HOMEWORK 4

1. **Objective**: We’re building a elevator to carry people from floor to floor. A person will push the hallway button, the elevator goes up to that floor, opens the door, take that person in, the person input his destination floor, carry him to his destination floor, opens the door and let him get out.

During this process, if anyone else push the hallway button, the elevator will also take in that order. In one small trip, the elevator car only goes up or down, until there’s no order to goes up or down further. The elevator will pick up anyone who ordered the elevator along the way, regardless of whether they want to go up or down (This is not how a normal elevator works, and we will explain why we didn’t implement like this later on). In one big trip, the elevator will take on multiple small trips until the queue of passengers requested is empty.

Then it will give the report of the activity of the elevators and profiles of each passengers.

1. **Design Process**:

1. After HW3, we found it really hard to incorporate what we had into HW4, so we restart.

2. Since there’re so many aspects to this problem, we wanted to divide and conquer:

+ First, we would build the foundational structures of **Inherent** information: Queue implementation, Clock, Passengers, and Elevator car with their inherent information like ID, originFloor,…

+ Then we tackle **Operational** variables and functions. We thought we should make it run first, this did work very well.

+ Then we would incorporate the **Time** counting element, including TimeRequested, WaitTime, TravelTime….

=> This did not work, even though we thought we could consider each task a module and finish it one by one, we could not build a structure intuitive enough to include all the time counting of each variables.

3. We redo the **Operational** and **Time** functions and variables, having learned that we could not separate them. The rest of objects works fine as they were. We did made the choice to change one data structure (Which we will explain in a bit)

**\* Process**:

**Step 1**: We started by building in Inherent information onto these classes, starting with Passengers, in this order:

- Passengers: PID, originFloor, destFloor, Served

- Elevator Car: CarID, CurrentFloor

- Motor & HallwayButton: (This is pretty isolated from everything) I created a struct with all the variables for easy access

- Clock: We created this based on the need of the program, exchanging continuously from using only seconds (for waitTime, travelTime) to using actual clock display (TimeRequested, Timer). Since this is also pretty isolated, we also coded this all the way and added in extra functions afterwards.

**c) Step 2: Object Oriented Design: Rationale for using Data Structure**

1. We decided that by the nature of this problem:

- **ServiceRequestQueues**: The elevator can only goes up( or down) one floor at a time, the Request structure would be perfect if we use **Priority Queue**. We can use 2 different Queues for both directions up and down, ***ServiceQueueUp*** and ***ServiceQueueDown***, with the condition that they’re sorted in opposite directions.

+ We thought of using **Heap** to code this Queue, but we thought it was just unnecessary considering a linear linked list work just as well. Also, even though our hypothetical building can have 10000 floors, realistically, it won’t have that many, so you won’t have to worry about searching or sorting with efficiency. 2 singly linear **Linked Lists** will suffice.

+ We also consider using only **one priority queue**. However, for example if the car just passed through floor 3, and another person want to go up from floor 2, we would not want floor 2 to be at the front of our current cue. As the elevator is going up, it can’t go back to 2.

+ We used the same structure for ***QueueofPassengerRequested***. All passengers who ordered the car, will be pass into this queue. This is not really the best structure for it, since the original design include enqueue(push\_back) and dequeue(pop\_front). But P1 could go from floor F1->F5 (so P1 would be at the front of this queue), but the car picks up P2 going from floor F2->F3, so P2 would be added in later, but removed from the queue first. Therefore, we chose to use Queue (Linked List) only because it’s the easiest to append and delete at will. (Array is always harder, and STL vector proved to be more troublesome later on)

+ We also use the same structure for ***ServiceQueueCurrent***, signifying who’s in the elevator right now. There’s no better alternative.

- ***Listofpassengerserved*** and List of Passenger to be pass into the function of the of the Car controller: We use **STL Vector** for this, we thought it would be easier to manipulate (append and dynamically retrieve elements, and be able to be passed through a function). However, we couldn’t figure out a fast way to delete the vector, and we added in more and more functionality to our linked list. The benefit of the Queue(Linked List) soon outweigh the cost of the vector. We regret picking vector for this, but since we’re too deep into the project, we couldn’t change something so foundational and deeply rooted in our program’s logic.

**Step 3: How it Operates**:

- When an unserved passenger press the hallway button, regardless of the person’s wish to go which direction, the car will stop to pick the passenger up. Since if we do it like an actual elevator, not picking the person up till the car’s and the person’s direction is the same, we will need another queue for next trip, (So 1 current QueueUp and 1 QueueUp after the trip, 1 current QueueDown and 1 QueueDown after the trip). We would need infinite queues, or alternating 2 QueueUps and 2 QueueDowns. We figure it was too much cost for a simple function, so we decide to just pick up whoever is on such floor.

- At first, the elevator will decide if it wants to go up or down.

- Then It will travel, one by one, to the next floor on the respective Queue.

- We constantly add 2 seconds onto the timer clock, so any order (both Hallway and Controller) added in that time will be added into the queue.

- Acceleration takes 2 seconds, Moving in constant speed takes 4 seconds, Deceleration takes 2 seconds. During which time, the Car software continuously interacts with the motor:

+ It takes 4 seconds to go to another floor.

+ Acceleration or Constant Motion 2 (See code) will add 1 to current floor. By now, it will check if there’s anyone ordered this floor (Both Hallway and Controller). If yes, it will decelerate and stop to drop off. If not, it will Constant Motion 1 to the next floor. (Each constant motion takes 2 seconds, both in total takes 4 seconds)

+ Stops, Opens Door, and the person gets out. This takes no time. We used to consider this action to take 2 seconds. However, it’s too complicated due to ***OpensDoor()*** being nested everywhere, which bloated up the personal WaitTime and TravelTime. We thought this was too much redesign for the benefit, so we skipped it. In real life we would need to take this into account.

- Car will move up (or down) till ***ServiceQueueUp*** (or ***ServiceQueueDown***) is empty. Then it changes directions, and move the other way. It won’t stop until ***ServiceQueueRequested*** is empty.

Step 4: How it Counts Time:

- Since every action of the car takes 2 seconds, we added 2 seconds every time a car moves half floor, then loop it in ***ElevatorSmallTrip()***

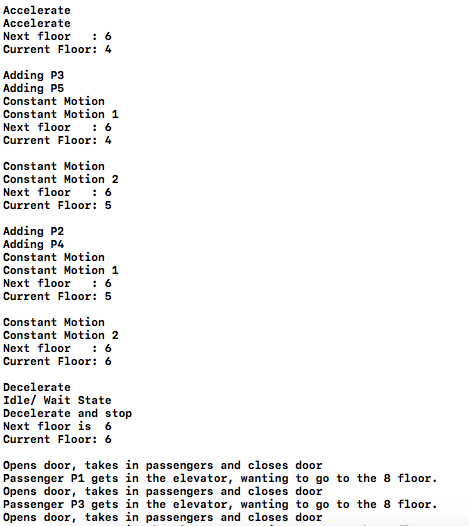
- Also at the same time, we wrote a function to add waitTime and travelTime every time the Timer add 2 seconds for every person in the requested queue.

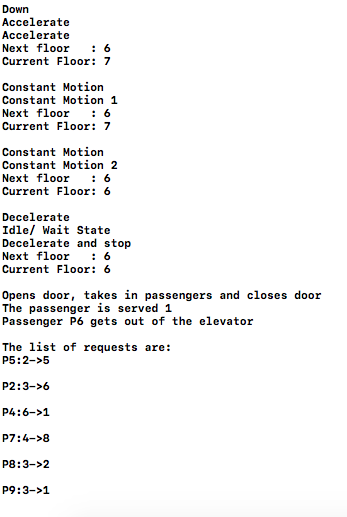
**Step 5: Testing Samples**:

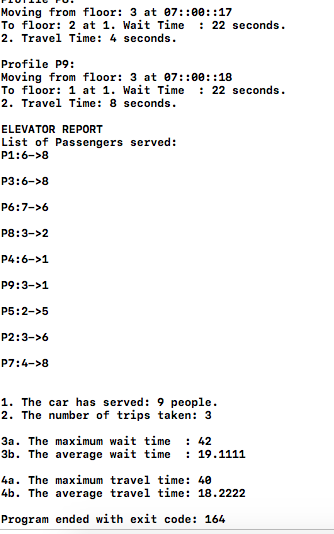
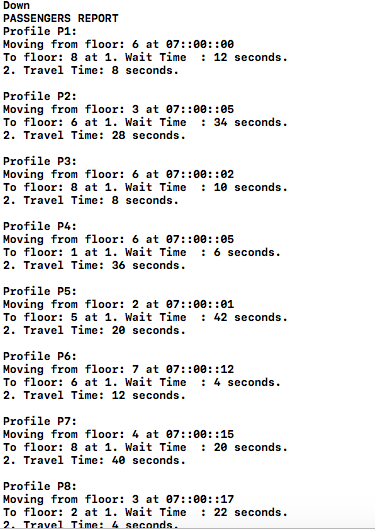
- I created 3 vectors of passengers will all their inherent information. All the time requested are within 60s, to simulate the complexity of orders by actual passengers.

- You can try 3 vectors into C1. You can change the values of floors and TimeRequested. The car will stop once it finishes its requests order, so if you put the time to far apart it will stop before taking in another element in the vector

- Snippets of Code Results:







**Step 6: How it Reports Analysis**

- We use simple variables like MaxWaitTime, MaxTravelTime,… (For elevator) and WaitTime, TravelTime (For people) to increment over the course of the program. We would be able to use the data in the end

Results:

**Vector1:**

**PASSENGERS REPORT**

**Profile P1:**

**Moving from floor: 6 at 07::00::00**

**To floor: 8 at 1. Wait Time : 12 seconds.**

**2. Travel Time: 8 seconds.**

**Profile P2:**

**Moving from floor: 3 at 07::00::05**

**To floor: 6 at 1. Wait Time : 34 seconds.**

**2. Travel Time: 28 seconds.**

**Profile P3:**

**Moving from floor: 6 at 07::00::02**

**To floor: 8 at 1. Wait Time : 10 seconds.**

**2. Travel Time: 8 seconds.**

**Profile P4:**

**Moving from floor: 6 at 07::00::05**

**To floor: 1 at 1. Wait Time : 6 seconds.**

**2. Travel Time: 36 seconds.**

**Profile P5:**

**Moving from floor: 2 at 07::00::01**

**To floor: 5 at 1. Wait Time : 42 seconds.**

**2. Travel Time: 20 seconds.**

**Profile P6:**

**Moving from floor: 7 at 07::00::12**

**To floor: 6 at 1. Wait Time : 4 seconds.**

**2. Travel Time: 12 seconds.**

**Profile P7:**

**Moving from floor: 4 at 07::00::15**

**To floor: 8 at 1. Wait Time : 20 seconds.**

**2. Travel Time: 40 seconds.**

**Profile P8:**

**Moving from floor: 3 at 07::00::17**

**To floor: 2 at 1. Wait Time : 22 seconds.**

**2. Travel Time: 4 seconds.**

**Profile P9:**

**Moving from floor: 3 at 07::00::18**

**To floor: 1 at 1. Wait Time : 22 seconds.**

**2. Travel Time: 8 seconds.**

**ELEVATOR REPORT**

**List of Passengers served:**

**P1:6->8**

**P3:6->8**

**P6:7->6**

**P8:3->2**

**P4:6->1**

**P9:3->1**

**P5:2->5**

**P2:3->6**

**P7:4->8**

**1. The car has served: 9 people.**

**2. The number of trips taken: 3**

**3a. The maximum wait time : 42**

**3b. The average wait time : 19.1111**

**4a. The maximum travel time: 40**

**4b. The average travel time: 18.2222**

**Program ended with exit code: 164**

**Vector2**

**PASSENGERS REPORT**

**Profile P10:**

**Moving from floor: 2 at 08::06::00**

**To floor: 12 at 1. Wait Time : 4 seconds.**

**2. Travel Time: 40 seconds.**

**ELEVATOR REPORT**

**List of Passengers served:**

**P10:2->12**

**1. The car has served: 1 people.**

**2. The number of trips taken: 2**

**3a. The maximum wait time : 4**

**3b. The average wait time : 4**

**4a. The maximum travel time: 40**

**4b. The average travel time: 40**

**Vector3**

**PASSENGERS REPORT**

**Profile P11:**

**Moving from floor: 1 at 09::10::05**

**To floor: 2 at**

**1. Wait Time : 8 seconds.**

**2. Travel Time: 4 seconds.**

**Profile P12:**

**Moving from floor: 5 at 09::10::07**

**To floor: 8 at**

**09::10::41**

**1. Wait Time : 22 seconds.**

**2. Travel Time: 12 seconds.**

**ELEVATOR REPORT**

**List of Passengers served:**

**P11:1->2**

**P12:5->8**

**1. The car has served: 2 people.**

**2. The number of trips taken: 2**

**3a. The maximum wait time : 22**

**3b. The average wait time : 15**

**4a. The maximum travel time: 12**

**4b. The average travel time: 8**