Programming Assignment #1

This assignment will give you some practice with structures, namespaces, references, dynamic memory allocation, and dynamically-allocated 2-dimensional arrays. It will also give you a chance to learn about interfaces and how to read source code. The task is to implement a very simple version of the popular board game *Battleship*. The player will place boats in an "ocean" that you create and will attempt to sink each one by taking shots into the ocean.

```
0   0   0   0   0   0   0   0
0   0   0   0   0   2   0   0
0   0   0   0   0   2   0   0
0   1   1   1   1   2   0   0
0   0   0   0   0   2   0   0
3   3   3   3   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
```

An 8x8 board with 3 boats placed.

```
-1  -1  -1   0  -1  -1   0  -1
-1  -1  -1  -1  -1 102  -1   0
-1   0  -1  -1  -1 102  -1  -1
-1 101 101 101 101 102  -1  -1
-1  -1   0  -1  -1 102  -1  -1
103 103 103 103  -1  -1  -1  -1
-1  -1  -1  -1  -1  -1  -1  -1
-1  -1  -1  -1  -1  -1  -1  -1
```

The board after sinking all 3 boats.

Details
Instead of simply hard-coding the size of the board at 10x10, we are going to allow the player to specify the dimensions of the board. (The board is the ocean.) You should be able to handle boards of any rectangular size (square and non-square). You are also going to allow the player to specify the number of boats to place in the ocean. The only limit to the number of boats is the size of the ocean, since boats will not be allowed to overlap or leave the ocean. (Contrary to what Ferdinand Magellan's crew claimed, the world is flat, and if you go too far, you will fall off the end of it.)

In order to keep the implementation simple (we are studying new C++ techniques, afterall), all of the boats will be the same size. Also, only the player will be taking shots, meaning that the computer will not be trying to find the player's boats. Again, we are studying C++ in this course, not artificial intelligence. (However, after completing this assignment, you will have most everything you need to create a more sophisticated game, as I will demonstrate.)

Like all programs, there are many ways to implement this. For this assignment, you are given some header files to use as a starting point. These files give you the layout of the solution. The interface to the game is included in a header file named `WarBoats.h`. This file contains all of the information that the client (the player) needs. You are also given a file named `Ocean.h`, which defines what an *Ocean* is. A partial `Ocean.cpp` file is provided which includes the implementation of the display function called *DumpOcean*. (I want to be sure everyone's program prints the exact same thing and the only way to ensure that is if everyone has the same code.)

All of your implementation will be placed in `Ocean.cpp`, and this will be the only source file that you will submit. You are not allowed to include any other files in `Ocean.cpp`. There are a couple of files included already, but you will not need any others.

Function Details

There are only 5 functions that you need to write for this assignment, and three of them are fairly simple.

Functions listed in the interface
`Ocean *CreateOcean(int num_boats, int x_quadrants, int y_quadrants);`
The client calls this to create an ocean (game board). Client specifies the dimensions of the ocean and the number of boats that will be placed in it. An ocean will be dynamically allocated and a pointer to it will be returned to the client.
`void DestroyOcean(Ocean *theOcean);`
Client calls this to clean-up after the game. The function destroys an ocean that was created above by simply making sure that all memory that was allocated is de-allocated (deleted) properly.
`BoatPlacement PlaceBoat(Ocean &ocean, const Boat& boat);`
The client calls this to place boats in the ocean. The client will create a boat and pass it to the function. The function must ensure that the boat will "fit" in the ocean. Do this by checking the location where it is to be placed, the orientation (horizontal/vertical), and whether or not it will overlap with another boat or stick outside of the ocean. The return value indicates whether or not the boat could be placed in the ocean. See `WarBoats.h` for valid values of `BoatPlacement`.
`ShotResult TakeShot(Ocean &ocean, const Point &coordinate);`
Client calls this in an attempt to hit one of the boats. The coordinate parameter indicates where the client is attempting to strike. There are several possible results: Hit, Miss, Sunk, Duplicate, or Illegal. Hit, Miss, and Duplicate are obvious. Sunk is returned when a shot hits the last undamaged part of a boat. Sunk implies Hit. Illegal is any coordinate that is outside of the ocean (e.g. x or y less than 0 or outside the range). See `WarBoats.h` for valid values of `ShotResult`.
`ShotStats GetShotStats(const Ocean &ocean);`
Client calls this to get the current statistics of the game. See `WarBoats.h` for the `ShotStats` structure.

To get a feel for how these functions are supposed to work, you simply need to look at the code that is provided in the sample driver. This is the best way to see how they are used. All of these functions are called in the sample driver, some of them many times. The client (driver) will typically call *CreateOcean* and *DestroyOcean* only once, at the beginning and end of the game, respectively. The driver will call *PlaceBoat* once for each boat to place. However, if the driver tries to place a boat in an illegal location, you will return a value indicating it's an illegal location and the driver will try again with another location. Once the boats are placed, the player (driver) makes repeated calls to *TakeShot*, which is the primary action during the game. The player can stop taking shots when all of the boats have been sunk.

The only functions that require non-trivial code are *PlaceBoat* and *TakeShot*. The *PlaceBoat* function requires you to make sure that a boat can be legally placed at the specified position. You'll have to have some logic that checks to make sure the boat placement is legal. The *TakeShot* function requires you to validate the coordinates and then to determine the type of shot (e.g. hit, miss, duplicate, etc.)

What to submit

You must submit your implementation file (Ocean.cpp) file to the appropriate submission page. Note that you are not submitting any other files.

Files	Description
Ocean.cpp	The implementation file. **All implementation goes here**. You must document this file (file header comment) and functions (function header comments) using the proper Doxygen tags as previously discussed.

Usual stuff

Your code must compile cleanly with the specified compilers to receive full credit. Details about how to submit are posted on the course website and in the syllabus.

Make sure your name and other info is on all documents.