Lab1: Classes and Structures

Let’s play Battleship!

We are attempting to emulate a simple version of the game Battleship. On a square board of user-defined size (4x4 for example), a user-defined number of ships are placed at random locations. Each ship occupies just ONE space on the board/grid. The user is then asked repeatedly to guess the location of the ship(s) until all ships are sunk and the game ends.

Here’s what a sample out put would look like:

Welcome to the B A T T L E S H I P !

How vast are your oceans? (Enter an integer for the size of the board): **4**

How many ships are in your fleet? (Enter an integer for the number of ships) **3**

Let the battle begin!

Enter your guess (Enter two integers for the ship’s location) **1 5**

Out of bounds!

Enter your guess (Enter two integers for the ship’s location) **1 3**

Sunk! Yay, 2 ships left.

Enter your guess (Enter two integers for the ship’s location) **2 2**

Miss!

Enter your guess (Enter two integers for the ship’s location) **2 1**

Sunk! Yay, 1 ships left.

Enter your guess (Enter two integers for the ship’s location) **2 4**

Sunk! Yay, 0 ships left. Congratulations! You won the game in **5** moves.

**Checkpoint 1 – due Tuesday July 8th at 10 pm**

Create a class *ship*. Define its attributes. Implement class functions: at the very least a constructor, destructor, get and set functions for the *ship* class. Besides the .h and .cpp files for the ship class, create a driver *main* program (*main* file), which should be able to create a ship (with user input for location) and then output to the console the location of the ship and the ship status (sunk or not).

**Checkpoint 2 due Wednesday July 9th at 10 pm**

Modify your driver *main* program (*main* file) to implement the functionality of the Battleship game.

1. Initialize the game board
   1. Read input from the user about the board size and the number of ships
   2. Create an array of *ship* objects, each positioned at random locations inside your board
2. Create a function called *operation()* which should be called in a loop until the game ends.
   1. Read input from the user about the next target location to fire upon.
   2. Check if there is a ship at the input location.
   3. If there is no ship send a “Miss!” message to the user.
   4. If a ship is located on the target, send a “Sunk!” message to the user, along with how many ships are left.
   5. If there are no more ships left, congratulate the user and output the number of guesses the user took to beat the game. Exit the game.