COMSC-200 Lab 8
Operator Overloading

GOOD PROGRAMMING PRACTICES *show / hide*

ABOUT THIS ASSIGNMENT
In this lab session, you begin the **ElevatorSimulation** program -- a program that will be completed over labs 8-10 and 14-16. This is not the same simulation that appears in the Deitel textbook -- it is a different approach to the solution. It allows for simulations involving multiple elevators and many floors. 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 **lab8** folder provided.

Lab 8a [`PhoneNumber.h` and `PhoneNumber.cpp`]
Write the class as presented in figures 11.3 and 11.4 of our Deitel textbook, with modifications per the universal requirements listed in lab 1a, and the grading policy. Name the files **PhoneNumber.h** and **PhoneNumber.cpp**, and fully test them with your own test CPP. Name the functions exactly as Dietel does.

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

Lab 8b Using Operator Overloading [`Floor.h`, `Floor.cpp`, `Rider.h`, and `Rider.cpp`]
This exercise is the beginning of a new elevator simulation, which will be developed in labs 8-10 and 14-16. It provides a way for you to see two different approaches to solving this problem, because there is also an elevator simulation presented in our Deitel textbook.

This new elevator simulation involves objects of four classes: `Rider`, `Floor`, `Elevator`, and `Building`. These four define the main "characters" in our "story". Note that this is less complicated than the Deitel solution, which involves three times as many classes.

In this lab, we will write our first version of the `Rider` class, and a very early version of the `Floor` class. A "rider" is a person who enters the program by showing up at a floor, with a destination floor in mind. Riders exit the program when they arrive by elevator at their destination floor, and get off of the elevator.

Here are the specifications for the `Floor` class:

1. It should consist of two files: **Floor.h** and **Floor.cpp**. It's CPP should contain only a #include of it's H (plus your identifying comments), because all functions are inline.

2. It has one private data member: which is the location of the floor, as measured in integer inches above ground level. This value is set via a public constructor with one integer parameter variable. It is `const`, since floors do not move once they are created.

3. It has one `const` public member function: **`getLocation()`**, which returns the floor's location, as an `int`.

Here are the specifications for the `Rider` class:

1. It should consist of two files: **Rider.h** and **Rider.cpp**.

2. Its data members are to be *private*. There will be no need for the other classes to directly access its data members, so none will need to be declared as friend classes. Do *not* use friend relationships.

3. Its data members are to be `const` wherever possible, because they will never change for a `Rider` object once it is created.

4. Its data members should include: (a) a `const` integer identification number, to uniquely identify a rider, and (b) a `const` pointer to a `const` object of the `Floor` class, representing a rider's destination floor. *(We could use a Floor reference instead of a Floor pointer, but an overloaded assignment operator will be added in a future lab, and that will not work with reference variables.)*

5. It should have a public constructor, with one `const` parameter: a reference to the destination floor. The rider's identification number should be assigned in the constructor, but it should *not* be included in the parameter list. This may be inline or not -- it's your choice. *HINT: Use a static integer to track and assign identification numbers.*

ANOTHER HINT: If you have nothing to put in the Rider.cpp, because you wrote everything inline in Rider.h, you're not doing this right! You cannot initialize static members in an H file.

6. Since `vector`s of riders will be used to track riders in the `Elevator` and `Floor` classes in future labs, provide overloaded operators for these operators: less than (<), and is equal to (==). Use the identification number for comparisons -- two riders are equal if they have the same identification number, and a rider is "less than" another if its identification number is less. These may be inline or not -- it's your choice.

7. Provide a `const` member function named `getDestination` to return a const reference to the destination floor. This may be inline or not -- it's your

choice. *(Note that the constructor does not allow the destination floor pointer to be zero, because it is passed as a reference instead of a pointer. So there is no need for validation or error checking.)*

*Post **Floor.h**, **Floor.cpp**, **Rider.h**, and **Rider.cpp** to the COMSC server for credit.* Save these file, because modifications will be made to this program in the next lab. Do *not* submit the **ElevatorSimulation1.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.

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"`. DO NOT put a #include in any of your H or CPP files if a forward declaration will suffice.

To test your code, compile, build, and run with the following **ElevatorSimulation1.cpp** Here's what's going on: (1) two different riders are created, and they should be shown to be different from eachother, (2) one rider is cloned, and should be shown to be the same as its clone, with the same destination.

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

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

int main()
{
  const Floor f(0); // a floor at ground level

  const Rider a(f); // a rider whose destination is "f"
  const Rider b(f); // a second rider whose destination is "f"
  if (a == b) cout << "equal (INCORRECT)\n"; else cout << "unequal (correct)\n";
  if (a < b) cout << "less (correct)\n"; else cout << "greater or equal (INCORRECT)\n";

  const Rider c(a); // a clone of the first rider
  if (a == c) cout << "equal (correct)\n"; else cout << "unequal (INCORRECT)\n";
  if (a < c) cout << "less (INCORRECT)\n"; else cout << "greater or equal (correct)\n";

  // checking the cloning of the Floor*
  if (&(a.getDestination()) == &(c.getDestination())) cout << "equal (correct)\n"; else cout << "unequal (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 **ElevatorSimulation1.cpp**):

```
cl ElevatorSimulation1.cpp Rider.cpp Floor.cpp -EHs
```

This creates an executable named **ElevatorSimulation1.exe**. To run, type the command **ElevatorSimulation1**.

Alternatively, to compile each code module separately:

```
cl ElevatorSimulation1.cpp -c -EHs
cl Rider.cpp -c -EHs
cl Floor.cpp -c -EHs
cl ElevatorSimulation1.obj Rider.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 **ElevatorSimulation1.cpp**):

```
g++ ElevatorSimulation1.cpp Rider.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 ElevatorSimulation1.cpp -o ElevatorSimulation1.o
g++ -c Rider.cpp -o Rider.o
g++ -c Floor.cpp -o Floor.o
g++ ElevatorSimulation1.o Rider.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*