COMSC-200 Lab 14
Using Templates

GOOD PROGRAMMING PRACTICES *show / hide*

ABOUT THIS ASSIGNMENT
In this lab session, you continue the **ElevatorSimulation** problem -- a problem that will be solved over labs 8-10 and 14-16. While there are template-based exercises in our Deitel textbook, and it is recommended that you try them out for yourself, you will not be submitting any of them.

As you complete this assignment, post the required files to the COMSC server. You may post them all at once, or one-at-a-time as you complete them, as you prefer. The individual labs are graded in order, starting with "a", which must be fully correct before "b" is graded, and so on. This assignment is worth 50-points, to be awarded after all labs are successfully completed. Use the "Submit your SP2015 work" link on the class website to post your file for this lab, using the **lab14** folder provided.

Lab 14 More STL vectors[`Floor.h` and `Floor.cpp`]
Expand the definition of the `Floor` class in two new versions of **Floor.h** and **Floor.cpp**. The constructor from the early version is modified to add a 2nd, optional parameter; the other members remain. These new floors will use vectors to track those riders waiting for an up-elevator, and those waiting for a down-elevator. Riders who exit elevators are *not* tracked once they leave the elevator (even though `Elevator::removeRidersForDestinationFloor()` does return a vector).

Do *not* use friend relationships, except for overloaded stream insertion operators.

Use the following in the class definition. Note that the H file will require the compiler directives, declarations, and includes, which you must determine.

```
class Floor
{
  const string NAME; // name of floor, for example: Mezzanine
  const int location; // inches above ground level
  vector<Rider> upRiders; // affected by addNewRider, removeUpRiders,...
  vector<Rider> downRiders; // ...and removeDownRiders functions

  public:
  Floor(const int, const char* const = "unnamed"); // name and location (inches above ground) of floor

  // inline functions
  int getLocation() const;
  bool hasRidersWaiting() const;
  bool hasUpRiders() const;
  bool hasDownRiders() const;
  int getUpRiderCount() const;
  int getDownRiderCount() const;
  string getName() const;

  // non-inline functions
  bool isPreferredDirectionUp() const; // based on Rider with smallest ID
  void addNewRider(const Rider&); // add to up- or down-vector
  vector<Rider> removeUpRiders(int); // int is max #of riders...
  vector<Rider> removeDownRiders(int); //...to move onto elevator

  // just in case you prefer to use "if (...== *toFloor)"
  //  in Elevator::removeRidersForDestinationFloor(), instead of "...== toFloor)"
  bool operator==(const Floor& floor) const {return NAME == floor.NAME;}

  friend ostream& operator<<(ostream&, const Floor&); // say name, location, #'s of up/down riders waiting
};
```

Write the inline functions in the H file -- do NOT use the `inline` keyword;

Write the non-inline functions in the CPP file. Advice for writing these functions will be presented in lecture.

The Floor's stream insertion operator should output *all on one line* to the console screen, so that results for larger simulations will be easier to read. Use abbreviations as necessary in order to accomplish this.

HINT: Remember the rules for forward declarations! You'll need them from this point and going forward, because now, Floors know about Riders, and Riders know about Floors!

*Post **Floor.h** and **Floor.cpp** to the COMSC server for credit.* Do *not* submit the **ElevatorSimulation4.cpp** file or any other file with `int main()` in it. The ElevatorSimulation series is the subject of six (6) lab assignments, including the last one. *If you have to make corrections to already-submitted H* *or CPP files, do so into the folder for this lab -- do NOT overwrite your already-graded files from previous labs.*

To test your code, compile, build, and run with all previous ElevatorSimulation CPPs, and the following **ElevatorSimulation4.cpp**. Here's what it should do: (1) take 2 riders from the 1st floor to the 2nd floor, then (2) take 5 riders from the 2nd floor to the basement, then (3) take 3 riders from the basement to the 2nd floor, and (4) end with 4 riders stranded on the 1st floor. All this should be a 5 inches per second movement of the elevator, without sudden jumps that might injure or kill its passengers:

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

#include <vector>
using std::vector;

#include "Floor.h"
#include "Rider.h"
#include "Elevator.h"

int main()
{
  // create three floors
  Floor firstFloor(0, "First Floor");
  Floor secondFloor(120, "Second Floor");
  Floor basement(-120, "Basement");

  // add people to the floors (who await elevators)
  int i, count = 0;
  for (i = 0; i < 2; i++) // 2 going from 1st floor to 2nd
    firstFloor.addNewRider(Rider(secondFloor));
  for (i = 0; i < 4; i++) // 4 going from 1st floor to basement
    firstFloor.addNewRider(Rider(basement));
  for (i = 0; i < 3; i++) // 3 going from basement to 2nd floor
    basement.addNewRider(Rider(secondFloor));
  for (i = 0; i < 5; i++) // 5 going from 2nd floor to basement
    secondFloor.addNewRider(Rider(basement));
  cout << firstFloor << endl;
  cout << "The first floor: " << firstFloor << endl;
  cout << secondFloor << endl;
  cout << "The second floor: " << secondFloor << endl;
  cout << basement << endl;
  cout << "The basement: " << basement << endl;

  // check the getUpRiderCount() and getDownRiderCount() functions
  if (firstFloor.getUpRiderCount() != 2) cout << "ERROR IN " << firstFloor.getName() << "'s up-rider count\n";
  if (firstFloor.getDownRiderCount() != 4) cout << "ERROR IN " << firstFloor.getName() << "'s down-rider count\n";
  if (basement.getUpRiderCount() != 3) cout << "ERROR IN " << basement.getName() << "'s up-rider count\n";
  if (secondFloor.getDownRiderCount() != 5) cout << "ERROR IN " << secondFloor.getName() << "'s down-rider count\n";

  // create an elevator (capacity 12, speed 5 in/sec)
  Elevator e(12, 5, firstFloor);
  cout << "The elevator: " << e << endl;
  cout << "Memory address of only elevator in this test: " << &e << endl;
  // set elevator to move up; load up-riders
  int n = e.getAvailableSpace();
  vector<Rider> riders = firstFloor.removeUpRiders(n);
  e.addRiders(riders);
  cout << "Elevator should have 2 riders: " << e << endl;
  e.setDestinationBasedOnRiders(); // this should find the 2nd floor
  cout << "Asked riders for destination: " << e << endl;
  e.setDirectionUp();
  cout << "Set direction up: " << e << endl;
  e.closeDoor();
  cout << "Closed door: " << e << endl;

  while(!e.isNearDestination())
  {
    if (e.isDirectionUp()) e.moveUp();
    else if (e.isDirectionDown()) e.moveDown();
    cout << e << endl;
    if (!(++count % 20)) {cout << "Press ENTER to continue...\n"; cin.get();}
  }
  e.moveToDestinationFloor();
  cout << "Moved to 2nd floor: " << e << endl;

  // remove riders for 2nd floor, add waiting riders, and go to basement
  riders = e.removeRidersForDestinationFloor(); // ignore returned vector
  n = e.getAvailableSpace();
  riders = secondFloor.removeDownRiders(n);
  e.addRiders(riders);
  cout << "Elevator should have 5 riders: " << e << endl;
  cout << "Moving to basement:\n";
  e.setDestinationBasedOnRiders(); // this should find the basement
  e.setDirectionDown();
  e.closeDoor();
  cout << e << endl;
  while(!e.isNearDestination())
  {
    if (e.isDirectionUp()) e.moveUp();
    else if (e.isDirectionDown()) e.moveDown();
    cout << e << endl;
    if (!(++count % 20)) {cout << "Press ENTER to continue...\n"; cin.get();}
  }
  e.moveToDestinationFloor();
  cout << "Moved to basement (without stopping for 1st floor down-riders): " << e << endl;

  // remove riders for basement, add waiting riders, and go to 2nd floor
  riders = e.removeRidersForDestinationFloor(); // ignore returned vector
  n = e.getAvailableSpace();
  riders = basement.removeUpRiders(n);
  e.addRiders(riders);
  cout << "Elevator should have 3 riders: " << e << endl;
  cout << "Moving to 2nd floor:\n";
  e.setDestinationBasedOnRiders(); // this should find the 2nd floor
  e.setDirectionUp();
  e.closeDoor();
  cout << e << endl;
  while(!e.isNearDestination())
  {
    if (e.isDirectionUp()) e.moveUp();
    else if (e.isDirectionDown()) e.moveDown();
    cout << e << endl;
    if (!(++count % 20)) {cout << "Press ENTER to continue...\n"; cin.get();}
  }
  e.moveToDestinationFloor();
  cout << "Moved to 2nd floor: " << e << endl;

  // done
  e.openDoor();
  riders = e.removeRidersForDestinationFloor();
  cout << "Opened door, let out " << riders.size() << " riders: " << e << endl;
  e.setIdle();

  // done
  e.openDoor();
  e.setIdle();
  cout << "Test end:\n" << e << endl;
  cout << firstFloor << " (should be 4 stranded riders) " << endl
    << secondFloor << endl
    << basement << endl;

  // check the getUpRiderCount() and getDownRiderCount() functions
  if (firstFloor.getDownRiderCount() != 4) cout << "ERROR IN " << firstFloor.getName() << "'s down-rider count\n";

  // additional testing of your own, if you have any
  ...
}
```

Command Line Compiling With Visual C++:
Using the Rider and Elevator classes from lab 10, here is the command for compiling this project in Visual C++ (assuming that main is in a file named **ElevatorSimulation4.cpp**):

```
cl ElevatorSimulation4.cpp Rider.cpp Floor.cpp Elevator.cpp -EHs
```

This creates an executable named **ElevatorSimulation4.exe**. To run, type the command **ElevatorSimulation414**.

Alternatively, to compile each code module separately:

```
cl ElevatorSimulation4.cpp -c -EHs
cl Rider.cpp -c -EHs
cl Floor.cpp -c -EHs
cl Elevator.cpp -c -EHs
cl ElevatorSimulation4.obj Rider.obj Floor.obj Elevator.obj
```

Command Line Compiling With g++:
Here is the command for compiling this project in g++ (assuming that main is in a file named **ElevatorSimulation4.cpp**):

```
g++ ElevatorSimulation4.cpp Rider.cpp Floor.cpp Elevator.cpp
```

This creates an executable named **a.exe** on a PC or **a.out** on a Mac. To run, type the command **a** on a PC or **./a.out** on a Mac.

Alternatively, to compile each code module separately:

```
g++ ElevatorSimulation4.cpp -c ElevatorSimulation4.o
g++ Rider.cpp -c Rider.o
g++ Floor.cpp -c Floor.o
g++ Elevator.cpp -c Elevator.o
g++ ElevatorSimulation4.o Rider.o Floor.o Elevator.o
```

Elevator Simulation Index: where to find stuff...

class Rider, lab 8	class Elevator, lab 9	class Floor, lab 14
Rider::operator=, lecture 9	Elevator EXEs, lecture 9	Floor::addNewRider, lecture 14
Elevator::isNearDestination, lab 9	Elevator::moveToDestinationFloor, lab 9	Floor::isPreferredDirectionUp, lecture 14
class Building, lab 15	Elevator::addRiders, lecture 10	Floor::removeDownRiders, lecture 14
getDifferentInts(int, int&, int&), lab 14	Elevator::removeRidersForDestinationFloor, lecture 10	Floor::removeUpRiders, lecture 14
poissonHits(double), lab 15	Elevator::setDestinationBasedOnRiders, lecture 10	GUI simulation option, lab 15

How to pause a console program: *show / hide*

GRADING POLICY *show/hide*
