# Assumptions:

I made the assumption that it would be ideal to make the code scalable. As such, I allowed for one instance of the program to handle any number of buildings, and any number of elevators, with up to 999 floors a building. That is only a soft limitation though, and it can easily be updated if needed.

Another assumption I made was that it would be a good idea to have each elevator have its own pseudo-queue, one that only takes from the front but could add anywhere. It was not a good idea. It added unnecessary complexity and limitations on what I could achieve. If I had more time I would redo the project using a different method.

# UML

Class: Building  
- int totalFloors  
+ vector<Elevator> elevatorList  
+ void generateInit()  
+ void tick()  
+ addPress(buttonPress init)  
+ addElevator()  
+ print()

Class: Elevator  
-callQueue queue  
-bool goingUpwards  
-int currentFloor  
+void move()  
+void stop(int totalFloors)  
+buttonPress generateEcho(int totalFloors)  
+int determineDistance(buttonPress targetFloor)  
+bool hasArrived()  
+int peekTargetFloor()  
+int getCurrentFloor()   
+void updateWaitTime()  
+void addPress(buttonPress x)   
+void print2()

Class: callQueue  
-buttonPress\* head  
-buttonPress\* tail  
-int queueSize  
+int determineDistance(buttonPress targetFloor, int elevatorCurrentFloor)  
+void addNewPress(buttonPress temp, int elevatorCurrentFloor)  
+void updateWaitTime(int elevatorCurrentFloor)  
+void pop();  
+void print2();  
+int peekFloor() { if (!isEmpty())  
+int peekFloorsBeforeService()   
+int peekStopsBeforeService()   
+int size()   
+bool peekDirection()   
+bool isEmpty()  
+bool isInit()

Struct: buttonPress  
+int floor  
+bool goingUp  
+bool isInit = false  
+int floorsBeforeService  
+int stopsBeforeService  
+buttonPress\* next  
+buttonPress\* prev

# Efficiency

The most intensive functions are in callQueue, however those are still only singular-depth while loops. The big o complexity is O(n) on average. The best case is O(1)—typically when the list is empty.

It certainly can be more efficient in places, but overall it is rather quick because it lacks nested loops.

# Contributions

I worked by myself on this project.

# References

The only reference I used on this project was for a single line of implementation. I learned how to pause the system for a set amount of real-life seconds using an old stack overflow post.

https://stackoverflow.com/questions/158585/how-do-you-add-a-timed-delay-to-a-c-program