COMSC-200 Lab 10
Using The `vector` Class

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. You will also practice programming with classes by doing exercises from our textbook.

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 **lab10** folder provided.

Lab 10a [`Date.h` and `Date.cpp`]
Write the class as presented in figures 11.9 and 11.10 of our Deitel textbook, with modifications per the universal requirements listed in lab 1a, and the grading policy. Name the files **Date.h** and **Date.cpp**, and fully test them with your own test CPP. Name the functions exactly as Dietel does.

*Post **Date.h** and **Date.cpp** to the COMSC server for credit.*

Lab 10b Using `vectors` `Rider.h`, `Rider.cpp`, `Elevator.h`, and `Elevator.cpp`
Finalize the `Rider` and `Elevator` classes, which you started in labs 8 and 9. These should be defined in new files: **Rider.h**, **Rider.cpp**, **Elevator.h**, and **Elevator.cpp**. Use the same Floor class that you used in previous labs. Do *not* use friend relationships.

The completed version of the `Rider` class will have an overloaded assignment operator, so that there can be vectors of riders. The completed version of the `Elevator` class will include riders. (The lab 9 version modeled an empty elevator). Make these changes:

RIDER CLASS CHANGE:
1. Add an overloaded assignment operator (`operator=`) as a public member function. This is needed because the class has const data members and assignment is required by the STL `vector` template's `push_back` function.

ELEVATOR CLASS CHANGES:
1. Add a private data member for storing the riders. Since it is to store objects of the `Rider` class, use the data type `vector<Rider>`, which is essentially a self-sizing array of riders.

2. Add these four inline functions:

```
bool hasRiders() const; // returns false if riders vector is empty
int getRiderCount() const; // return size of riders vector
int getCapacity() const; // return capacity
int getAvailableSpace() const; // return capacity minus size of riders vector
```

3. Add these three non-inline functions:

```
vector<Rider> removeRidersForDestinationFloor(); // remove riders from vector whose destination is reached
void addRiders(const vector<Rider>&); // copy riders from parameter vector to riders vector
void setDestinationBasedOnRiders(); // reset toFloor based on riders' destinations
```

The first two functions are straight-forward and self-explanatory, both dealing with vectors. The third requires some logic. Check each rider on the elevator. Determine whose destination floor is closest to the elevator's current location, and set the elevator's destination floor to the same. *The logic for these functions will be discussed in lecture.*

Be careful in Elevator::setDestinationBasedOnRiders()! You are supposed to calculate the distance from the *elevator* to the rider's destination floor. Do NOT calculate the distance from the *elevator's destination* to the rider's destination floor.

4. Add statements to the Elevator's `operator<<` friend function to say something about the number of riders. For example: *no riders*, *1 rider*, *10 riders*. Correct grammar is required -- that is, do not say "one riders". It should output *all on one line* of the console screen, so that results for larger simulations will be easier to read.

Note: Be sure to properly apply the rules for includes and forward declarations!

Hint: The `abs` function is in the `cstdlib` library, and it works for INT values. The `fabs` function is also in the `cstdlib` library, and it works for FLOAT and DOUBLE values.

Post **Rider.h**, **Rider.cpp**, **Elevator.h**, and **Elevator.cpp** to your **lab10** folder on the COMSC server. Do *not* submit the **ElevatorSimulation3.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 both previous ElevatorSimulation CPPs, and the following **ElevatorSimulation3.cpp**:. Here's what it should do: (1) move the elevator 100" up, then (2) add 12 riders take them to the ground floor, then (3) move the elevator 100" down, then (4) add 1 rider and take that rider to the ground 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 <algorithm>
using std::sort;

#include "Floor.h" //from lab 8
#include "Rider.h"
#include "Elevator.h"

int main()
{
  // create an elevator (capacity 12, speed 5 in/sec)
  const Floor startingFloor(0); // ground floor
  Elevator e(12, 5, startingFloor);
  cout << "The only elevator in this test: " << e << endl;

  // testing the getCapacity function
  cout << "Capacity is " << e.getCapacity();
  if (e.getCapacity() == 12) cout << "(correct)\n"; else cout << "(INCORRECT)\n";

  // move the elevator up for 20 seconds
  e.closeDoor();
  e.setDestination(0);
  e.setDirectionUp();
  int i, count = 0;
  for (i = 0; i < 20; i++)
    e.moveUp();
  e.setIdle();
  e.openDoor();
  cout << "Moved up for 20 seconds: " << e << endl;

  // add riders to the elevator, to capacity
  vector<Rider> riders;
  int n = e.getAvailableSpace();
  for (i = 0; i < n; i++)
    riders.push_back(Rider(startingFloor));
  e.addRiders(riders);
  cout << "Added riders to capacity: " << e << endl;
  cout << "#of riders is " << e.getRiderCount();
  if (e.getRiderCount() == 12) cout << "(correct)\n"; else cout << "(INCORRECT)\n";
  vector<Rider> originalRiders(riders);

  // move down to first floor until it reaches
  e.setDestinationBasedOnRiders();
  e.closeDoor();
  e.setDirectionDown();
  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 destination floor: " << e << endl;

  // done
  e.openDoor();
```

```
  riders = e.removeRidersForDestinationFloor();
  cout << "Opened door, let out " << riders.size() << " riders: " << e << endl;
  cout << "#of riders is " << e.getRiderCount();
  if (e.getRiderCount() == 0) cout << "(correct)\n"; else cout << "(INCORRECT)\n";
  e.setIdle();

  // see if same riders got off who got on originally
  if (riders.size() < originalRiders.size())
    cout << "ERROR: LOST RIDERS!\n";
  else if (riders.size() > originalRiders.size())
    cout << "ERROR: MYSTERIOUSLY-CREATED RIDERS!\n";
  else
  {
    sort(riders.begin(), riders.end());
    sort(originalRiders.begin(), originalRiders.end());
    n = riders.size();
    for (i = 0; i < n; i++)
    {
      if (!(riders[i] == originalRiders[i]))
        cout << "ERROR: RIDER#" << i << " HAS NEW IDENTITY!\n";
  } }

  // now, move the elevator down for 20 seconds
  e.closeDoor();
  e.setDestination(0);
  e.setDirectionDown();
  for (i = 0; i < 20; i++)
    e.moveDown();
  e.setIdle();
  e.openDoor();
  cout << "Moved down for 20 seconds: " << e << endl;

  // add a rider going up
  riders.clear();
  riders.push_back(Rider(startingFloor));
  e.addRiders(riders);
  cout << "#of riders is " << e.getRiderCount();
  if (e.getRiderCount() == 1) cout << "(correct)\n"; else cout << "(INCORRECT)\n";

  // move up to first floor until it reaches
  e.setDestinationBasedOnRiders();
  e.closeDoor();
  e.setDirectionUp();
  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 << "Move to starting floor: " << e << endl;

  // done
  e.openDoor();
  e.removeRidersForDestinationFloor();
  e.setIdle();
  cout << "Test end: " << e << endl;
  cout << "#of riders is " << e.getRiderCount();
  if (e.getRiderCount() == 0) cout << "(correct)\n"; else cout << "(INCORRECT)\n";

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

Command Line Compiling With Visual C++:
Here is the command for compiling this project in Visual C++ (assuming that main is in a file named **ElevatorSimulation3.cpp**):

```
cl ElevatorSimulation3.cpp Rider.cpp Elevator.cpp Floor.cpp -EHs
```

This creates an executable named **ElevatorSimulation3.exe**. To run, type the command **ElevatorSimulation3**.

Alternatively, to compile each code module separately:

```
cl ElevatorSimulation3.cpp -c -EHs
cl Rider.cpp -c -EHs
cl Elevator.cpp -c -EHs
cl Floor.cpp -c -EHs
cl ElevatorSimulation3.obj Rider.obj Elevator.obj Floor.obj
```

Command Line Compiling With g++:
Here is the command for compiling this project in g++ (assuming that main is in a file named **ElevatorSimulation3.cpp**):

```
g++ ElevatorSimulation3.cpp Rider.cpp Elevator.cpp Floor.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++ -c ElevatorSimulation3.cpp -o ElevatorSimulation3.o
g++ -c Rider.cpp -o Rider.o
g++ -c Elevator.cpp -o Elevator.o
g++ -c Floor.cpp -o Floor.o
g++ ElevatorSimulation3.o Rider.o Elevator.o Floor.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*
