ELEVATOR SIMULATOR USING CPP

By

Antonio Balunos Jr.,

Reggie Dimagiba Jr.

Submitted in Partial Fulfillment of the Requirements for the

Parallel and Distributed Computing **CS0051**

At

Instructor: **Hadji Tejuco**

FEU Institute of Technology

January / 27 / 2025

TABLE OF CONTENTS

INTRODUCTION

Purpose: The purpose of this program is to implement the use of threads in C++ through an elevator simulation.

Objective: The main objective of this project is to apply threading concepts in a real-world simulation. The program should be able to process multiple user requests simultaneously and handle floor sorting dynamically using C++ threads and mutex locks.

Scope: This program includes thread implementation in C++ and applies it to an elevator simulation involving multiple users.

PROJECT OVERVIEW

Problem Statement: This project aims to demonstrate the functionality of C++ threads by implementing them in an elevator simulation for a multi-floor building. The simulation showcases how threads can effectively manage concurrent processes. By simulating real-world elevator operations, the project highlights the efficiency and synchronization benefits of multithreading.

Key Features: This project presents a detailed demonstration of how threads manage multiple elevator requests. It highlights the ability of threads to handle concurrent processes, ensuring smooth synchronization when processing multiple user inputs. By simulating real-world elevator operations, it serves as a practical example of using C++ threads for efficient task management.

REQUIREMENTS ANALYSIS

Functional Requirements:

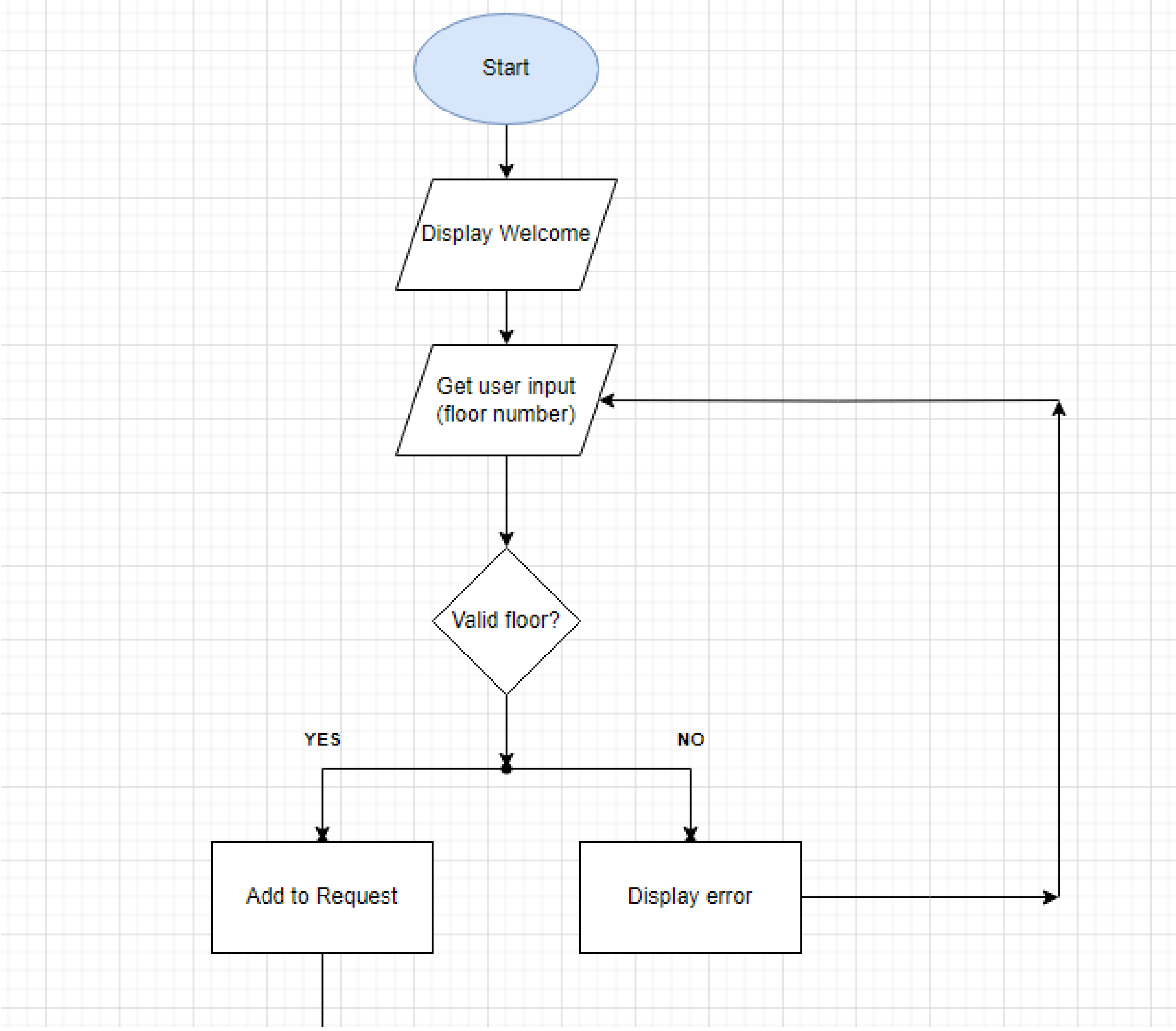
- The system should accept user input for floor requests.
- The elevator must process multiple requests concurrently.

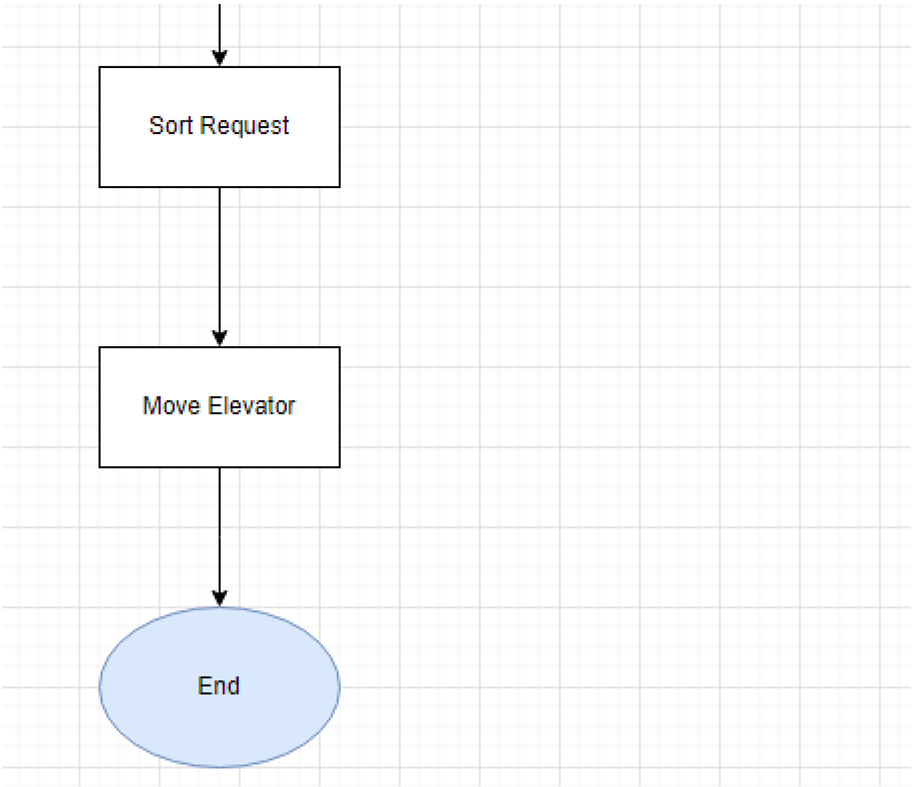
- The system should display elevator movements in real-time.

Non-Functional Requirements:

- The system should efficiently handle concurrent requests using threads.
- The program should be optimized to minimize processing delays.
- The system should ensure thread synchronization using mutex locks.

SYSTEM DESIGN





IMPLEMENTATION

Technologies Used:

Programming Language:

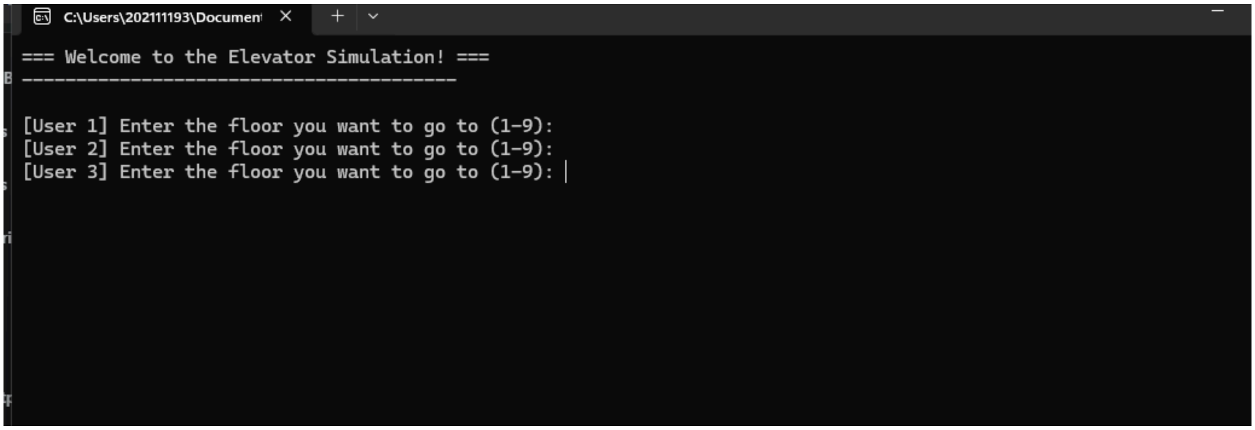
- C++

Libraries:

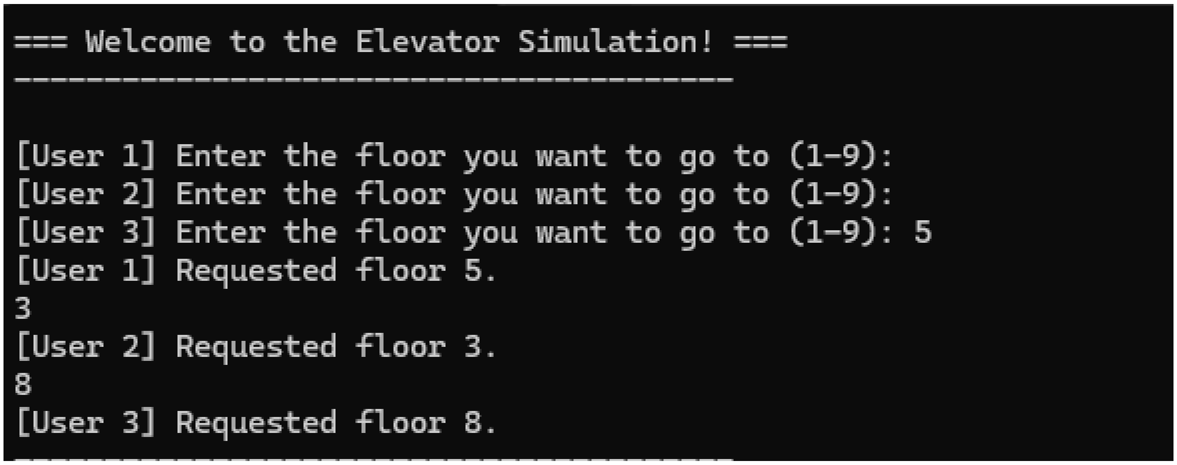
- <thread>
- <mutex>
- <iostream>
- <vector>
- <algorithm>

Tools:

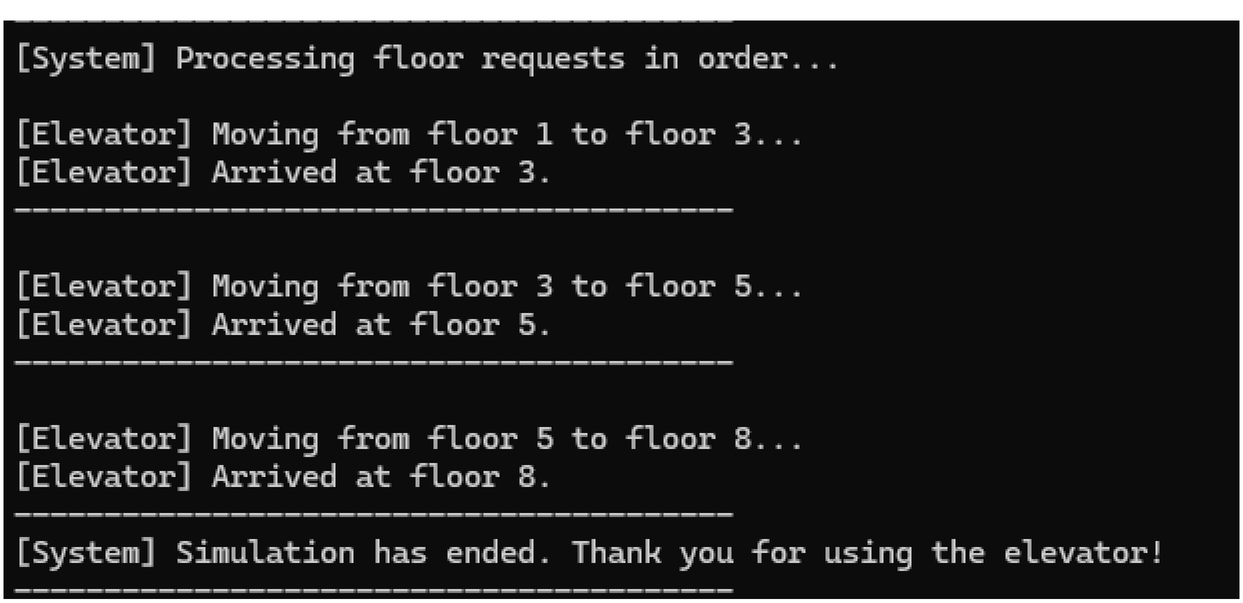
- Code Blocks



```
=== Welcome to the Elevator Simulation! ===
----------------------------------------

[User 1] Enter the floor you want to go to (1-9):
[User 2] Enter the floor you want to go to (1-9):
[User 3] Enter the floor you want to go to (1-9):
```



```
=== Welcome to the Elevator Simulation! ===
----------------------------------------

[User 1] Enter the floor you want to go to (1-9):
[User 2] Enter the floor you want to go to (1-9):
[User 3] Enter the floor you want to go to (1-9): 5
[User 1] Requested floor 5.
3
[User 2] Requested floor 3.
8
[User 3] Requested floor 8.
----------------------------------------
```



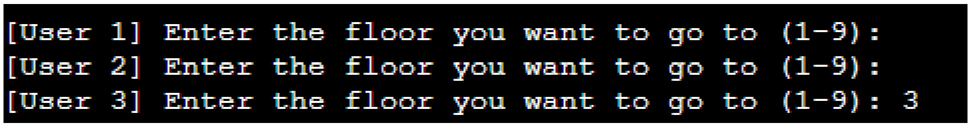
```
----------------------------------------
[System] Processing floor requests in order...

[Elevator] Moving from floor 1 to floor 3...
[Elevator] Arrived at floor 3.
----------------------------------------

[Elevator] Moving from floor 3 to floor 5...
[Elevator] Arrived at floor 5.
----------------------------------------

[Elevator] Moving from floor 5 to floor 8...
[Elevator] Arrived at floor 8.
----------------------------------------
[System] Simulation has ended. Thank you for using the elevator!
----------------------------------------
```

TESTING

Test Cases: List test cases with inputs, expected outputs, and actual results.
CASE:1

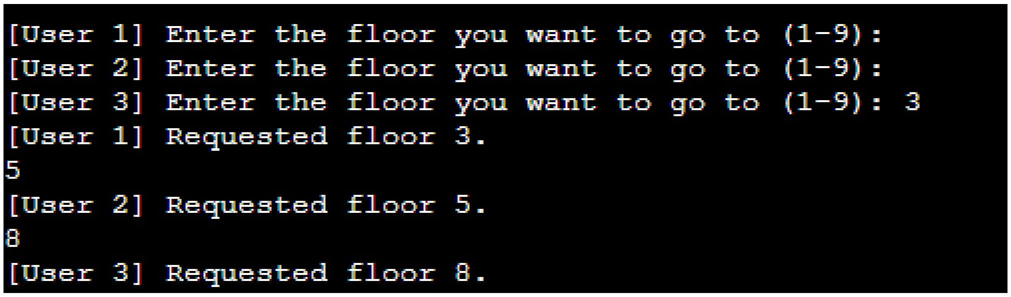
INPUT 1:

Expected Output: Will display the user input(number)



```
[User 1] Enter the floor you want to go to (1-9):
[User 2] Enter the floor you want to go to (1-9):
[User 3] Enter the floor you want to go to (1-9): 3
```

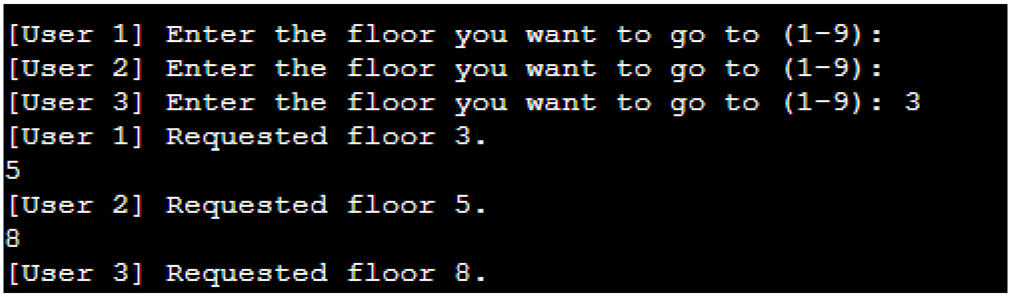
Actual Output: Will display the user input(number)



```
[User 1] Enter the floor you want to go to (1-9):
[User 2] Enter the floor you want to go to (1-9):
[User 3] Enter the floor you want to go to (1-9): 3
[User 1] Requested floor 3.
5
[User 2] Requested floor 5.
8
[User 3] Requested floor 8.
```

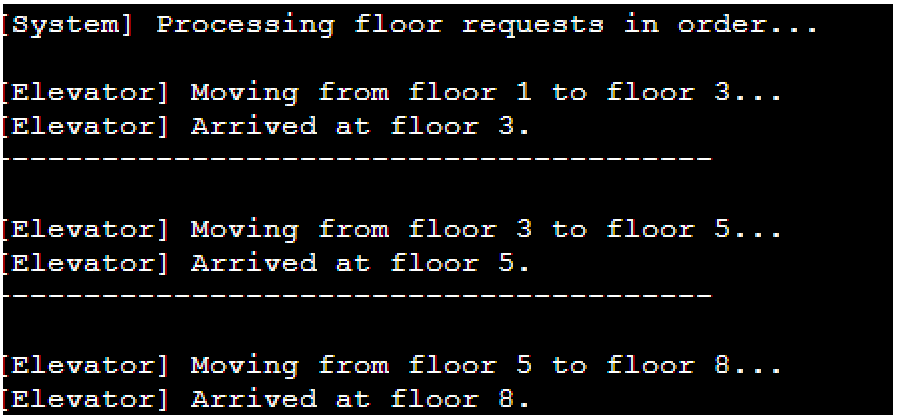
INPUT 2:

Expected Output: Display the process that the elevator moving



```
[User 1] Enter the floor you want to go to (1-9):
[User 2] Enter the floor you want to go to (1-9):
[User 3] Enter the floor you want to go to (1-9): 3
[User 1] Requested floor 3.
5
[User 2] Requested floor 5.
8
[User 3] Requested floor 8.
```

Actual Output: Display the process that the elevator moving

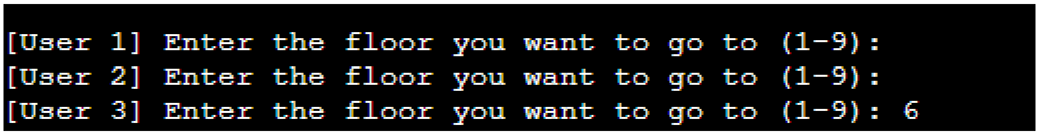


```
[System] Processing floor requests in order...

[Elevator] Moving from floor 1 to floor 3...
[Elevator] Arrived at floor 3.
-------------------------------------------


[Elevator] Moving from floor 3 to floor 5...
[Elevator] Arrived at floor 5.
-------------------------------------------


[Elevator] Moving from floor 5 to floor 8...
[Elevator] Arrived at floor 8.
```

CASE:2

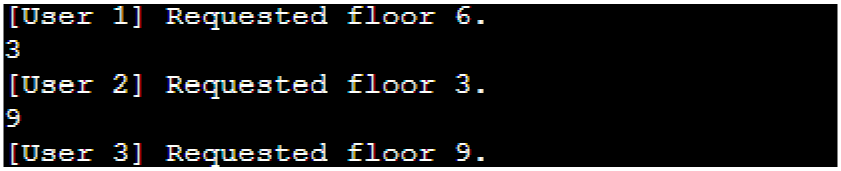
INPUT 1:

Expected Output: Will display the user input(number)



```
[User 1] Enter the floor you want to go to (1-9):
[User 2] Enter the floor you want to go to (1-9):
[User 3] Enter the floor you want to go to (1-9): 6
```

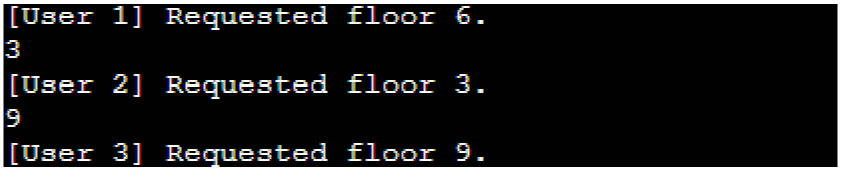
Actual Output: Will display the user input(number)



```
[User 1] Requested floor 6.
3
[User 2] Requested floor 3.
9
[User 3] Requested floor 9.
```

INPUT 2:

Expected Output: Display the process that the elevator moving and making it into Sequential order



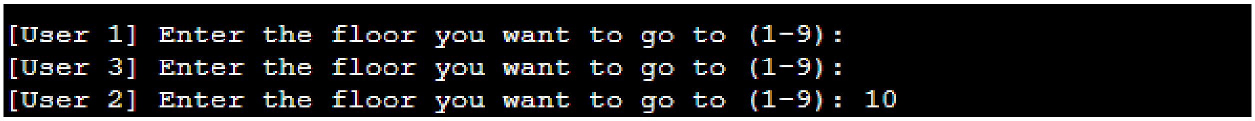
```
[User 1] Requested floor 6.
3
[User 2] Requested floor 3.
9
[User 3] Requested floor 9.
```

Actual Output: Display the process that the elevator moving and making it into Sequential order

CASE:3

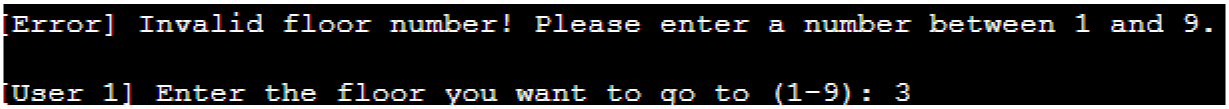
INPUT 1:

Expected Output: Will display the user input(number)



```
[User 1] Enter the floor you want to go to (1-9):
[User 3] Enter the floor you want to go to (1-9):
[User 2] Enter the floor you want to go to (1-9): 10
```

Actual Output: Error if the input was above 9

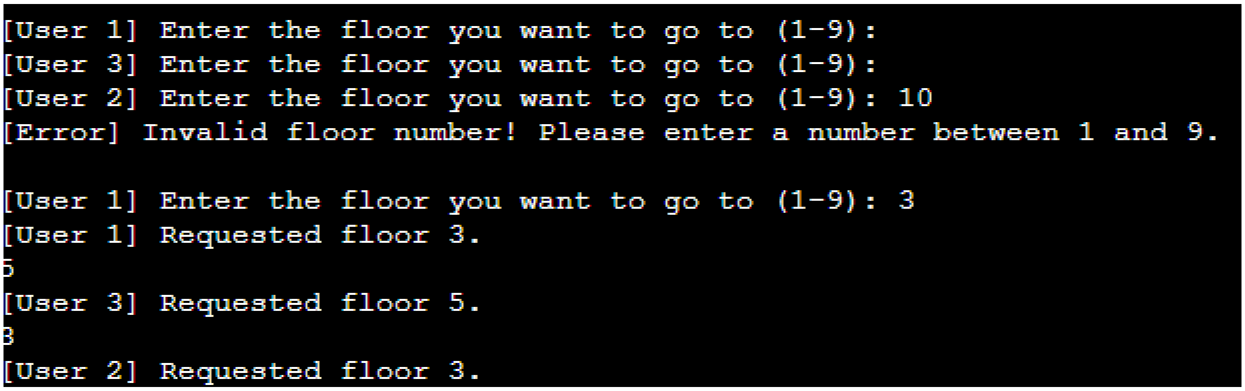


```
[Error] Invalid floor number! Please enter a number between 1 and 9.

[User 1] Enter the floor you want to go to (1-9): 3
```

INPUT 2:

Expected Output: Will display the user input(number)

Actual Output: Will make you retype the error floor number into a valid number and proceed

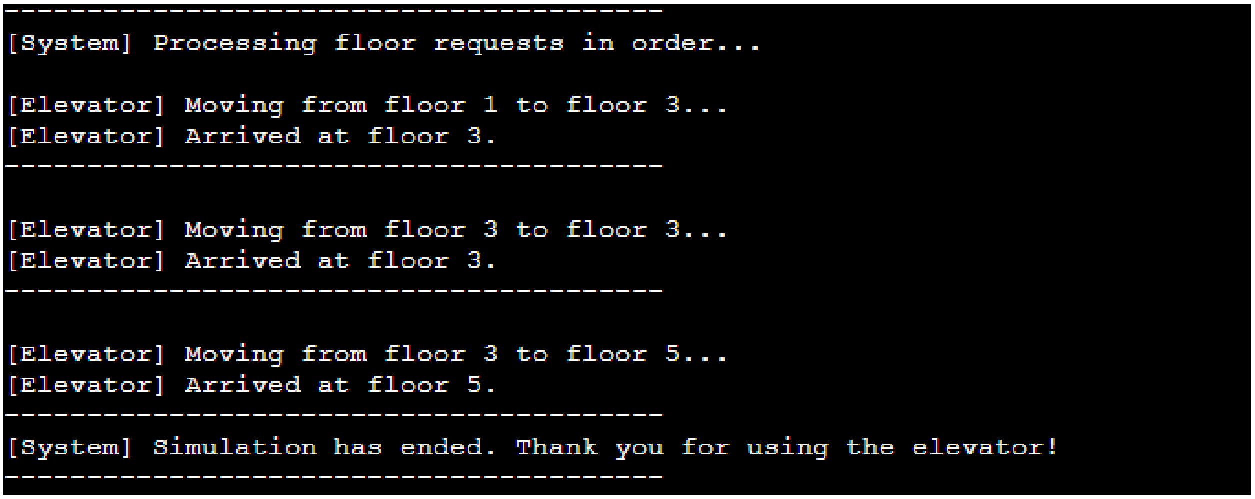


```
[User 1] Enter the floor you want to go to (1-9):
[User 3] Enter the floor you want to go to (1-9):
[User 2] Enter the floor you want to go to (1-9): 10
[Error] Invalid floor number! Please enter a number between 1 and 9.

[User 1] Enter the floor you want to go to (1-9): 3
[User 1] Requested floor 3.
5
[User 3] Requested floor 5.
3
[User 2] Requested floor 3.
```

INPUT 3:

Expected Output: Display the process that the elevator moving and making it into Sequential order

Actual Output: Display the process that the elevator moving and making it into Sequential order



```
----------------------------------------
[System] Processing floor requests in order...

[Elevator] Moving from floor 1 to floor 3...
[Elevator] Arrived at floor 3.
----------------------------------------

[Elevator] Moving from floor 3 to floor 3...
[Elevator] Arrived at floor 3.
----------------------------------------

[Elevator] Moving from floor 3 to floor 5...
[Elevator] Arrived at floor 5.
----------------------------------------
[System] Simulation has ended. Thank you for using the elevator!
----------------------------------------
```

Result: The first case shows that any floor up to 1 to 9 will be accessible and goes into a Sequential order while the second case the input is in a random order it will re organize it into a Sequential order and for the last case if the user input is above 9 it will show that the number input is invalid and must put in the valid number to proceed.

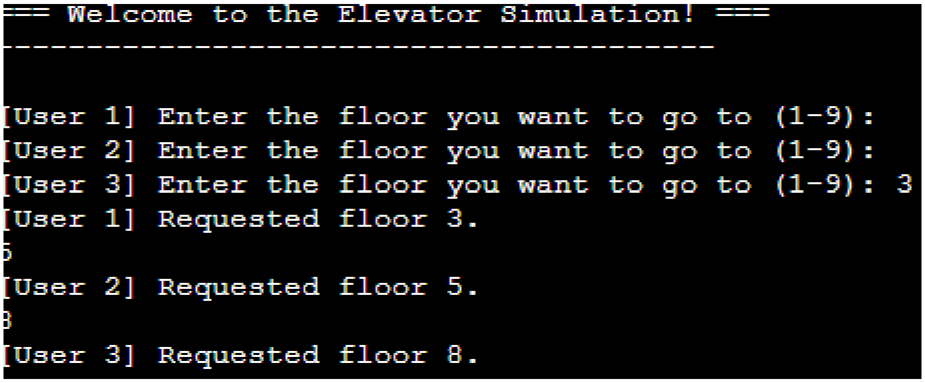
USER MANUAL

Installation Guide:

1. Download and extract the C++ project.
2. Open the project in Code::Blocks or VS Code.
3. Compile and run the elevator_simulation.cpp file.

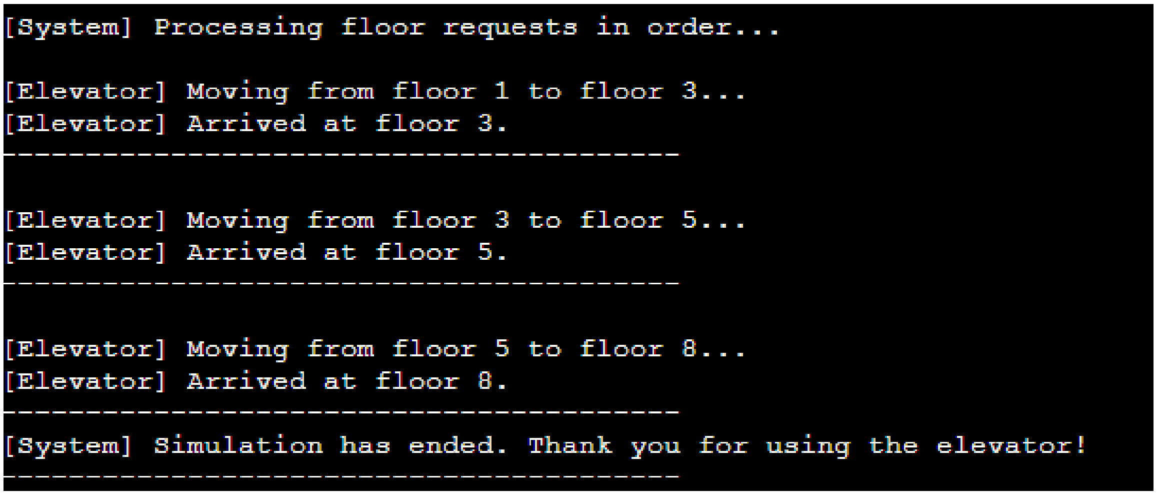
Usage Instructions:

1. Run the program.
2. Enter the floor number when prompted.



```
=== Welcome to the Elevator Simulation! ===
------------------------------------------

[User 1] Enter the floor you want to go to (1-9):
[User 2] Enter the floor you want to go to (1-9):
[User 3] Enter the floor you want to go to (1-9): 3
[User 1] Requested floor 3.
5
[User 2] Requested floor 5.
8
[User 3] Requested floor 8.
```

3. Observe the elevator processing requests in real-time.



```
[System] Processing floor requests in order...

[Elevator] Moving from floor 1 to floor 3...
[Elevator] Arrived at floor 3.
----------------------------------------

[Elevator] Moving from floor 3 to floor 5...
[Elevator] Arrived at floor 5.
----------------------------------------

[Elevator] Moving from floor 5 to floor 8...
[Elevator] Arrived at floor 8.
----------------------------------------
[System] Simulation has ended. Thank you for using the elevator!
----------------------------------------
```

CHALLENGES AND SOLUTION

Challenges Faced:

- Managing concurrent user inputs.
- Ensuring proper thread synchronization.
- Implementing an efficient floor request sorting system.

Solutions Implemented:

- Used std::mutex for safe thread synchronization.
- Implemented a sorting algorithm to process floor requests in order.

FUTURE ENHANCEMENTS

Potential Improvements:

- Implement a graphical user interface (GUI).
- Support multiple elevators for improved realism.

CONCLUSION

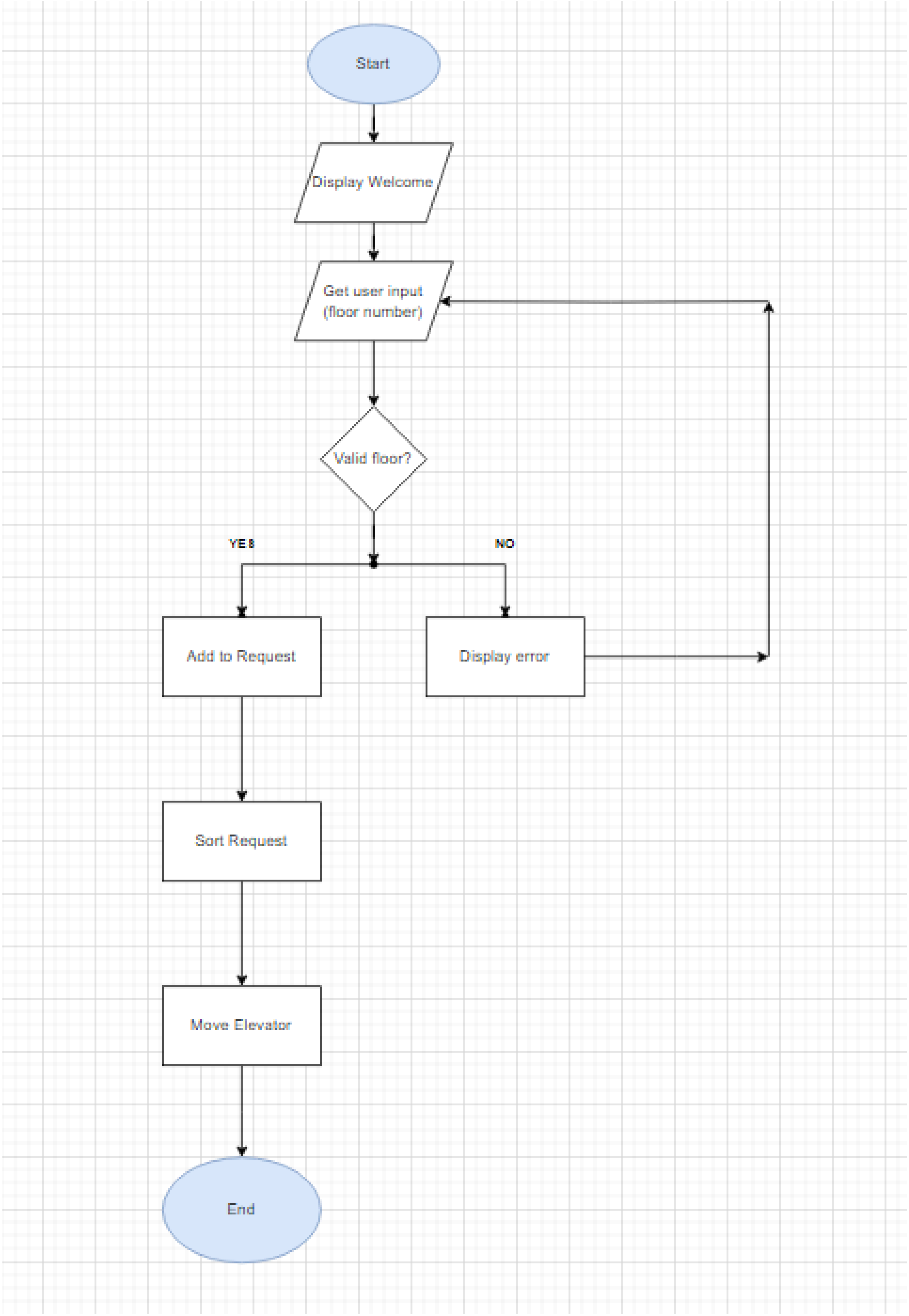
The Elevator Simulation successfully demonstrates the use of multithreading, mutex locks, and request handling in C++. It provides an interactive way to understand concurrency concepts and system synchronization.

REFERENCES:

https://www.w3schools.com/cpp/cpp_algorithms.asp
https://www.geeksforgeeks.org/chrono-in-c/
https://www.w3schools.com/cpp/cpp_vectors.asp
https://github.com/TechTutorialHub/CS0051/blob/main/Module2/Code7.cpp

APPENDICES:

APPENDIX A



APPENDIX B

```
#include <iostream>

#include <thread>

#include <mutex>

#include <vector>

#include <algorithm>

#include <chrono>


std::mutex mtx;

int current_floor = 1;

const int NUM_FLOORS = 9;

std::vector<int> requests;



void print_separator() {

std::cout << "----------------------------------------\n";

}


// elevator moving

void move_to_floor(int requested_floor) {

std::lock_guard<std::mutex> lock(mtx);

std::cout << "\n[Elevator] Moving from floor " << current_floor

<< " to floor " << requested_floor << "...\n";
```

```
std::this_thread::sleep_for(std::chrono::milliseconds(std::abs(current_floor - requested_floor) * 500));
current_floor = requested_floor;
std::cout << "[Elevator] Arrived at floor " << current_floor << ".\n";
print_separator();
}


// User floor request simulation
void request_simulation(int user_id) {
int requested_floor;
do {
{
std::lock_guard<std::mutex> lock(mtx);
std::cout << "\n[User " << user_id
<< "] Enter the floor you want to go to (1-" << NUM_FLOORS << "): ";
}
std::cin >> requested_floor;


if (requested_floor < 1 || requested_floor > NUM_FLOORS) {
std::lock_guard<std::mutex> lock(mtx);
std::cout << "[Error] Invalid floor number! Please enter a number between 1 and " // if the floor is above 9
<< NUM_FLOORS << ".\n";
}
} while (requested_floor < 1 || requested_floor > NUM_FLOORS);


{

std::lock_guard<std::mutex> lock(mtx);

std::cout << "[User " << user_id << "] Requested floor " << requested_floor << ".\n";

requests.push_back(requested_floor);

}

}


int main() {

std::cout << "=== Welcome to the Elevator Simulation! ===\n";

print_separator();


// user input

std::thread user1(request_simulation, 1);

std::thread user2(request_simulation, 2);

std::thread user3(request_simulation, 3);


// wait for all users to finish

user1.join();

user2.join();

user3.join();
```

```
print_separator();
std::cout << "[System] Processing floor requests in order...\n";

// sort requests in ascending order
{
std::lock_guard<std::mutex> lock(mtx);
std::sort(requests.begin(), requests.end());
}

// process requests in order
for (int floor : requests) {
move_to_floor(floor);
}

std::cout << "[System] Simulation has ended. Thank you for using the elevator!\n";
print_separator();

return 0;
}
```