Honors Project #1

Assignment Overview
This is the first of two projects needed to get Honors credit. This one is due before midnight on March 31st, 2017.

Overview
For the Honors project, you have to implement an elevator simulator. An elevator simulator simulates the activity of a working elevator where people in building request an elevator to pick them up and deliver them to a different floor. It is a challenge to be efficient about the activities of your elevator, where "efficient" might mean different things, for example: total distance traveled by the elevator, average wait time per person served, etc. Part 1 will be to develop the simulator as outlined below. Part 2 will be to develop an optimal strategy (to be specified in the second specification document).

Project Information
We are going to read from redirected standard cin some information that will start our simulation. The file data.txt contains two lines: one set of parameters for the single_run and one set of parameters for the multi_run (see below). Each line will have the following information as space separated values in this order:

- The speed of the elevator (timeUnit/floor) (double)
- Time it takes for a single passenger to load/unload (timeUnit/passenger) (double)
- maximum number of floors in the building (floors range from 1 to this number) (long)
- the total number of passengers that will be in this simulation (long)
- the number of passenger loads to generate (1 for the first line, something bigger for the second).
- random number seed (long)

Your overall goal is to generate a sample passenger load for the simulation. Each passenger will consist of three values:
- the time they arrive to be picked up (when they pushed the elevator button)
- the floor they are presently on
- the floor that they are going to

We provide some support functions for generating passenger loads from the data you read in (see Support below)

Tracking Time
You are going to simulate the passing of time with your simulation. This is the fundamental principle of nearly any simulation. You are going to keep a "clock" that represents the present time in the simulation, beginning at the value 0 when the
Simulation starts. You are going to update that clock based on the events that take place. For example:

- every time the elevator moves to new floor some number of time units pass based on the data.txt value provided (floors * time/floor)
- each time you load or unload a passenger some fraction of time passes based on the data.txt value provided (passengers * time/passenger)
- the passenger load is generated in a time order. You can process (or at least consider processing) any passenger whose arrival time is less than or equal to the current clock value. However, you cannot (and should not) look forward into the list of passengers that have not arrived yet! In this way, your elevator simulates a real-time load.

**Assumption**

For this set of simulations, we are going to make the following (unreasonable) assumption. We are not going to set a capacity, a number of passengers that the elevator can hold. That is, we will allow the elevator to hold any number of passengers. This will be changed for the second part of the project, so be aware!

**Support**

I have provided the files random_support.h, random_support.cpp and main-skeleton.cpp as a way to generate passenger loads and to work more easily with random numbers.

- random_long_in_range(int lo, int hi, mt19937_64 & random_engine)
  
  It will generate a random long in the range of the provided parameters but requires a seeded random engine be passed from the calling program.

- deque<deque<long>> gen_passengers(long floor_max, long passenger_max, mt19937_64 &reng)
  
  o this will generate a passenger load for your simulation. It also requires a seeded random engine
  o what's a deque? It is basically a vector but you can push_front as well as push_back, you can pop_front as well as pop_back. Being able to work from the front as well as the back might be helpful (hint hint)
  o The return value is a 2D deque where each row/deque is a single passenger with three values: arrival time, start floor, destination floor in that order. Thus the entire return value are multiple rows of passengers that must be dealt with.
  o We guarantee that the order of rows/deque in the 2D deque are in time order. That is the arrival time of the first passenger (the first row) is less than or equal to the second, which is less than or equal to the third etc. The time values are random but they are in order!

- string print_2d_deque (const deque<deque<long>> &d)
  
  o prints the contents of the 2d deque so you can see the load easily.
• **main** program shows you the basics on reading in the data.txt values and running the provided functions.
  o Note that you seed the random number generator exactly once!
  o You never, ever pass a random engine except by reference, otherwise it resets the seed (and the sequence will repeat)
  o look at the output of the given passenger load to see what it looks like.

**Simulations**
You are going to create **two strategies** to simulate an elevator that processes passengers.

1. **RandomElevator:** This strategy utilizes random numbers for its processing. We have to establish some priorities so that we can process the passengers algorithmically. We process as indicated below until every passenger in the simulation has been delivered to their floor. Here is the random strategy:

   **First priority:** If there are passengers on the elevator:
   a. The strategy should select which floor to visit to drop passengers off in random order.
      i. That is, from the list of passengers in the elevator and the destinations that they all have, you should pick one passenger at random and deliver that passenger to their destination floor.
      ii. move to that floor
      iii. Update the current time.
   b. Once it arrives at a destination floor
      i. allow all passengers who have that floor as a destination to exit (at least one, perhaps more). Update the current time.
      ii. if there are also passengers waiting at that floor to board the elevator (given the current time), load all those passengers into the elevator.
      iii. Update the current time.

   **Second priority:** If at some point there are no more passengers on the elevator
   a. Select a destination floor randomly from the list of all floors where there is at least one passenger waiting **at the current time** (or earlier) from your passenger load.
      a. Move to that floor
      b. Update the current time
   b. After going to that floor, the elevator should load all the passengers that are waiting on that floor.
      a. Update the time for passengers loaded.

   **Third priority:**
   a. If at the current time there are neither any passengers in the elevator nor any passengers waiting from your passenger load **at the current time**, stay at the current floor and let time pass until a passenger is waiting at some future time.
      i. if we are at the end of the simulation (we have processed every passenger in the file), then quit and report.
2. **Strategy Elevator**: Again we must establish priorities, but in this case try to be a bit smarter

**First Priority**: If there are passengers in the elevator:
   a. Select the destination **closest** to the present elevator position:
      i. That is, for all the destinations of passengers presently on the
elevator, select the destination floor closest to the present elevator
position and move there.
      ii. If multiple passengers are closest, choose randomly among the close
floors.
   b. Move to that floor
      i. Update the current time.
   c. Allow all passengers who have that floor as a destination to exit (at least one,
perhaps more).
      i. if there are also passengers waiting at that floor to board the elevator
(given the current time), load all those passengers into the elevator.
      ii. Update the current time for moving all passengers

**Second priority**: If at some point there are no more passengers on the elevator
   a. Pick a floor with the most passengers waiting at the current time.
      i. If multiple floors have the same “most” passengers waiting, choose
the closest floor among them.
      ii. If multiple floors are closest, choose arbitrarily among them.
   b. Move to that floor
      i. Update the current time.

**Third priority**: 
   a. If at the current time there are neither any passengers in the elevator nor any
passengers waiting on any floor, stay at the current floor and let time pass
until a passenger is waiting at some future time.
   ii. if we are at the end of the simulation (we have processed every
passenger in the file), then quit and report.

**More detail**
Two kinds of simulations are going to be run, a single-run using the two simulators and a
multi-run of simulations with the two simulators. This all happens from the single startup
of your game.

**Single-Run**:
- your program will generate a passenger load based on the values provided by
the first line of data.txt

- your program will then simulate both the RandomElevator and StrategyElevator on the same passenger load and report the output of both runs to a file exactly named "single_run.txt" to be created in the directory of your program.
- RandomElevator goes first, StrategyElevator second
- Provide as much detail as you can imagine to make clear what happened for each event during the simulation and why. We will keep this passenger load small so we can see what's going on.
- you will be evaluated upon the detail you provide, for example:
  - source floor
  - destination floor
  - number of passengers onboard during movement
  - load time of passengers at source floor
  - unload time of passengers at destination floor
  - time when this movement is done, i.e. movement ends when passengers arriving at their destination floor have exited (before new passengers, if any, at that floor begin loading).
  - total time for this movement (time from start of movement at source floor (before loading passengers, if any, at source floor) to time when passengers, if any, have unloaded at destination floor).

**Multi-Run**

- Following single-run, you will use the number of simulations to run of RandomElevator vs. StrategyElevator and report statistics of the results to a file called "multiple_run.txt".
- You will generate multiple passenger loads using the parameters read in from line two of data.txt.
  - because we are using a random engine, each load will be different.
    Check it yourself and see.
- For each load, you will run the load with one strategy then the other (same load, different strategy).
- You will do this for as many loads as indicated by the file (potentially 1000's)
- Across all the loads we are looking to record simulation statistics to compare the two strategies:
  - Not the actual games, just statistics.
  - The goal is to compare the strategies over a lot of random runs and see how they did in comparison to each others
- The statistics to include for each strategy are:
  - average of total time it took the elevator to deliver all passengers
  - average wait time for all the passengers (note that this is not the average of averages). Wait time for a passenger is defined to start when a passenger arrives at the elevator and end when they exit the elevator at their destination.
  - maximum and minimum total time across all runs
  - maximum and minimum passenger wait time across all runs
  - anything else you think might be a good idea
Deliverables
You turn in exactly one file. If you used my support stuff I'll compile against that myself, but you turn in only 1 file. Makes it easier for me.

The deliverable for this assignment is the following file:

    honors1.cpp – the source code for your program

Be sure to use the specified file name and to submit it for grading via the handin system before the project deadline. Do not use multiple files, even though that is a good idea. It must be in one file.

Turn it in to handin as always. This will be project 12 in the handin system.