



FEU Institute of Technology
COLLEGE OF ENGINEERING • COLLEGE OF COMPUTER STUDIES

GROUP 9 - ELEVATOR

Project Documentation

CS0051

Submitted by:

Rene Vincent B. Cosme - Developer

Ritz John S. Layderos - Developer

February 8, 2025

Project Documentation

1. Title Page

Project Title: GROUP 9 - Elevator Simulation

Course Name: CS0051

Instructor: Mr. Hadji Tejuco

Team Members:

Rene Vincent B. Cosme, Developer

Ritz John S. Layderos, Developer

Submission Date: February 8, 2025

2. Table of Contents

Introduction

Project Overview

Requirements Analysis

System Design

Implementation

Testing

User Manual

Challenges and Solutions

Future Enhancements

Conclusion

References

Appendices

3. Introduction

Purpose:

The purpose of this project is to simulate the operation of an elevator system, focusing on managing passenger movement efficiently while simulating real-world scenarios. It aims to provide an interactive tool for understanding how elevators handle multiple passengers, floor stops, and capacity constraints, which is significant for both

educational and operational analysis in building management systems. Additionally, the project incorporates threads and mutexes to ensure smooth, concurrent operations, providing an opportunity to explore key concepts of thread synchronization and resource sharing.

Objectives:

1. To simulate elevator movement between floors based on passenger destinations.
2. To manage and update passenger information, including boarding, disembarking, and capacity limits.
3. To provide real-time status updates on the elevator's operation.
4. To ensure efficient operation even with multiple passengers and floors.

Scope:

The scope of the project includes simulating an elevator that operates in a building with 9 floors, handling up to 20 passengers. It covers managing passenger boarding and disembarking, real-time elevator status updates, and ensuring efficient capacity management. The project does not cover physical hardware integration, such as actual elevator control systems or advanced AI for predictive scheduling. Future extensions may include additional features like multiple elevators, advanced optimization, or graphical user interfaces.

4. Project Overview

Problem Statement

The project addresses the challenge of simulating an elevator system that manages the movement of passengers between multiple floors efficiently. It focuses on optimizing the elevator's direction and stops based on passenger destinations, ensuring smooth operation even when the elevator is idle or when there are passengers waiting on different floors. The system must also handle constraints such as passenger capacity and provide a dynamic response to changing conditions.

Key Features

The software simulates elevator movement, picking up and dropping off passengers at the appropriate floors based on their target destinations. It ensures the

elevator operates within its capacity limits and adjusts its direction according to the needs of passengers, either moving up or down depending on the direction of waiting passengers. The simulation runs concurrently using threads to handle the elevator's operation, and mutexes are employed to synchronize access to shared resources, preventing race conditions. Real-time updates are displayed, showing the elevator's current floor, direction, and the number of passengers onboard. The simulation continues until all passengers have reached their destinations, and no passengers are left waiting.

5. Requirements Analysis

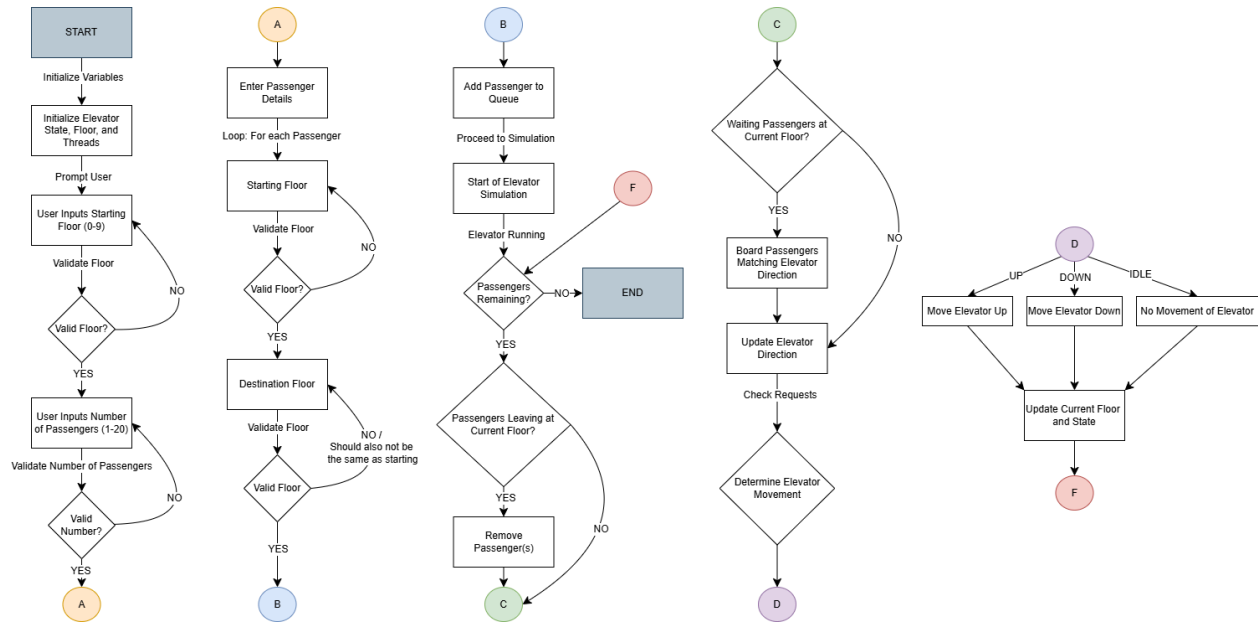
Functional Requirements:

1. Simulate the elevator's movement between floors, stopping where passengers need to board or exit.
2. Manage passenger capacity, ensuring it does not exceed the limit.
3. Provide real-time updates on the elevator's current status (floor, direction, passengers).
4. Determine the elevator's direction based on passenger destinations.
5. End the simulation once all passengers have reached their destinations.

Non-Functional Requirements:

1. The system must perform efficiently, handling up to 20 passengers and 9 floors smoothly.
2. The user interface should be intuitive, ensuring easy interaction with the system.
3. The system must securely handle user inputs, preventing invalid data entry.
4. The system should be reliable and stable, running without crashes or errors.
5. The system should be scalable, allowing easy integration of future features, such as multiple elevators or advanced scheduling.

6. System Design



7. Implementation

Programming Language: C++

Libraries and Frameworks:

- Standard Template Library (STL)
- `<thread>` for multithreading
- `<mutex>` for thread synchronization
- `<queue>` for managing passenger queues
- `<condition_variable>` for synchronization

Tools:

- IDE: Visual Studio / GDB online Debugger
- Compiler: GCC/ G++

Version Control: Git

Screenshots:

```

Welcome to the Elevator Simulation!

Enter the starting floor for the elevator (0-9, 0 = Ground Floor): 0
Enter the number of passengers (1-20): 4

Enter passenger details:

Passenger 1:
Starting floor (0-9, 0 = Ground Floor): 0
Destination floor (0-9, 0 = Ground Floor): 6
Added passenger 1 waiting at floor G going to floor 6

Passenger 2:
Starting floor (0-9, 0 = Ground Floor): 0
Destination floor (0-9, 0 = Ground Floor): 7
Added passenger 2 waiting at floor G going to floor 7

Passenger 3:
Starting floor (0-9, 0 = Ground Floor): 6
Destination floor (0-9, 0 = Ground Floor): 2
Added passenger 3 waiting at floor 6 going to floor 2

Passenger 4:
Starting floor (0-9, 0 = Ground Floor): 9
Destination floor (0-9, 0 = Ground Floor): 0
Added passenger 4 waiting at floor 9 going to floor G

```

```

=== Elevator Status ===
Current Floor: Ground
Direction: UP
Passengers in elevator: 2/9
Total passengers processed: 0/4

Waiting Passengers:
Floor 9: 1 waiting
Floor 8: None
Floor 7: None
Floor 6: 1 waiting
Floor 5: None
Floor 4: None
Floor 3: None
Floor 2: None
Floor 1: None
Floor G: None
=====

```

```

=== Elevator Status ===
Current Floor: 6
Direction: UP
Passengers in elevator: 1/9
Total passengers processed: 1/4

Waiting Passengers:
Floor 9: 1 waiting
Floor 8: None
Floor 7: None
Floor 6: 1 waiting
Floor 5: None
Floor 4: None
Floor 3: None
Floor 2: None
Floor 1: None
Floor G: None
=====

```

```

=== Elevator Status ===
Current Floor: 8
Direction: DOWN
Passengers in elevator: 0/9
Total passengers processed: 2/4

Waiting Passengers:
Floor 9: 1 waiting
Floor 8: None
Floor 7: None
Floor 6: 1 waiting
Floor 5: None
Floor 4: None
Floor 3: None
Floor 2: None
Floor 1: None
Floor G: None
=====

```

```

=== Elevator Status ===
Current Floor: 4
Direction: UP
Passengers in elevator: 0/9
Total passengers processed: 3/4

Waiting Passengers:
Floor 9: 1 waiting
Floor 8: None
Floor 7: None
Floor 6: None
Floor 5: None
Floor 4: None
Floor 3: None
Floor 2: None
Floor 1: None
Floor G: None
=====

```

```
=== Elevator Status ===
Current Floor: Ground
Direction: IDLE
Passengers in elevator: 0/9
Total passengers processed: 4/4

Waiting Passengers:
Floor 9: None
Floor 8: None
Floor 7: None
Floor 6: None
Floor 5: None
Floor 4: None
Floor 3: None
Floor 2: None
Floor 1: None
Floor G: None
=====

Simulation complete! All passengers have reached their destinations.
```

```
Welcome to the Elevator Simulation!

Enter the starting floor for the elevator (0-9, 0 = Ground Floor): a
Invalid floor! Please enter a number between 0 (Ground Floor) and 9.
Enter the starting floor for the elevator (0-9, 0 = Ground Floor): -1
Invalid floor! Please enter a number between 0 (Ground Floor) and 9.
Enter the starting floor for the elevator (0-9, 0 = Ground Floor): 10
Invalid floor! Please enter a number between 0 (Ground Floor) and 9.
Enter the starting floor for the elevator (0-9, 0 = Ground Floor): 0
Enter the number of passengers (1-20): a
Invalid input! Please enter a numerical value between 1 and 20.
Enter the number of passengers (1-20): -2
Invalid input! Please enter a numerical value between 1 and 20.
Enter the number of passengers (1-20): 21
Number must be between 1 and 20!
Enter the number of passengers (1-20): 1

Enter passenger details:

Passenger 1:
Starting floor (0-9, 0 = Ground Floor): a
Invalid floor! Please enter a number between 0 (Ground Floor) and 9.
Starting floor (0-9, 0 = Ground Floor): -1
Invalid floor! Please enter a number between 0 (Ground Floor) and 9.
Starting floor (0-9, 0 = Ground Floor): 11
Invalid floor! Please enter a number between 0 (Ground Floor) and 9.
Starting floor (0-9, 0 = Ground Floor): 0
Destination floor (0-9, 0 = Ground Floor): 0
Destination floor must be different from starting floor!
Destination floor (0-9, 0 = Ground Floor): -1
Invalid floor! Please enter a number between 0 (Ground Floor) and 9.
Destination floor (0-9, 0 = Ground Floor): a
Invalid floor! Please enter a number between 0 (Ground Floor) and 9.
Destination floor (0-9, 0 = Ground Floor): 12
Invalid floor! Please enter a number between 0 (Ground Floor) and 9.
Destination floor (0-9, 0 = Ground Floor): 2
```

8. Testing

Test Cases:

Test Case#1: Single passenger (Going up)

Input: Starting floor: 0 (Ground floor)

Passenger: From floor 0 to floor 5

Expected Output: The elevator should move to floor 5, drop off the passenger, and stop.

Result: The elevator correctly moved from floor 0 to floor 5, picking up and dropping off the passenger.

Test Case#2: Single passenger (Going down)

Input: Starting floor: 0 (Ground floor)

Passenger: From floor 9 to floor 5

Expected Output: The elevator should move to floor 5, drop off the passenger, and stop.

Result: The elevator correctly moved from floor 9 to floor 5, picking up and dropping off the passenger.

Test Case#3: Multiple passenger (Going up and down)

Input: Starting floor: 0 (Ground floor)

Passenger 1: From floor 0 to floor 9

Passenger 2: From floor 5 to floor 9

Passenger 3: From floor 7 to floor 3

Expected Output: The elevator should:

- pick up the passenger 1 on floor 1
- pick up the passenger 2 on floor 5
- proceed to floor 9 to drop off both passengers
- go down to floor 7 to pick up passenger 3 and drop off on floor 3.

Result: The elevator operated as intended.

Test Case#4: Full Capacity

Input: Starting floor: 0 (Ground floor)

Passengers: 9 passengers from ground floor and 1 from any floor. All passengers are going to the 9th floor.

Expected Output: The elevator should be able to pick up all 9 passengers and move up to 9th floor without picking up the other passenger.

Result: The elevator successfully handled all passengers without exceeding the capacity.

Test Case#5: Inputting a non-numerical value

Input: Starting floor: a

Expected Output: The program will not terminate and will prompt the user to enter a valid input.

Result: The message *"Invalid floor! Please enter a number between 0 (Ground Floor) and 9."* will be displayed until the user enters a valid input.

Test Case#6: Inputting a same floor value

Input: Starting floor: 5

Destination floor: 5

Expected Output: The program will not terminate and will prompt the user to enter a valid input.

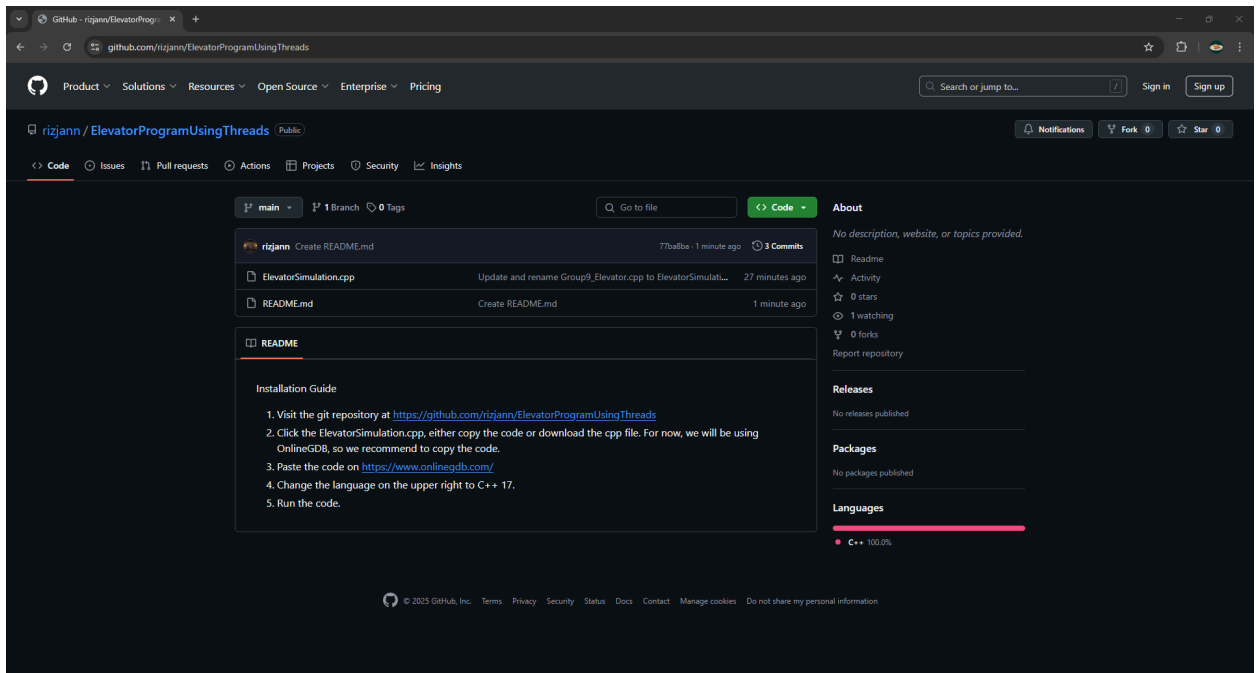
Result: The message *"Destination floor must be different from starting floor!"* will be displayed until the user enters a valid input.

9. User Manual

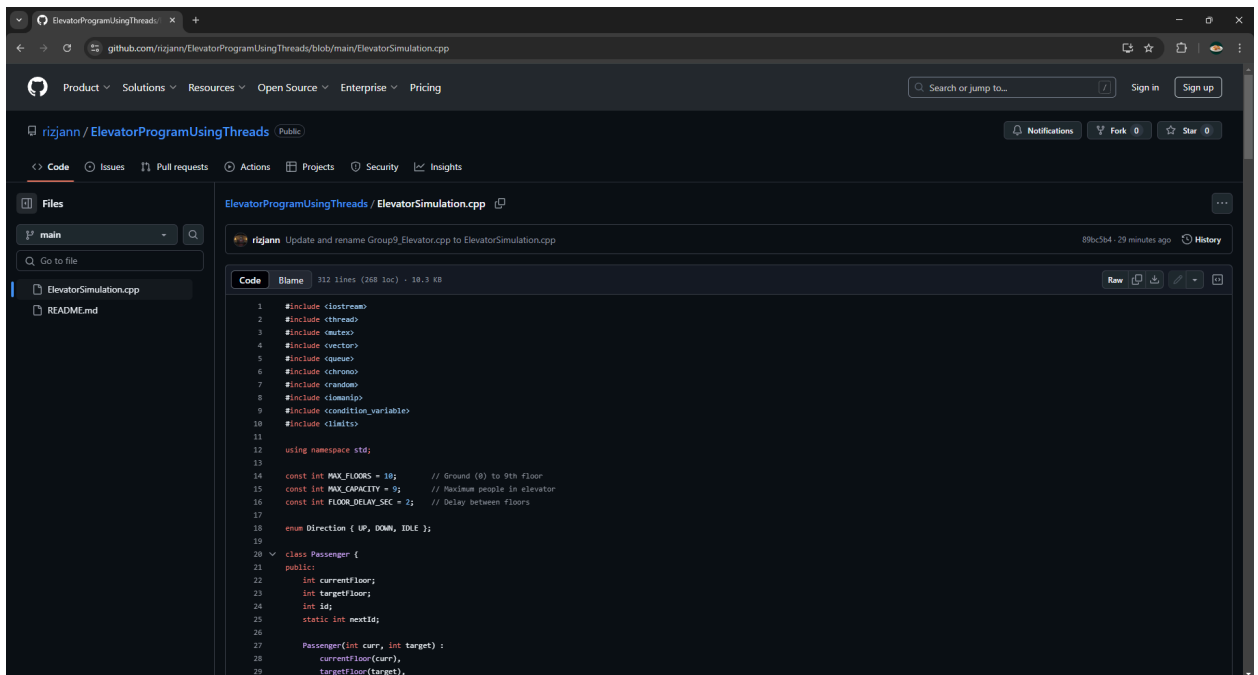
Installation Guide:

1. Visit the git repository at

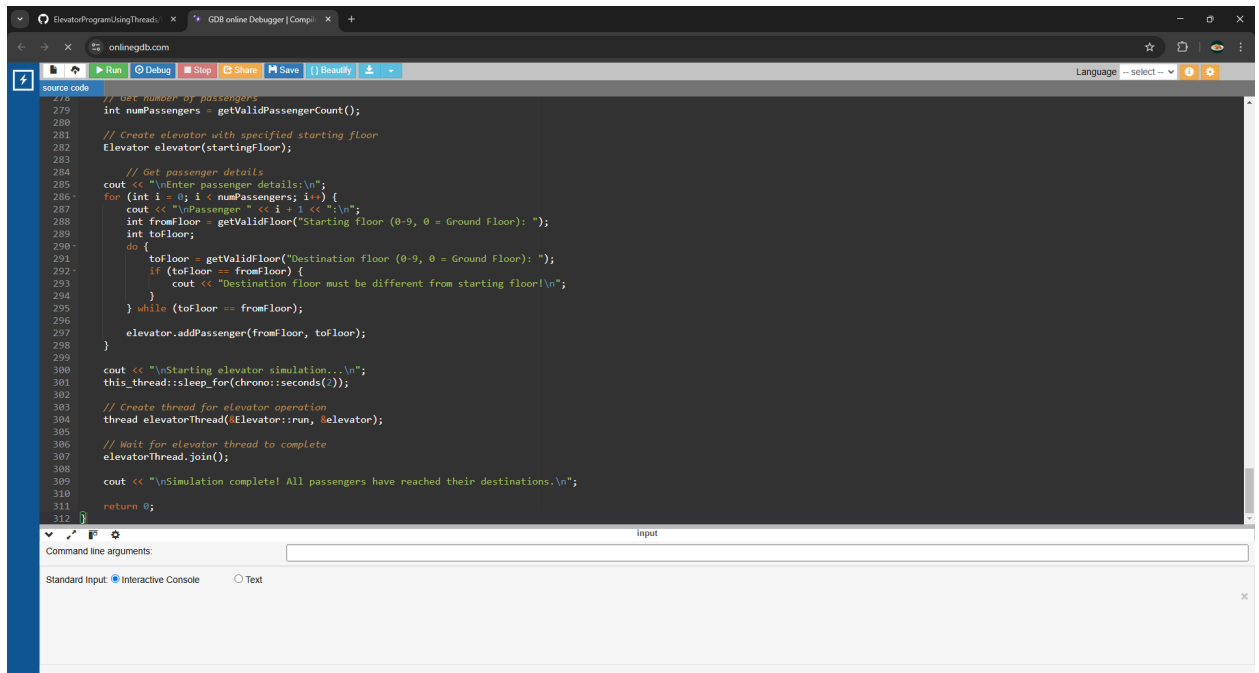
<https://github.com/rizjann/ElevatorProgramUsingThreads>



2. Click the ElevatorSimulation.cpp, either copy the code or download the cpp file.
For now, we will be using OnlineGDB, so we recommend to copy the code.

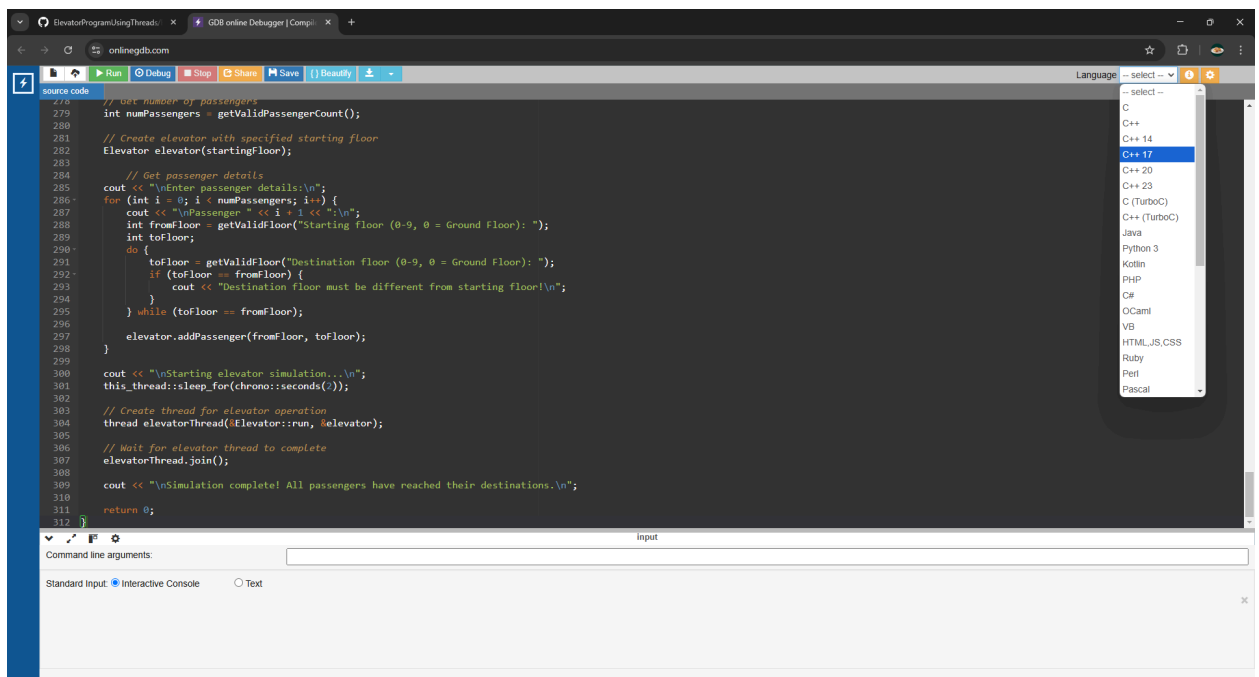


3. Paste the code on <https://www.onlinegdb.com/>



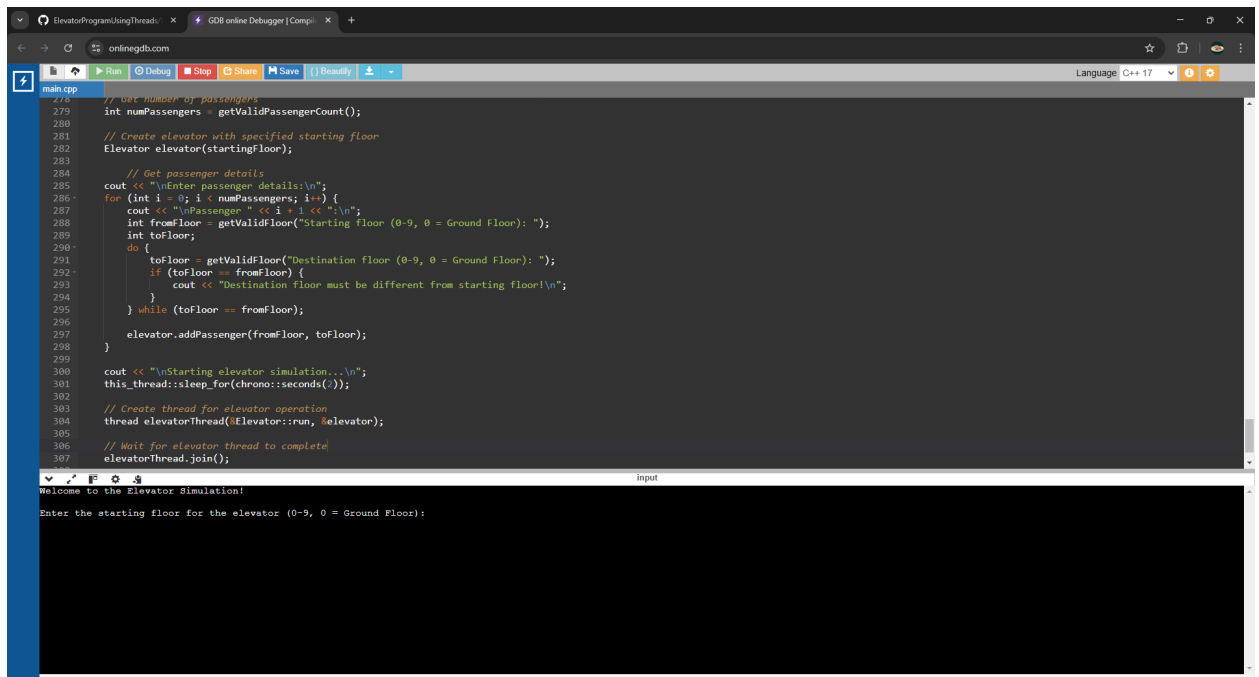
```
278 // Get number of passengers
279 int numPassengers = getValidPassengerCount();
280
281 // Create elevator with specified starting floor
282 Elevator elevator(startingFloor);
283
284 // Get passenger details
285 cout << "\nEnter passenger details:\n";
286 for (int i = 0; i < numPassengers; i++) {
287     cout << "Passenger " << i + 1 << ":\n";
288     int fromFloor = getValidFloor("Starting floor (0-9, 0 = Ground Floor): ");
289     int toFloor;
290     do {
291         toFloor = getValidFloor("Destination floor (0-9, 0 = Ground Floor): ");
292         if (toFloor == fromFloor) {
293             cout << "Destination floor must be different from starting floor!\n";
294         }
295     } while (toFloor == fromFloor);
296     elevator.addPassenger(fromFloor, toFloor);
297 }
298
299 cout << "\nStarting elevator simulation...\n";
300 this_thread::sleep_for(chrono::seconds(2));
301
302 // Create thread for elevator operation
303 thread elevatorThread(&Elevator::run, &elevator);
304
305 // Wait for elevator thread to complete
306 elevatorThread.join();
307
308 cout << "\nSimulation complete! All passengers have reached their destinations.\n";
309
310 return 0;
311 }
```

4. Change the language on the upper right to C++ 17.



```
278 // Get number of passengers
279 int numPassengers = getValidPassengerCount();
280
281 // Create elevator with specified starting floor
282 Elevator elevator(startingFloor);
283
284 // Get passenger details
285 cout << "\nEnter passenger details:\n";
286 for (int i = 0; i < numPassengers; i++) {
287     cout << "Passenger " << i + 1 << ":\n";
288     int fromFloor = getValidFloor("Starting floor (0-9, 0 = Ground Floor): ");
289     int toFloor;
290     do {
291         toFloor = getValidFloor("Destination floor (0-9, 0 = Ground Floor): ");
292         if (toFloor == fromFloor) {
293             cout << "Destination floor must be different from starting floor!\n";
294         }
295     } while (toFloor == fromFloor);
296     elevator.addPassenger(fromFloor, toFloor);
297 }
298
299 cout << "\nStarting elevator simulation...\n";
300 this_thread::sleep_for(chrono::seconds(2));
301
302 // Create thread for elevator operation
303 thread elevatorThread(&Elevator::run, &elevator);
304
305 // Wait for elevator thread to complete
306 elevatorThread.join();
307
308 cout << "\nSimulation complete! All passengers have reached their destinations.\n";
309
310 return 0;
311 }
```

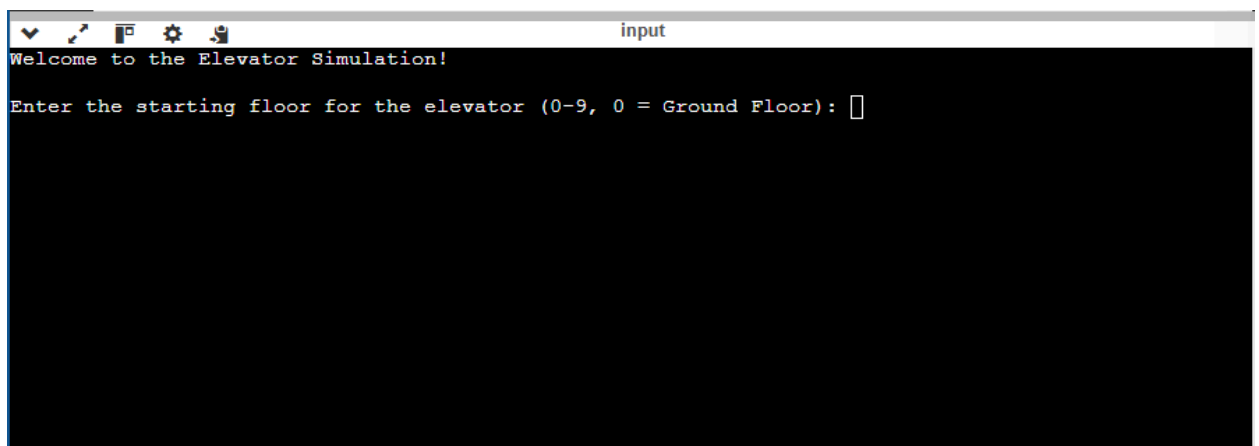
5. Run the code.



```
275 // Get number of passengers
276 int numPassengers = getValidPassengerCount();
277
278 // Create elevator with specified starting floor
279 Elevator elevator(startingFloor);
280
281 // Get passenger details
282 cout << "\nEnter passenger details:\n";
283 for (int i = 0; i < numPassengers; i++) {
284     cout << "\nPassenger " << i + 1 << ":\n";
285     int fromFloor = getValidFloor("Starting floor (0-9, 0 = Ground Floor): ");
286     int toFloor;
287     do {
288         toFloor = getValidFloor("Destination floor (0-9, 0 = Ground Floor): ");
289         if (toFloor == fromFloor) {
290             cout << "Destination floor must be different from starting floor!\n";
291         }
292     } while (toFloor == fromFloor);
293     elevator.addPassenger(fromFloor, toFloor);
294 }
295
296 cout << "\nStarting elevator simulation...\n";
297 this_thread::sleep_for(chrono::seconds(2));
298
299 // Create thread for elevator operation
300 thread elevatorThread(&Elevator::run, &elevator);
301
302 // Wait for elevator thread to complete
303 elevatorThread.join();
```

Usage Instructions:

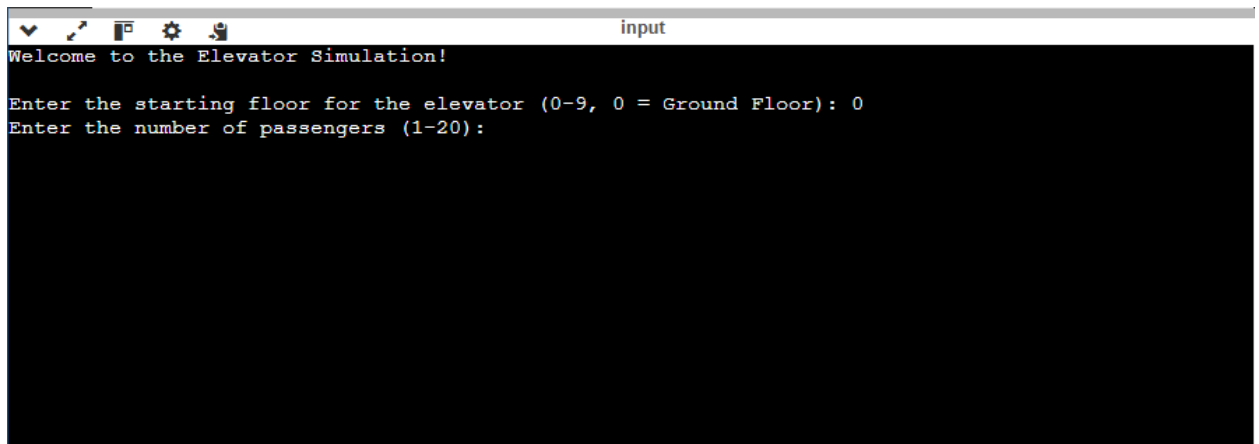
- Starting the Floor of the elevator.** Pick which Floor the elevator is on, and 0 is the Ground Floor. (0-9)



```
Welcome to the Elevator Simulation!

Enter the starting floor for the elevator (0-9, 0 = Ground Floor): 
```

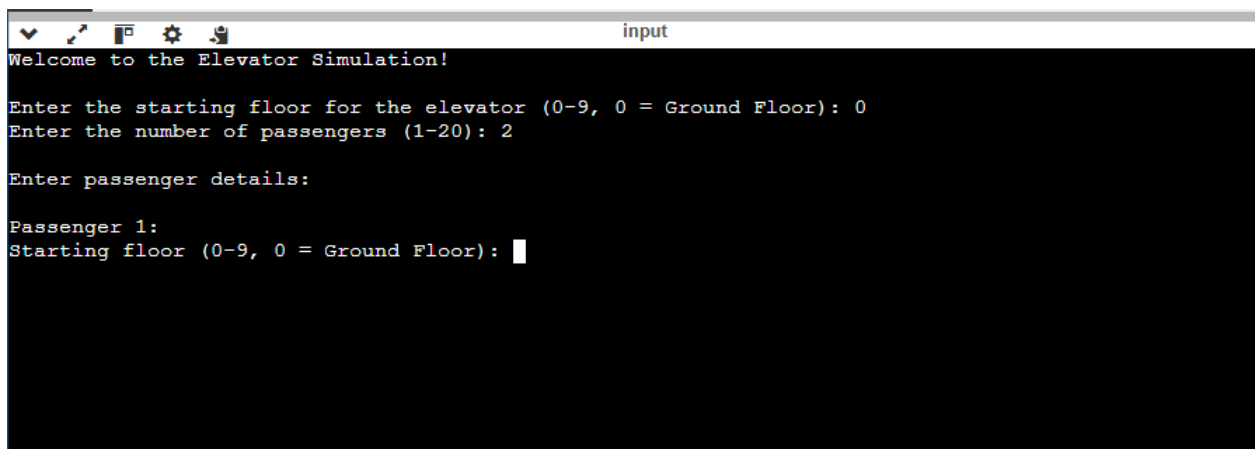
- Number of passengers.** Enter the number of passengers from 1 to 20.



```
input
Welcome to the Elevator Simulation!

Enter the starting floor for the elevator (0-9, 0 = Ground Floor): 0
Enter the number of passengers (1-20):
```

- c. **Starting Floor of the passenger.** For each of the passengers, enter the starting Floor.



```
input
Welcome to the Elevator Simulation!

Enter the starting floor for the elevator (0-9, 0 = Ground Floor): 0
Enter the number of passengers (1-20): 2

Enter passenger details:

Passenger 1:
Starting floor (0-9, 0 = Ground Floor):
```

- d. **Destination Floor of passenger.** For each of the passengers, enter the destination floor.

```
Welcome to the Elevator Simulation!

Enter the starting floor for the elevator (0-9, 0 = Ground Floor): 0
Enter the number of passengers (1-20): 2

Enter passenger details:

Passenger 1:
Starting floor (0-9, 0 = Ground Floor): 0
Destination floor (0-9, 0 = Ground Floor): 9
Added passenger 1 waiting at floor 0 going to floor 9

Passenger 2:
Starting floor (0-9, 0 = Ground Floor): 3
Destination floor (0-9, 0 = Ground Floor): 8
```

10. Challenges and Solutions

Challenges Faced:

One of the main challenges faced during the project was deciding between using randomly generated passenger data for simulation or requiring user input for each scenario. Random generation could make the simulation more dynamic and less repetitive, but it would limit control over specific test cases and could lead to unpredictable results. On the other hand, allowing user input would offer more flexibility in controlling the simulation but could complicate the user experience with more steps for configuration.

Solutions Implemented:

To address this challenge, we used a user-driven approach for adding passengers, where users specify the starting and destination floors. Although random passenger generation isn't currently implemented, it could be added to allow for dynamic testing. For now, the system relies on user input, providing a controlled environment for specific test cases and scenarios.

11. Future Enhancements

Future improvements could include dynamic passenger priority, multiple elevators for better coordination, and a graphical user interface for real-time updates. Advanced algorithms for optimizing elevator paths, as well as the addition of maintenance mode and emergency scenarios, would enhance the simulation. More realistic passenger behavior, customizable floor-specific features, and dynamic capacity

management could be added. Finally, a speed control feature would allow users to adjust the simulation's pace for faster or more detailed testing.

12. Conclusion

Summary:

The elevator simulation project successfully models the operation of an elevator system, efficiently managing passenger movement between floors. By incorporating threads and mutexes, the system allows for smooth concurrent operations, simulating real-world scenarios such as capacity limits and direction changes. The project met its objectives, including providing real-time updates and ensuring the elevator operates efficiently, even under different passenger conditions. Through rigorous testing, the system demonstrated reliable performance and accuracy in simulating elevator movements, passenger boarding, and disembarking.

Lessons Learned:

This project reinforced the importance of thread synchronization and resource sharing using mutexes, which ensured that concurrent operations did not result in race conditions. It also highlighted the challenges of balancing user input flexibility with dynamic simulations, where both approaches can be valuable. The experience provided insight into how complex systems, such as elevator operations, can be modeled efficiently, and how thread-based simulations can handle real-time data in a controlled manner. The project also emphasized the significance of designing for scalability and future improvements, which could be implemented through dynamic passenger management and enhanced system features.

13. References

List all the resources, tools, and references used in the project (e.g., books, articles, websites).

<https://drive.google.com/file/d/1TVOpbgjx3DOi6Gz3O8ZLcyBtO8GjGMwJ/view?usp=sharing>

https://drive.google.com/file/d/1ZtldkylNVtxlH_j2zhqkMIJgO5EF15au/view?usp=sharing

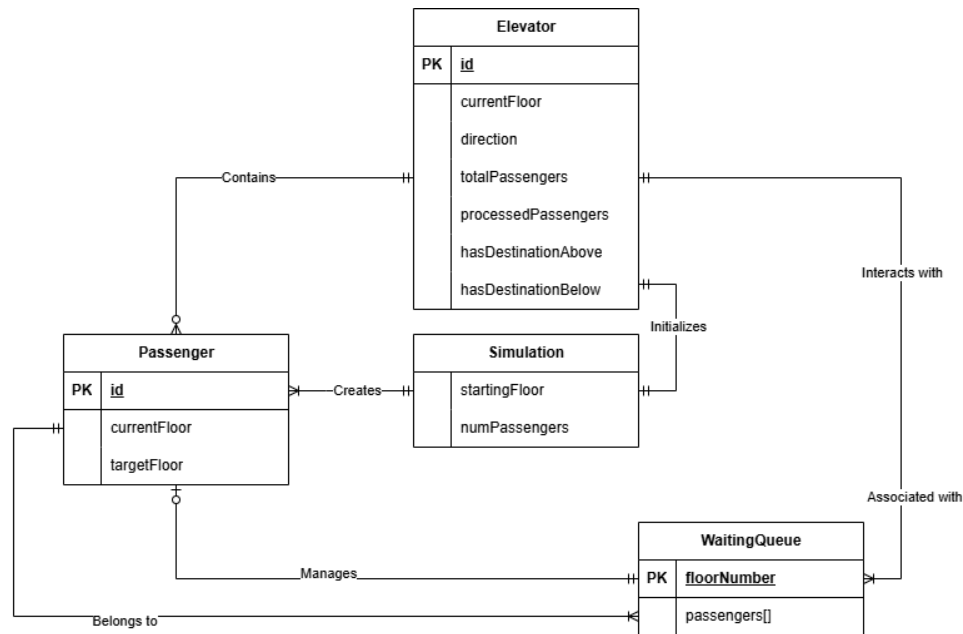
<https://github.com/rizjann/ElevatorProgramUsingThreads/blob/main/ElevatorSimulation.cpp>

draw.io

14. Appendices

Appendix A: Additional diagrams or charts.

Entity-Relationship Diagram



Appendix B: Complete Code

```
#include <iostream>
#include <thread>
#include <mutex>
#include <vector>
#include <queue>
#include <chrono>
#include <random>
#include <iomanip>
#include <condition_variable>
#include <limits>
```



```
using namespace std;
```

```
const int MAX_FLOORS = 10;    // Ground (0) to 9th floor  
const int MAX_CAPACITY = 9;   // Maximum people in elevator  
const int FLOOR_DELAY_SEC = 2; // Delay between floors
```

```
enum Direction { UP, DOWN, IDLE };
```

```
class Passenger {
```

```
public:
```

```
    int currentFloor;
```

```
    int targetFloor;
```

```
    int id;
```

```
    static int nextId;
```

```
    Passenger(int curr, int target) :
```

```
        currentFloor(curr),
```

```
        targetFloor(target),
```

```
        id(nextId++) {}
```

```
};
```

```
int Passenger::nextId = 1;
```

```
class Elevator {
```

```
private:
```

```
    vector<Passenger> passengers;
```

```
    queue<Passenger> waitingPassengers[MAX_FLOORS];
```

```
    int currentFloor;
```

```
    Direction direction;
```

```
    mutex mtx;
```

```
    condition_variable cv;
```

```

bool running;
int totalPassengers;
int processedPassengers;
bool hasDestinationAbove;
bool hasDestinationBelow;

void updateDestinationFlags() {
    hasDestinationAbove = false;
    hasDestinationBelow = false;

    // Check passengers in elevator
    for (const auto& p : passengers) {
        if (p.targetFloor > currentFloor) hasDestinationAbove = true;
        if (p.targetFloor < currentFloor) hasDestinationBelow = true;
    }

    // Check waiting passengers
    for (int i = 0; i < MAX_FLOORS; i++) {
        if (!waitingPassengers[i].empty()) {
            if (i > currentFloor) hasDestinationAbove = true;
            if (i < currentFloor) hasDestinationBelow = true;
        }
    }
}

Direction determineDirection() {
    updateDestinationFlags();

    if (direction == UP) {
        if (hasDestinationAbove) return UP;
        if (hasDestinationBelow) return DOWN;
    }
}

```

```

        return IDLE;
    }
    if (direction == DOWN) {
        if (hasDestinationBelow) return DOWN;
        if (hasDestinationAbove) return UP;
        return IDLE;
    }
    // If IDLE, choose based on waiting passengers
    if (hasDestinationBelow) return DOWN;
    if (hasDestinationAbove) return UP;
    return IDLE;
}

```

public:

```

Elevator(int startingFloor) :
    currentFloor(startingFloor),
    direction(IDLE),
    running(true),
    totalPassengers(0),
    processedPassengers(0),
    hasDestinationAbove(false),
    hasDestinationBelow(false) {}

```

```

void clearScreen() {
    cout << "\033[2J\033[1;1H";
}

```

```

void displayStatus() {
    clearScreen();
    cout << "=== Elevator Status ===" << endl;
}

```

```

        cout << "Current Floor: " << (currentFloor == 0 ? "Ground" : to_string(currentFloor)) <<
endl;
        cout << "Direction: " << (direction == UP ? "UP" : direction == DOWN ? "DOWN" :
"IDLE") << endl;
        cout << "Passengers in elevator: " << passengers.size() << "/" << MAX_CAPACITY <<
endl;
        cout << "Total passengers processed: " << processedPassengers << "/" << totalPassengers
<< endl;

```

```

        cout << "\nWaiting Passengers:" << endl;
        for (int i = MAX_FLOORS - 1; i >= 0; i--) {
            cout << "Floor " << (i == 0 ? "G" : to_string(i)) << ": ";
            if (!waitingPassengers[i].empty()) {
                cout << waitingPassengers[i].size() << " waiting";
            } else {
                cout << "None";
            }
            cout << endl;
        }
        cout << "=====" << endl;
    }

```

```

void addPassenger(int fromFloor, int toFloor) {
    lock_guard<mutex> lock(mtx);
    waitingPassengers[fromFloor].push(Passenger(fromFloor, toFloor));
    totalPassengers++;
    cout << "Added passenger " << Passenger::nextId - 1
        << " waiting at floor " << (fromFloor == 0 ? "G" : to_string(fromFloor))
        << " going to floor " << (toFloor == 0 ? "G" : to_string(toFloor)) << endl;
}

```

```

bool shouldStopAtFloor(int floor) {
    // Check if any passengers want to get off
    for (const auto& passenger : passengers) {
        if (passenger.targetFloor == floor) return true;
    }

    // Check if any waiting passengers can be picked up
    if (!waitingPassengers[floor].empty() && passengers.size() < MAX_CAPACITY) {
        Passenger front = waitingPassengers[floor].front();
        if ((direction == UP && front.targetFloor > floor) ||
            (direction == DOWN && front.targetFloor < floor) ||
            direction == IDLE) {
            return true;
        }
    }

    return false;
}

bool isSimulationComplete() {
    lock_guard<mutex> lock(mtx);
    return processedPassengers >= totalPassengers && passengers.empty();
}

void run() {
    while (running && !isSimulationComplete()) {
        unique_lock<mutex> lock(mtx);

        // Handle passengers getting off
        auto it = passengers.begin();
        while (it != passengers.end()) {

```

```

        if (it->targetFloor == currentFloor) {
            cout << "Passenger " << it->id << " getting off at floor "
                << (currentFloor == 0 ? "G" : to_string(currentFloor)) << endl;
            processedPassengers++;
            it = passengers.erase(it);
        } else {
            ++it;
        }
    }

    // Pick up waiting passengers
    while (!waitingPassengers[currentFloor].empty() && passengers.size() <
MAX_CAPACITY) {
        Passenger front = waitingPassengers[currentFloor].front();
        if ((direction == UP && front.targetFloor > currentFloor) ||
            (direction == DOWN && front.targetFloor < currentFloor) ||
            direction == IDLE) {
            passengers.push_back(front);
            waitingPassengers[currentFloor].pop();
            cout << "Passenger " << front.id << " boarding at floor "
                << (currentFloor == 0 ? "G" : to_string(currentFloor))
                << " going to floor "
                << (front.targetFloor == 0 ? "G" : to_string(front.targetFloor)) << endl;
        } else {
            break;
        }
    }

    // Update direction
    direction = determineDirection();
    displayStatus();

```

```

// Move elevator
if (direction != IDLE) {
    lock.unlock();
    this_thread::sleep_for(chrono::seconds(FLOOR_DELAY_SEC));
    lock.lock();

    if (direction == UP && currentFloor < MAX_FLOORS - 1) {
        currentFloor++;
    } else if (direction == DOWN && currentFloor > 0) {
        currentFloor--;
    }
} else {
    lock.unlock();
    this_thread::sleep_for(chrono::seconds(1));
    lock.lock();
}
}

void stop() {
    running = false;
    cv.notify_all();
}

};

int getValidFloor(const string& prompt) {
    int floor;
    while (true) {
        cout << prompt;
        if (cin >> floor && floor >= 0 && floor <= 9) {

```

```

        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        return floor;
    }
    cout << "Invalid floor! Please enter a number between 0 (Ground Floor) and 9.\n";
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}
}

```

```

int getValidPassengerCount() {
    int numPassengers;
    string input;
    while (true) {
        cout << "Enter the number of passengers (1-20): ";
        getline(cin, input);

        // Check if input is empty
        if (input.empty()) {
            cout << "Please enter a number between 1 and 20.\n";
            continue;
        }

        // Check if input contains only digits
        bool isValid = true;
        for (char c : input) {
            if (!isdigit(c)) {
                isValid = false;
                break;
            }
        }
    }
}

```



```

    if (!isValid) {
        cout << "Invalid input! Please enter a numerical value between 1 and 20.\n";
        continue;
    }

    // Convert string to integer
    try {
        numPassengers = stoi(input);
        if (numPassengers >= 1 && numPassengers <= 20) {
            return numPassengers;
        } else {
            cout << "Number must be between 1 and 20!\n";
        }
    } catch (const exception& e) {
        cout << "Invalid input! Please enter a numerical value between 1 and 20.\n";
    }
}

int main() {
    cout << "Welcome to the Elevator Simulation!\n\n";

    // Get starting floor
    int startingFloor = getValidFloor("Enter the starting floor for the elevator (0-9, 0 = Ground Floor): ");

    // Get number of passengers
    int numPassengers = getValidPassengerCount();

    // Create elevator with specified starting floor
    Elevator elevator(startingFloor);

```

```

        // Get passenger details
cout << "\nEnter passenger details:\n";
for (int i = 0; i < numPassengers; i++) {
    cout << "\nPassenger " << i + 1 << ":\n";
    int fromFloor = getValidFloor("Starting floor (0-9, 0 = Ground Floor): ");
    int toFloor;
    do {
        toFloor = getValidFloor("Destination floor (0-9, 0 = Ground Floor): ");
        if (toFloor == fromFloor) {
            cout << "Destination floor must be different from starting floor!\n";
        }
    } while (toFloor == fromFloor);

    elevator.addPassenger(fromFloor, toFloor);
}

cout << "\nStarting elevator simulation...\n";
this_thread::sleep_for(chrono::seconds(2));

// Create thread for elevator operation
thread elevatorThread(&Elevator::run, &elevator);

// Wait for elevator thread to complete
elevatorThread.join();

cout << "\nSimulation complete! All passengers have reached their destinations.\n";

return 0;
}

```

Appendix C: Any other relevant material.

<https://drive.google.com/file/d/1TVOpbgjx3DOi6Gz3O8ZLcyBtO8GjGMwJ/view?usp=sharing>

https://drive.google.com/file/d/1ZtIdkylNVtxlH_j2zhqkMIJgO5EF15au/view?usp=sharing

<https://github.com/rizjann/ElevatorProgramUsingThreads/blob/main/ElevatorSimulation.cpp>