COMSC-200 Lab 9
Dealing With `const` Members

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 **lab9** folder provided.

Lab 9a [`Array.h and Array.cpp`]
Write the class as presented in figures 11.6 and 11.7 of our Deitel textbook, with modifications per the universal requirements listed in lab 1a, and the grading policy. Name the files **Array.h** and **Array.cpp**, and fully test them with your own test CPP. Name the functions exactly as Dietel does.

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

Lab 9b Using Inline Functions And Constants [`Elevator.h and Elevator.cpp`]
Define an early version of the `Elevator` class in two new files: **Elevator.h** and **Elevator.cpp**. The final versions of these files will be completed in the next lab assignment, after we cover the `vector` class next week. Since we will use a `vector` to "store" the riders, this early version of the `Elevator` class will deal with everything but the riders. Do *not* use friend relationships, except for stream insertion operators.

Use the following in the class definition, which makes extensive use of inline functions and constants. Note that the H file will require the compiler directives, declarations, and includes, which you must determine.

```
class Elevator
{
  // class-defined constants
  static int elevatorID; // initialize to zero; use to assign unique ID to each elevator
  static const int IDLE; // a unique numeric code
  static const int UP; // another unique numeric code
  static const int DOWN; // yet another unique numeric code

  // private member variables
  const int ID; // a unique identifier
  const int capacity; // max number of riders, set in constructor
  const int speed; // inches per second, up or down, set in constructor
  const Floor* toFloor; // destination floor, initialized in constructor to zero
  int location; // inches above ground level, initialized in constructor based on starting floor
  int direction; // equal to IDLE, UP, DOWN, initialized to IDLE in constructor
  bool doorOpen; // initialized to true in constructor

  public:
  Elevator(const int, const int, const Floor&); // capacity, speed, and starting floor

  // inline functions
  bool isDoorOpen() const; // returns value of doorOpen
  bool isIdle() const; // returns true if direction is IDLE
  bool isDirectionUp() const; // returns true if direction is UP
  bool isDirectionDown() const; // returns true if direction is DOWN
  void closeDoor(); // set doorOpen to false
  void openDoor(); // set doorOpen to true
  void setIdle(); // set direction to IDLE
  void setDirectionUp(); // set direction to UP
  void setDirectionDown(); // set direction to DOWN
  void moveUp(); // increment location by #inches per second speed of the elevator
  void moveDown(); // decrement location by #inches per second speed of the elevator
  void setDestination(const Floor* floor); // set toFloor pointer
  const Floor& getDestination() const; // return reference to toFloor (warning!)
  int getLocation() const; // return location value
  bool hasADestination() const; // return false if toFloor is zero

  // non-inline functions
```

```
  bool isNearDestination() const; // true if distance to destination is less than OR EQUAL TO the speed
  void moveToDestinationFloor(); // set location to that of destination floor (if there is one)

  // friend function
  friend ostream& operator<<(ostream&, const Elevator&);
};
```

Write the functions labeled as `// inline functions` as inline functions into the H file presented above. Write the others as regular, non-inline functions in the CPP file. Remember that all functions that are not void should return something, even if the destination floor value is zero. The actual return value in such cases is going to be ignored, so it does not matter what you return, but the compiler does not like to get to the end of such functions without a return statement! Advice for writing these functions will be presented in lecture.

Be sure to NEVER call Elevator::getDestination() without first calling Elevator::hasADestination() to make sure that the elevator has a destination. That's what the "warning" is about above. Do NOT call getDestination if toFloor is zero.

The overloaded stream insertion operator should include this information about the elevator: it's location (inches above the ground), direction of movement (up, down, or idle), and door status (open or closed). It should output *all on one line* of the console screen, so that results for larger simulations will be easier to read.

Note that some of these functions require knowledge of the **Floor** class, in order to determine a floor's location. Use the version of **Floor** class from lab 8. If you need to make corrections to the lab 8 version, do so and post the corrected files to your lab 9 folder.

Note that the **Rider** class is not needed in this lab. At this early stage of development, elevators and floors do not know about riders. But they will be introduced to them in the next lab.

Note: Be sure to properly apply the rules for includes and forward declarations! For example, if your H or CPP contains only Rider* and Rider&, and does not call any Rider member functions, use `class Rider;`. Otherwise use `#include "Rider.h"`.

Variations: You may use `enum`, instead of `static const int`, if you wish to do so. If you do use `static const int`, you can assign their values in their declaration, if you wish to do so.

*Post **Elevator.h** and **Elevator.cpp** to the COMSC server for credit.* Do *not* submit the **ElevatorSimulation2.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 the previous ElevatorSimulation CPP, and the following **ElevatorSimulation2.cpp**. If you don't see 20 sets of output as the elevator descends from 100 to 0, or 20 sets as it ascends from -100 to 0, then you are not doing something right!. Here's what it should do: (1) start with the elevator at the ground floor, then (2) move the elevator up for 20 seconds, then (3) let the elevator find its way back to the ground over 20 seconds, then (4) move the elevator down for 20 seconds, then (5) let the elevator find its way back to the ground over 20 seconds.

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

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

// TESTING THE ELEVATOR CONTROLS
// 1. Starting at the ground floor, send the elevator up for 20 seconds,
//    then let it go down until it finds and stops at the ground.
// 2. Starting at the ground floor, send the elevator down for 20 seconds,
//    then let it go up until it finds and stops at the ground.
int main()
{
  int i; // loop counter

  // create one floor and one elevator
  const Floor groundFloor(0); // the ground floor
  Elevator e(12, 5, groundFloor); // capacity 12, speed 5 in/sec
  cout << e << " (should at zero inches, idle, with the door open)\n";

  // move the elevator up for 20 seconds
  e.closeDoor();
  e.setDestination(0);
  e.setDirectionUp();
  for (i = 0; i < 20; i++)
    e.moveUp();
  e.setIdle();
  e.openDoor();
  cout << e << " (should be at 100 inches above the ground)\n";

  // move down until it reaches the upper floor -- should take 20 steps
```

```
  e.setDestination(&groundFloor);
  e.closeDoor();
  e.setDirectionDown();
  while(!e.isNearDestination())
  {
    if (e.isDirectionUp()) e.moveUp();
    else if (e.isDirectionDown()) e.moveDown();
    cout << e << endl; // this should output 19 times
  }
  e.moveToDestinationFloor();
  cout << e << " (should be at ground level)\n";

  // done
  e.openDoor();
  e.setIdle();
  cout << e << " (should be idle with the door open)\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 << e << " (should be at 100 inches below the ground)\n";

  // move up to first floor until it reaches -- should take 20 steps
  e.setDestination(&groundFloor);
  e.closeDoor();
  e.setDirectionUp();
  cout << e << " (should be going up with the door closed)\n";
  while(!e.isNearDestination())
  {
    if (e.isDirectionUp()) e.moveUp();
    else if (e.isDirectionDown()) e.moveDown();
    cout << e << endl; // this should output 19 times
  }
  e.moveToDestinationFloor();
  cout << e << " (should be at ground level)\n";

  // done
  e.openDoor();
  e.setIdle();
  cout << e << " (should be idle with the door open)\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 **ElevatorSimulation2.cpp**):

```
cl ElevatorSimulation2.cpp Elevator.cpp Floor.cpp -EHs
```

This creates an executable named **ElevatorSimulation2.exe**. To run, type the command **ElevatorSimulation2**.

Alternatively, to compile each code module separately:

```
cl ElevatorSimulation2.cpp -c -EHs
cl Elevator.cpp -c -EHs
cl Floor.cpp -c -EHs
cl ElevatorSimulation2.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 **ElevatorSimulation2.cpp**):

```
g++ ElevatorSimulation2.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 ElevatorSimulation2.cpp -o ElevatorSimulation2.o
g++ -c Elevator.cpp -o Elevator.o
g++ -c Floor.cpp -o Floor.o
g++ ElevatorSimulation2.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*
