: Plane %d landed ", currentTime, Plane.planeID);
    printf("in queue %d units \n", wait);
    (airPort->NumOfLand)++ ;
    (airPort->waitForLanding) += wait ;
}

void fly(airport *airPort, plane Plane, int curtime) {
    int wait ;
    wait = curtime - Plane.fuelTime ;
    printf("%d: Plane %d took off ", curtime, Plane.planeID);
    printf("in queue %d units \n", wait);
    (airPort->NumOfTakeOffs)++ ;
}

```

```

    (airPort->waitForTakeoff) += wait ;
}

```

```

void idle(airport *airPort, int curtime) {
    printf("%d: Runway is idle.\n", curtime);
    airPort->idletime++;
}

```

```

void Result(airport *airPort, int totalTime) {
    printf("\nResult>>>\n");
    printf("\tSimulation has concluded after %d units.\n", totalTime);
    printf("\tTotal number of planes processed: %d\n",
airPort->NumOfPlanes);
    printf("\tNumber of planes landed: %d\n", airPort->NumOfLand);
    printf("\tNumber of planes taken off: %d\n", airPort->NumOfTakeOffs);
    printf("\tNumber of planes refused use: %d\n", airPort->NumOfRefuses);
    printf("\tNumber left ready to land: %d\n", sizeOfAirport(*airPort,
ARRIVE));
    printf("\tNumber left ready to take off: %d\n", sizeOfAirport(*airPort,
DEPART));
    if(totalTime>0){
        double percentIdleTime =(((double)airPort->idletime/totalTime)*100.0;
        printf("\tPercentage of time runway idle: %.3lf \n", percentIdleTime);
    }
    if(airPort->NumOfLand>0){
        double avgWaitTimeForLand
=(((double)airPort->waitForLanding/airPort->NumOfLand);
        printf("\tAverage wait time to land: %.3lf \n", avgWaitTimeForLand);
    }
}

```

```

    if(airPort->NumOfTakeOffs>0){
        double avgWaitTimeForTakeOff
        =((double)airPort->waitForTakeoff/airPort->NumOfTakeOffs);
        printf("\tAverage wait time to take off: %.3lf \n",avgWaitTimeForTakeOff);
    }

}

```

```

int randomnumber(void) {
    int random;
    random=rand()%58;
    return random;
}

```

```

void airportAddToQueue(airport *airPort, int type) {
    switch(type) {
        case ARRIVE :
            addToQueue(airPort->landingQueue, airPort->p);
            break ;

        case DEPART :
            addToQueue(airPort->takeoffQueue, airPort->p);
            break ;

        default:
            printf("Error");
            exit(0);

    }
}

```

```
}
```

```
plane airportPopQueue(airport *airPort, int type) {  
    plane p1 ;  
    switch(type) {  
        case ARRIVE :  
            p1 = popQueue(airPort->landingQueue);  
            return p1 ;  
  
        case DEPART :  
            p1 = popQueue(airPort->takeoffQueue);  
            return p1 ;  
  
        default:  
            printf("Error");  
            exit(0);  
    }  
}
```

```
int sizeOfAirport(airport airPort, int type) {  
    switch(type) {  
        case ARRIVE :  
            return(sizeOfQueue(*(airPort.landingQueue)));  
  
        case DEPART :  
            return(sizeOfQueue(*(airPort.takeoffQueue)));  
  
        default:  
            printf("Error");
```

```
        exit(0);
    }
}
```

```
int isFullAirport(airport airPort, int type) {
    switch(type) {
        case ARRIVE :
            return(isFullQueue(*(airPort.landingQueue)));

        case DEPART :
            return(isFullQueue(*(airPort.takeoffQueue)));

        default:
            printf("Error");
            exit(0);
    }
}
```

```
int isEmptyAirport(airport airPort, int type) {
    switch(type) {
        case ARRIVE :
            return(isEmptyQueue(*(airPort.landingQueue)));

        case DEPART :
            return(isEmptyQueue(*(airPort.takeoffQueue)));

        default:
            printf("Error");
    }
}
```



```
        exit(0);  
    }  
}
```

## *server.c (compliant code)*

//All the includes

```
#include<stdio.h>  
#include<stdlib.h>  
#include<ctype.h>  
#include<math.h>  
#include<time.h>  
#include<limits.h>
```

```
#include"server.h"
```

```
void clean_stdin() {
```

//Since the default buffer clear function

//is system dependent,this function clears the buffer in all cases

```
int c;  
while ((c = getchar()) != '\n' && c != EOF){}  
}
```

```
void clrscr(){
```

```
//This func clears the buffer
clean_stdin();
//and also clears the screen when you press any key
//It is system dependent
```

```
printf("\nPress Enter key to exit\n");
#ifdef _WIN64
    getchar();
    system("cls");
#endif
#ifdef _APPLE_
    getchar();
    system("clear");
#endif
#ifdef _linux_
    getchar();
    system("clear");
#endif
}
```

```
void delay(float seconds)
{
    //This function mimics the sleep function in python
    //by taking control for an amount of time
    int milli=1000*seconds;
    clock_t start=clock();
    while(clock())<start+milli;
}
```

```

void initializeQueue(queue* q){
    //This function initializes the queue values
    q->count = 0;
    q->front = 0;
    q->back = -1;
}

void addToQueue(queue* q, plane next){
    //This function adds planes to the queue
    if(q->count >= MAX_POSSIBLE){
        // if the queue is full, no more planes can be handled
        printf("\nQueue is isFull\n");
        return ;
    }
    //else it adds one more plane to the queue
    // and increases the count by one
    (q->count)++;
    q->back++;
    q->back %= MAX_POSSIBLE;
    q->arrayOfPlanes[q->back] = next ;
}

```

```

plane popQueue(queue *q) {
    //This function deletes the elements of the queue
    plane Plane ;

    if(q->count <= 0) {
        //If the queue is already empty, nothing is popped
        printf("\nQueue is isEmptyQueue.\n");
        Plane.planeID = 0 ;
    }
}

```

```

    Plane.fuelTime = 0 ;
}else {
    //else the first element entered is popped
    //and the second element is made to be the first element
    (q->count)--;
    //the loss of one element reduces the count by one
    Plane=q->arrayOfPlanes[q->front] ;
    q->front=(q->front+ 1) % MAX_POSSIBLE ;
    //this part does the popping
}
return Plane ;
//returns the plane object
}

```

```

int sizeOfQueue(queue q) {
    //this function returns the size of the queue
    //count is an element of the above defined queue object
    return q.count ;
}

```

```

int isEmptyQueue(queue q){
    //this returns a boolean value which tells whether the sizeOfQueue
    //is non-zero or not
    return(sizeOfQueue(q) < 1);
}

```

```

int isFullQueue(queue q){
    //this function returns a boolean value which tells
    //whether the queue is full or not

```

```
    return(sizeofQueue(q) >= MAX_POSSIBLE);  
}
```

```
void initializeAirport(airport *airPort){  
    initializeQueue(&(airPort->landingInit));  
    initializeQueue(&(airPort->takeoffInit));  
    //The above lines call the function which initialises the queues  
    //The initialized queues are queue objects, which are replaced  
    //by queue pointers for easy handling  
    //landingQueue and takeoffQueue are pointers  
    airPort->landingQueue = &(airPort->landingInit);  
    airPort->takeoffQueue = &(airPort->takeoffInit);  
    //the below values are initialised to zero  
    airPort->NumOfPlanes = airPort->NumOfLand = airPort->NumOfTakeOffs  
=0;  
    airPort->NumOfRefuses = airPort->idletime = 0 ;  
    airPort->waitForLanding = airPort->waitForTakeoff = 0 ;  
}
```

```
void start(int *totalTime){  
    //the function responsible for starting the whole process  
    int flag = 1;  
    double temp;  
  
    printf("\nProgram that simulates an airport with only one runway.\n");  
    printf("One plane can land or depart in each unit of time.\n");  
    printf("Up to %d planes can be waiting to land or take off at any time.\n",  
MAX_POSSIBLE);  
    //the random generator in c will generate the same pattern each time
```

unless we explicitly seed it

```
seed();
while(flag){
//making sure that the values we take are not meaningless
//we use a while loop and break out if everything is fine
printf("How many units of time will the simulation run?");
scanf("%lf", &temp);
*totalTime = temp/1;
//If the input by mistake is given as a float for total time
//the above lines handle that
//Since its logically meaningless to give negative values
//we keep on taking the values as long as there are wrong values
if(*totalTime < 0){
    printf("These numbers must not be negative.\nPlease re-enter the
values\n");
    flag = 1;
}
else
    flag=0;
}

printf("\n%d\n",*totalTime);
printf("\n\n\n");
return;
}
```

```
void createNewPlane(airport *airPort, int curtime, int type) {
//This function generates a new plane
```

```

(airPort->NumOfPlanes)++;
//the id will be 'n' for 'n'th plane
airPort->p.planeID = airPort->NumOfPlanes ;
airPort->p.fuelTime = curtime ;
//alloting the neccessary values
switch(type) {
    //printing the statements
    //since return is used, there is no need for break
    case ARRIVE :
        printf("\nPlane %d ready to land.\n", airPort->NumOfPlanes);
        return;

    case DEPART :
        printf("\nPlane %d ready to take off.\n", airPort->NumOfPlanes);
        return ;

    default:
        printf("Error");
        exit(0);
        break;
}
}

void refuseLand(airport *airPort) {
    //if there is an incoming queue of landing planes
    //the take offs must be refused, since they are of lower priority
    printf("\nPlane %d told to try later.\n", airPort->p.planeID);
    //since only takeoffs can be refused, there is no need for an identifier
    //for whether its a takeoff or a landing

```

```
(airPort->NumOfRefuses)++;  
}
```

```
void land(airport *airPort, plane Plane, int currentTime) {  
    //This function effectively lands the plane  
    //By changing the variables which must be changed after landing  
    int wait;  
    wait = currentTime - Plane.fuelTime;  
    printf("\n%d: Plane %d landed ", currentTime, Plane.planeID);  
    printf("in queue %d units \n", wait);  
    (airPort->NumOfLand)++;  
    //The num of planes landed will increase  
    //The total wait time will also be incremented  
    (airPort->waitForLanding) += wait;  
}
```

```
void fly(airport *airPort, plane Plane, int curtime) {  
    //This function effectively takes off the plane  
    //By changing the variables that must be changed after landing  
    int wait;  
    wait = curtime - Plane.fuelTime;  
    printf("%d: Plane %d took off ", curtime, Plane.planeID);  
    printf("in queue %d units \n", wait);  
    (airPort->NumOfTakeOffs)++;  
    //The num of planes took off will increase  
    //Also the total time waited will also be incremented  
    (airPort->waitForTakeoff) += wait;  
}
```



```

void idle(airport *airPort, int curtime) {
    //This function calculates the amount of time runway is left idle
    printf("%d: Runway is idle.\n", curtime);
    airPort->idletime++;
}

void Result(airport *airPort, int totalTime) {
    //This functions prints all the results
    printf("\nResult>>>\n");
    printf("\tSimulation has concluded after %d units.\n", totalTime);
    printf("\tTotal number of planes processed: %d\n",
airPort->NumOfPlanes);
    printf("\tNumber of planes landed: %d\n", airPort->NumOfLand);
    printf("\tNumber of planes taken off: %d\n", airPort->NumOfTakeOffs);
    printf("\tNumber of planes refused use: %d\n", airPort->NumOfRefuses);
    printf("\tNumber left ready to land: %d\n", sizeOfAirport(*airPort,
ARRIVE));
    printf("\tNumber left ready to take off: %d\n", sizeOfAirport(*airPort,
DEPART));
    //the below statements must be executed only if
    //certain values are defined

    //eg, if numoflands = 0, then there is no point in calculationg
avgWaitForLand
    if(totalTime>0){
        double percentIdleTime =((double)airPort->idletime/totalTime)*100.0;
        printf("\tPercentage of time runway idle: %.3lf \n", percentIdleTime);
    }
    if(airPort->NumOfLand>0){

```

```

    double avgWaitTimeForLand
=((double)airPort->waitForLanding/airPort->NumOfLand);
    printf("\tAverage wait time to land: %.3lf \n", avgWaitTimeForLand);
}
if(airPort->NumOfTakeOffs>0){
    double avgWaitTimeForTakeOff
=((double)airPort->waitForTakeoff/airPort->NumOfTakeOffs);
    printf("\tAverage wait time to take off: %.3lf \n",avgWaitTimeForTakeOff);
}

}

```

```

int randomnumber(void) {
    //This function generates random number
    //The seeding was done beforehand
    int randomi;
    srand(time(0));
    //Since random number generation in c has a flaw of repetition
    //We have taken quite a few steps to avoid that
    randomi =(random() + random())/2;
    randomi *=(random() %63);
    //to generate true random numbers we are combining three random
numbers
    //All the hardcoded values chosen here are totally random
    randomi %= 3;
    return randomi;
}

```

```

void airportAddToQueue(airport *airPort, int type) {

```

```
//This function adds a plane to the queue of the airport
//it is basically a caller function which decides
//whether the parameter must be takeoff or landing queue
```

```
switch(type) {
    case ARRIVE :
        addToQueue(airPort->landingQueue, airPort->p);
        break ;

    case DEPART :
        addToQueue(airPort->takeoffQueue, airPort->p);
        break ;

    default:
        printf("Error");
        exit(0);
}
}
```

```
plane airportPopQueue(airport *airPort, int type) {
    //This function deletes a plane from the queue of the airport
    //it is basically a caller function which decides
    //whether the parameter must be takeoff or landing queue
    plane p1 ;
    switch(type) {
        case ARRIVE :
            p1 = popQueue(airPort->landingQueue);
            return p1 ;
    }
```

```
case DEPART :
```

```
    p1 = popQueue(airPort->takeoffQueue);
```

```
    return p1 ;
```

```
default:
```

```
    printf("Error");
```

```
    exit(0);
```

```
}
```

```
}
```

```
int sizeOfAirport(airport airPort, int type) {
```

```
    //This function finds the size of the airport
```

```
    //it is basically a caller function which decides
```

```
    //whether the parameter must be takeoff or landing queue
```

```
    switch(type) {
```

```
        case ARRIVE :
```

```
            return(sizeofQueue(*(airPort.landingQueue)));
```

```
        case DEPART :
```

```
            return(sizeofQueue(*(airPort.takeoffQueue)));
```

```
    default:
```

```
        printf("Error");
```

```
        exit(0);
```

```
}
```

```
}
```

```
int isFullAirport(airport airPort, int type) {
```

```

//This function finds the size of the airport and
//determines whether it is isFullQueue
//it is basically a caller function which decides
//whether the parameter must be takeoff or landing queue
switch(type) {
    case ARRIVE :
        return(isFullQueue(*(airPort.landingQueue)));

    case DEPART :
        return(isFullQueue(*(airPort.takeoffQueue)));

    default:
        printf("Error");
        exit(0);
}
}

```

```

int isEmptyAirport(airport airPort, int type) {
    //This function finds the size of the airport and
    //determines whether it is empty
    //it is basically a caller function which decides
    //whether the parameter must be takeoff or landing queue
    switch(type) {
        case ARRIVE :
            return(isEmptyQueue(*(airPort.landingQueue)));

        case DEPART :
            return(isEmptyQueue(*(airPort.takeoffQueue)));
    }
}

```

```
    default:
        printf("Error");
        exit(0);
    }
}

void seed(void){
    //This function seeds the rand function
    //making it generate a different pattern
    srand((unsigned int)(time(NULL) % 10000));
}
```

# 8.0 Testing

---

## *8.1 Vulnerable Test Cases*

### Random number generation:

#### **Rules and Recommendations used-**

1. Rule 02. Declarations and Initialization (DCL)
  - 1.1 DCL37-C. Do not declare or define a reserved identifier
2. Rule 03. Expressions (EXP)
  - 2.1 EXP30-C. Do not depend on the order of evaluation for side effects
3. Rule 48. Miscellaneous (MSC)
  - 3.1 MSC30-C. Do not use the rand() function for generating pseudorandom numbers
  - 3.2 MSC32-C. Properly seed pseudorandom number generators

```

int randomnumber(void) {
    //This function generates random number
    //The seeding was done beforehand
    int randomi;
    srand(time(0));
    //Since random number generation in c has a flaw of repetition
    //We have taken quite a few steps to avoid that
    randomi =(random() + random())/2;
    randomi *=(random() %63);
    //to generate true random numbers we are combining three random numbers
    //All the hardcoded values chosen here are totally random
    randomi %-= 3;
    return randomi;
}

```

*(the above rules and recommendations seen to be applied in this code snippet)*

#### 4. Rule 04. Integers (INT)

INT30-C. Ensure that unsigned integer operations do not wrap (we've used unsigned for seeding in random number generation)

```

void seed(void){
    //This function seeds the rand function
    //making it generate a different pattern
    srand((unsigned int)(time(NULL) % 10000));
}

```

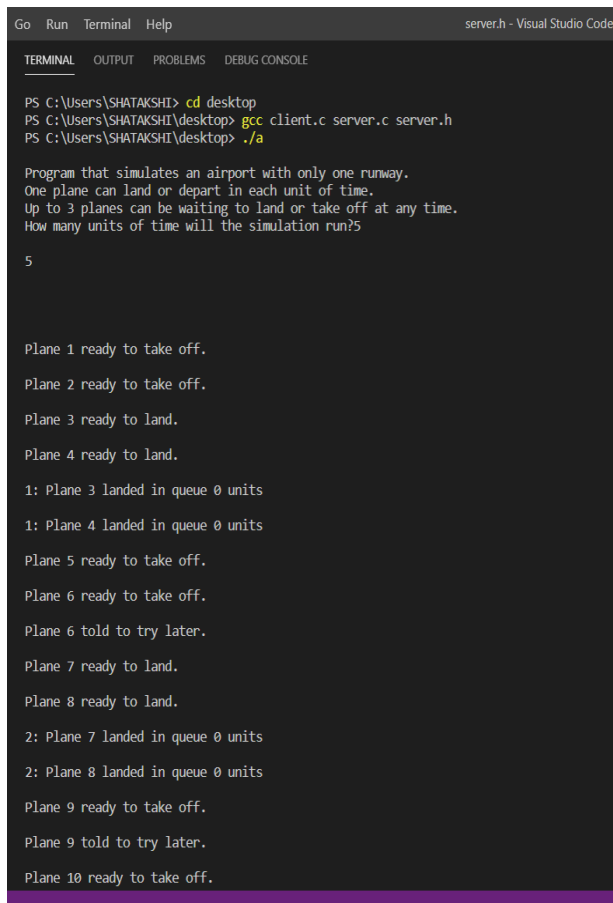
*(the above rules and recommendations seen to be applied in this code snippet)*



Without proper seeding and without using the POSIX random function in our application, we came across repetition in number generation. At different times we observed the same sequence of numbers being generated for the same inputs.

Seeding makes sure the compliant version of our code faces no issues regarding output repetition.

For 5 units of time, we observe different outputs at different time stamps:



```
Go Run Terminal Help server.h - Visual Studio Code

TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE

PS C:\Users\SHATAKSHI> cd desktop
PS C:\Users\SHATAKSHI\desktop> gcc client.c server.c server.h
PS C:\Users\SHATAKSHI\desktop> ./a

Program that simulates an airport with only one runway.
One plane can land or depart in each unit of time.
Up to 3 planes can be waiting to land or take off at any time.
How many units of time will the simulation run?5
5

Plane 1 ready to take off.
Plane 2 ready to take off.
Plane 3 ready to land.
Plane 4 ready to land.
1: Plane 3 landed in queue 0 units
1: Plane 4 landed in queue 0 units
Plane 5 ready to take off.
Plane 6 ready to take off.
Plane 6 told to try later.
Plane 7 ready to land.
Plane 8 ready to land.
2: Plane 7 landed in queue 0 units
2: Plane 8 landed in queue 0 units
Plane 9 ready to take off.
Plane 9 told to try later.
Plane 10 ready to take off.
```

TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE

```
Plane 10 told to try later.
Plane 11 ready to land.
Plane 12 ready to land.
3: Plane 11 landed in queue 0 units
3: Plane 12 landed in queue 0 units
Plane 13 ready to take off.
Plane 13 told to try later.
Plane 14 ready to take off.
Plane 14 told to try later.
Plane 15 ready to land.
Plane 16 ready to land.
4: Plane 15 landed in queue 0 units
4: Plane 16 landed in queue 0 units
5: Plane 1 took off in queue 4 units
Result>>>
    Simulation has concluded after 5 units.
    Total number of planes processed: 16
    Number of planes landed: 8
    Number of planes taken off: 1
    Number of planes refused use: 5
    Number left ready to land: 0
    Number left ready to take off: 2
    Percentage of time runway idle: 0.000
    Average wait time to land: 0.000
    Average wait time to take off: 4.000
```

Press Enter key to exit

PS C:\Users\SHATAKSHI\desktop> █

TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE

Plane 4 ready to land.

2: Plane 4 landed in queue 0 units

Plane 5 ready to take off.

Plane 6 ready to land.

3: Plane 6 landed in queue 0 units

Plane 7 ready to take off.

Plane 7 told to try later.

Plane 8 ready to land.

4: Plane 8 landed in queue 0 units

Plane 9 ready to take off.

Plane 9 told to try later.

Plane 10 ready to land.

5: Plane 10 landed in queue 0 units

Result>>>

Simulation has concluded after 5 units.

Total number of planes processed: 10

Number of planes landed: 5

Number of planes taken off: 0

Number of planes refused use: 2

Number left ready to land: 0

Number left ready to take off: 3

Percentage of time runway idle: 0.000

Average wait time to land: 0.000

Press Enter key to exit

PS C:\Users\SHATAKSHI\desktop> █

## Unexpected inputs:

### Rules and Recommendations used-

#### 1. Rule 09. Input Output (FIO)

FIO41-C. Do not call `getc()`, `putc()`, `getwc()`, or `putwc()` with a stream argument that has side effects

```
void clean_stdin() {  
    //Since the default buffer clear function  
    //is system dependent, this function clears the buffer in all cases  
    int c;  
    while ((c = getchar()) != '\n' && c != EOF){}  
}
```

#### 2. Rec. 12. Error Handling (ERR)

ERR00-C. Adopt and implement a consistent and comprehensive error-handling policy

```

while(flag){
    //making sure that the values we take are not meaningless
    //we use a while loop and break out if everything is fine
    printf("How many units of time will the simulation run?");
    scanf("%lf", &temp);
    *totalTime = temp/1;
    //If the input by mistake is given as a float for total time
    //the above lines handle that
    //Since its logically meaningless to give negative values
    //we keep on taking the values as long as there are wrong values
    if(*totalTime < 0){
        printf("These numbers must not be negative.\nPlease re-enter the values\n");
        flag = 1;
    }
    else
        flag=0;
}

printf("\n%d\n", *totalTime);
printf("\n\n\n");
return;
}

```

Though our non compliant program did not show any errors when negative integers or strings were given as input, it also did not redirect the user to provide an acceptable input.

The compliant program asks for input again if unacceptable input is provided till the user gives an appropriate input.

TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell <https://aka.ms/pscore6>

PS C:\Users\SHATAKSHI> cd desktop

PS C:\Users\SHATAKSHI\desktop> gcc client.c server.c server.h

PS C:\Users\SHATAKSHI\desktop> ./a

Program that simulates an airport with only one runway.

One plane can land or depart in each unit of time.

Up to 3 planes can be waiting to land or take off at any time.

How many units of time will the simulation run?-1

These numbers must not be negative.

Please re-enter the values

How many units of time will the simulation run?█

## 8.2 Non-Vulnerable Test Cases

The following the outputs for non-vulnerable inputs:  
For 3 units of time-

```
Press Enter key to exit
PS C:\Users\SHATAKSHI\desktop> gcc client.c server.c server.h
PS C:\Users\SHATAKSHI\desktop> ./a

Program that simulates an airport with only one runway.
One plane can land or depart in each unit of time.
Up to 3 planes can be waiting to land or take off at any time.
How many units of time will the simulation run?3

3

Plane 1 ready to take off.
Plane 2 ready to take off.
Plane 3 ready to land.
Plane 4 ready to land.

1: Plane 3 landed in queue 0 units
1: Plane 4 landed in queue 0 units
2: Plane 1 took off in queue 1 units
3: Plane 2 took off in queue 2 units

Result>>>
Simulation has concluded after 3 units.
Total number of planes processed: 4
Number of planes landed: 2
Number of planes taken off: 2
Number of planes refused use: 0
Number left ready to land: 0
Number left ready to take off: 0
Percentage of time runway idle: 0.000
Average wait time to land: 0.000
Average wait time to take off: 1.500
```

## For 7 units of time-

```
Go Run Terminal Help server.h - Visual Studio Code

TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE

PS C:\Users\SHATAKSHI\desktop> gcc client.c server.c server.h
PS C:\Users\SHATAKSHI\desktop> ./a

Program that simulates an airport with only one runway.
One plane can land or depart in each unit of time.
Up to 3 planes can be waiting to land or take off at any time.
How many units of time will the simulation run?7

Plane 1 ready to take off.
Plane 2 ready to land.
1: Plane 2 landed in queue 0 units
Plane 3 ready to take off.
Plane 4 ready to land.
2: Plane 4 landed in queue 0 units
Plane 5 ready to take off.
Plane 6 ready to take off.
Plane 6 told to try later.
Plane 7 ready to land.
Plane 8 ready to land.
3: Plane 7 landed in queue 0 units
3: Plane 8 landed in queue 0 units
Plane 9 ready to take off.
Plane 9 told to try later.
Plane 10 ready to take off.
```



TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE

Plane 10 ready to take off.

Plane 10 told to try later.

Plane 11 ready to land.

Plane 12 ready to land.

4: Plane 11 landed in queue 0 units

4: Plane 12 landed in queue 0 units

Plane 13 ready to take off.

Plane 13 told to try later.

Plane 14 ready to take off.

Plane 14 told to try later.

Plane 15 ready to land.

Plane 16 ready to land.

5: Plane 15 landed in queue 0 units

5: Plane 16 landed in queue 0 units

Plane 17 ready to take off.

Plane 17 told to try later.

Plane 18 ready to take off.

Plane 18 told to try later.

Plane 19 ready to land.

Plane 20 ready to land.

6: Plane 19 landed in queue 0 units

6: Plane 20 landed in queue 0 units

Plane 21 ready to take off.

## TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE

```
Plane 17 ready to take off.
Plane 17 told to try later.
Plane 18 ready to take off.
Plane 18 told to try later.
Plane 19 ready to land.
Plane 20 ready to land.

6: Plane 19 landed in queue 0 units
6: Plane 20 landed in queue 0 units

Plane 21 ready to take off.
Plane 21 told to try later.
Plane 22 ready to take off.
Plane 22 told to try later.
Plane 23 ready to land.
Plane 24 ready to land.

7: Plane 23 landed in queue 0 units
7: Plane 24 landed in queue 0 units

Result>>>
    Simulation has concluded after 7 units.
    Total number of planes processed: 24
    Number of planes landed: 12
    Number of planes taken off: 0
    Number of planes refused use: 9
    Number left ready to land: 0
    Number left ready to take off: 3
    Percentage of time runway idle: 0.000
    Average wait time to land: 0.000

Press Enter key to exit
PS C:\Users\SHATAKSHI\desktop> |
```

# 9.0 Maintenance

---

## *9.1 Modification*

Our code has been designed keeping modularity as a primary feature, thus if need be it can be modified as per the user requirements. As the macros and defined constants declared in the `server.c`, the user can change their values essentially varying the default presumptions of our airport simulator.

We can modify the waiting and processing time in our code according to user demands. Also the random function which essentially decides the simulation, can also be changed by altering the seed function so the user can generate their self preferred pattern.

## *9.2 Creating Multiple Versions*

Since we've created three separate files for this application, maintenance is fairly easy with its overall layout providing the scope for improvement. Multiple versions with further enhancements of code are possible as the functions have been explicitly specified with their explanations and furthermore there is always scope for further abstraction, thus making it modular.

We've adhered to most recommendations as specified in the coding standards but one can find ways to further incorporate an additional number of them provided they do not alter the overall purpose of the code and in some way benefit the end user in terms of memory usage, security, time etc. In a practical scenario this application can be updated and modified by developers based on their requirements for example - one can specifically assign memory for a plane structure thus creating a new version of the software.

### *9.3 Scalability*

It is the ability of a computer application or product (hardware or software) to continue to function well when it (or its context) is changed in size or volume in order to meet a user's needs. Typically, the rescaling is to a larger size or volume. The rescaling can be of the product itself (for example, a line of computer systems of different sizes in terms of storage, RAM, and so forth) or in the scalable object's movement to a new context (for example, a new operating system).

For this purpose we have taken a few measures such as this piece of code ensures the functioning of the

software in most operating softwares.

```
void clrscr(){
    //This func clears the buffer
    clean_stdin();
    //and also clears the screen when you press any key
    //It is system dependent

    printf("\nPress Enter key to exit\n");
    #ifdef _WIN64
        getchar();
        system("cls");
    #endif
    #ifdef _APPLE_
        getchar();
        system("clear");
    #endif
    #ifdef _linux_
        getchar();
        system("clear");
    #endif
}
```

Whenever we've used any data structures we allocate memory for the data structure based on the user input and hence we are scalable even when a user provides a large input.

We can add more functionalities to the code such as increasing the number of runways to simulate the working of a real airport. And furthermore we can include a functionality which enables reading of real time data about flight arrivals and departures from a database instead of random generation of flights.

# 10.0 Conclusion

---

The CERT security recommendations might be generally ignored or forgotten at our stage, thus this project has been an incredible way to introduce and get in the habit of using those rules and regulations. There might not be the threat of a prevailing cyber attack on our code but still we do have to cross the hurdles of syntactical, logical, run-time and memory errors.

Despite having correct logic and mathematics in our code, during compilation, a buffer overflow or a segmentation dump happen quite often. We cower to acknowledge and correct the bugs, therefore following the CERT we can not only remove them but avoid them like we should and that is what we have implemented and learnt while doing this project.

# 11.0 References

---

- 1) <https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard>
- 2) <https://cwe.mitre.org/index.html>
- 3) Secure Coding in C and C++, Second Edition, Robert C. Seacord
- 4) SEI CERT, C Coding Standard, Rules for Developing Safe, Reliable, and Secure Systems, 2016 Edition, Carnegie Mellon University