Assignment 4: Battleship

Due: 23:59, Sat 4 Nov 2023 File name: `battleshipboard.cpp` `battleshipgame.cpp` `battleshipboard.h` Full marks: 100

Introduction

The objective of this assignment is to let you practice the use of arrays and pointers. You will write a C++ console application program to simulate a classic board game called *Battleship*.

Battleship is a two-player strategy-type guessing game. Each player has their own gameboard which is typically a 10x10 square grid. Each player secretly arranges their ships on their grid. The ships can be placed horizontally or vertically, but not diagonally. The ships cannot overlap each other. The game's objective is to accurately guess and sink all of your opponent's ships before they sink yours.

Each player's fleet consists of five ships of different types listed in Table 1. The size of a ship means the number of consecutive cells it occupies on the game board.

Table 1: Types of ships, their sizes, and symbols

	Ship Type	Ship Symbol	Size (No. of cells)
1	Carrier	C	5
2	Battleship	B	4
3	Destroyer	D	3
4	Submarine	S	3
5	Patrol Boat	P	2

Players take turns calling out a grid coordinate to target a specific cell on their opponent's grid. For example, "B5" or "F9". The opponent must respond with "hit" if the targeted cell contains a part of a ship, or "miss" if it is an empty cell. The attacking player marks the result of their shot on their own tracking grid by using different symbols (X and O, say) to represent hits and misses. If a player hits all the cells of a ship successfully, their opponent must announce "sunk" to indicate that the ship is completely destroyed. The player who first sinks all the ships of the opponent is the game winner. You may play this online game version briefly for more ideas about the gameplay.

```
Enemy Ocean          Your Ocean
   A B C D E F G H      A B C D E F G H
 1 . . . . . . . .    1 . . . . . D D D
 2 . . . . . . . .    2 . . . . . . . .
 3 . . . . . . . .    3 . . . . . . . B
 4 . . . . . . . .    4 C C C C C . . B
 5 . . . . . . . .    5 . . . . . P P B
 6 . . . . . . . .    6 . S S S . . . B
 7 . . . . . . . .    7 . . . . . . . .
 8 . . . . . . . .    8 . . . . . . . .
Enemy Fleet: CBDSP   Your Fleet: CBDSP
```

Figure 1: The Battleship game board simulated in the console (game mode)

You are going to write a C++ program to implement this game which is played via the console. For our case, one player is the program user and the other the computer. Figure 1 depicts the game board which shows the 2-D grids of the players arranged side by side. The left grid belongs to the computer player (enemy). The locations of the enemy's fleet are hidden. The gameplay is illustrated in Figure 2. The user enters a location, F3, on the left board to shoot. Now, F3 is a miss (denoted by 'O'). The turn goes to the computer; it shoots at G5 on the right grid, resulting in a hit on P (Patrol Boat). So, G5 is marked as 'X'. Then the user shoots at D4, resulting in a hit of an unknown ship. The turn goes back to the computer; it shoots at G6 but it is an empty cell, so a miss ('O') is marked at G6 on the right grid.

```
Enemy Ocean          Your Ocean
   A B C D E F G H      A B C D E F G H
 1 . . . . . . . .    1 . . . . . D D D
 2 . . . . . . . .    2 . . . . . . . .
 3 . . . . . O . .    3 . . . . . . . B
 4 . . . X . . . .    4 C C C C C . . B
 5 . . . . . . . .    5 . . . . . P X B
 6 . . . . . . . .    6 . S S S . . O B
 7 . . . . . . . .    7 . . . . . . . .
 8 . . . . . . . .    8 . . . . . . . .
Enemy Fleet: CBDSP   Your Fleet: CBDSP
```

Figure 2: The Battleship gameplay (game mode)

The bottom of each grid shows a "status bar" or "health bar" about the fleet: "CBDSP" means all the five ships are alive. After some rounds of gameplay, some ships have sunk. For example, in Figure 3, the computer has sunk two ships, denoted by letters P and B, of the human player. The health bar is updated as "CXDSX", meaning the user's **P**atrol Boat and **B**attleship have sunk (X).

```
Enemy Ocean          Your Ocean
   A B C D E F G H      A B C D E F G H
 1 . . . . . . . .    1 . . . . . D D D
 2 . . . X . . . .    2 . . . . . . . .
 3 . . . X . O . .    3 . . . . . . . X
 4 . . O X . . . .    4 C C C C C . O X
 5 . . . O . . . .    5 . . . . . X X X
 6 . . . . . . . .    6 . S S S . . O X
 7 . . . . . O X .    7 . . . . . . . .
 8 . . . . . . . .    8 . . . . . . . .
Enemy Fleet: CBXSP   Your Fleet: CXDSX
```

Figure 3: The Battleship gameplay continued (game mode)

When all the ships of a player are sunk, i.e., when the player's fleet status bar becomes "XXXXX", the game is over. For example, Figure 4 shows the final snapshot of the game boards. The computer player (enemy fleet) is defeated; the winner is the human player who still has three ships alive on the right board.


```
Enemy Ocean          Your Ocean
   A B C D E F G H      A B C D E F G H
 1 . . . . . . . .    1 . . . . . D D D
 2 . . . X . . . .    2 . . . . . . . O
 3 . X . X . O . .    3 O O . . . . O X
 4 . X O X . . . .    4 X X X C C O O X
 5 . X . O . . X .    5 . O . . O X X X
 6 . X . . . . X X    6 . S S S . O O X
 7 . X . . . O X X    7 . . . . . . . O
 8 . . X X X . X .    8 . . . . . . . .
Enemy Fleet: XXXXX   Your Fleet: CXDSX
Game over!
Your fleet wins!
```

Figure 4: Reaching the end of the game (game mode)

To facilitate testing and debugging, the program can be started in a debug mode which reveals the ship locations of the computer player since Round 1 (See Figure 5 below).

```
Enemy Ocean          Your Ocean
   A B C D E F G H      A B C D E F G H
 1 . . . . . . . .    1 . . . . . D D D
 2 . . . D . . . .    2 . . . . . . . .
 3 . C . D . . . .    3 . . . . . . . B
 4 . C . D . . . .    4 C C C C C . . B
 5 . C . . . . B .    5 . . . . . P P B
 6 . C . . . . B P    6 . S S S . . . B
 7 . C . . . . B P    7 . . . . . . . .
 8 . . S S S . B .    8 . . . . . . . .
Enemy Fleet: CBDSP   Your Fleet: CBDSP
```

Figure 5: The gameboard of round 1 of a game started in debug mode

Program Specification

This section describes the requirements, starter code, some necessary functions to complete, and program flow.

Basic Requirements

- You *cannot declare any global variables* (variables declared outside any functions) in all files.
- Your program is scalable to board size, which can be changed at compile time (not hardcoded).

Starter Code

- To help you get started, you are provided with all the required source files:

File name	What's inside
`battleshipboard.h`	Several global constants defining the board size, characters denoting a hit, miss and blank cell, static arrays defining the names and sizes of the 5 types of ships, a C++ struct called `Fleet` which stores the name of the player and a 5-element int array implementing the status (health) bar of the fleet, and the function prototypes of the functions required by the client's `main()` function.

`battleshipboard.cpp`	The source file implementing all the functions for printing the game boards and fleet status bars, validating, and making human moves and computer moves to shoot board cells, and placing ships on the boards.
`battleshipgame.cpp`	The client source file implementing the `main()` function which contains the main game loop for alternating turns for the human and computer players. It includes the header file and calls the functions provided by the above source file.

- Basically, you don't need to change the header file unless you need to add extra functions to the source files and their function prototypes to the header file.
- To achieve the second requirement mentioned above, indeed, the header file has defined a global constant integer N for the board size to make it scalable. It is now set to 8. We may grade your program with N changed to a different value within the range of 5 to 15. When you test your program against smaller or bigger boards, you can change N to other values. Reset N to 8 before you submit the header file.
- The source files have provided the basic program flow and organized the entire program into different functions with well-defined parameter lists. Your main task in this assignment is to search for all *"TODO: Add your code"* comments and fill in the missing parts of code. After adding your code, you should remove all occurrences of the *"TODO: …"* comments.

Global Constants and the `Fleet` Structure

There are several important global constants and a C++ struct called `Fleet` defined in the header file, which are recapped as follows.

```
// Global constants
const int N = 8;             // board size
const char HIT = 'X';        // hit
const char MISS = 'O';       // miss
const char BLANK = '.';      // empty cell
const string SHIP_TYPE[5] = {"Carrier", "Battleship", "Destroyer", "Submarine",
                             "Patrol Boat"};
const int SHIP_SIZE[5] = {5, 4, 3, 3, 2};  // initial health point of each ship

// Structures
struct Fleet {
    string name;   // a name denoting the player
    int alive[5];  // current health points of the player's ships
};
```

You must understand them before you can complete the missing code.

As mentioned, the integer N defines the board size. Whenever you have need to use the board size, you should use N instead of hardcoded values throughout every source file.

For `HIT`, `MISS` and `BLANK`, their meanings are obvious and used for printing the board content. For example, whenever you need to mark a hit, use `HIT` instead of hardcoding character 'X' in your code body.

The array `SHIP_TYPE` defines the names of the five types of ships listed in Table 1. There is no separate array defining the letter symbols C, B, D, S, P for the ships. When use of these symbols is needed, for example, printing a fleet status bar or printing ships on the boards, you may retrieve the letter from the `SHIP_TYPE` array elements, e.g., `SHIP_TYPE[0][0]` for 'C', `SHIP_TYPE[1][0]` for 'B', …, `SHIP_TYPE[4][0]` for 'P'.

The `SHIP_SIZE` array defines the size of each type of ship – 5 for Carrier, 4 for Battleship, etc. This array is useful when you need to check against the original size of each ship. For example, when placing ships onto the board at game start, you should know a ship's size in order to determine if the placement of it at a particular location is valid or not.

The `Fleet` struct is a composite data type that consists of two members: the string `name` and the 5-element `int` array called `alive`. A variable of `Fleet` type is mainly used to save the health status of a fleet. Each fleet has 5 ships. The 5 integer elements of `alive` record the health points of the ships. They are initialized as {5, 4, 3, 3, 2} for the ships {C, B, D, S, P} respectively. For example, the line of code below creates a fleet named "Enemy" with ships' health points initialized to their sizes:

```
Fleet enFleet = {"Enemy", {5, 4, 3, 3, 2}};
```

When a ship in the fleet receives hits, the corresponding element in this `alive` array is deducted. When an element becomes zero, it means the ship has sunk. The corresponding ship letter in a fleet status bar string should be shadowed by 'X' instead.

For updating the `alive` array in some functions, the `Fleet` struct is passed via a pointer parameter.

Provided and To-do Functions

Your program must contain the following functions. *Some of them are written for you already (Provided) and you shall not modify their contents.* The others will be completed by you (To-do). These functions shall be implemented in the source file `battleshipboard.cpp` with the function prototypes in the header file `battleshipboard.h`. These functions shall be called *somewhere* in your program. You must *not* modify the prototypes of all these functions. You can design extra functions if you find necessary.

`string fleetStatus(Fleet fleet)` (To-do)

Returns a 5-letter string showing which ships of the specified fleet have sunk. Check the `alive` array in the `fleet` argument to build this string. For a ship of `alive` element > 0, show its letter. For a ship with `alive` element = 0, show 'X' (HIT) instead.

`void printBoards(char board1[][N], char board2[][N], bool gameMode)` (To-do)

Prints the user and computer game boards side by side. For the top header line "Enemy Ocean Your Ocean" and the column headings (A B C …) of the boards, the starter code has done their printing for you. You need to add code to print row indexes (on the left) and board content (blanks, hits, misses, ship letters) for both game boards. Note that the `gameMode` argument is used to control whether ship letters are shown or hidden. If `gameMode` is true, you should print a dot (`BLANK`) instead of ship letters for the cells occupied by the ships. Also note that row indexes have fixed width of 2 characters. For single-digit row numbers, there is a space padding ahead of the number. There is a gap of two spaces between the two boards.

`bool isValidCell(int y, int x)` (To-do)

Returns true if the specified location (y, x) is within the board, and false otherwise.

`bool isValidShoot(char board[][N], int y, int x)` (To-do)

Returns true if the location (y, x) is valid to shoot, and false otherwise. A location (y, x) is valid to shoot if (y, x) is a valid cell and it has not yet been shot before (no matter hit or missed).

`int indexOf(char ship)` (To-do)

Returns the index of the `ship` letter in the SHIP_TYPE array. It returns -1 in case `ship` cannot be found in the array (although this won't happen by assumption).

`bool shoot(char board[][N], int y, int x, Fleet* fleet)` (To-do)

Carries out a shot on the board at location (y, x) in attempt to hit a ship of the specified fleet. This function should be called only after `isValidShoot(board, y, x)` has been called to confirm location (y, x) is a valid shoot location on the board. Set the cell at (y, x) to `HIT` if it is not a blank, and to `MISS` otherwise. If it is a hit, deduct the health point by 1 for the ship at that location. If the health point is deducted to zero, print a message (see Sample Runs for the format) to announce the hit ship of the fleet has sunk. Returns true if it is a hit, and false otherwise.

`bool getCellFromConsole(int& y, int& x)` (To-do)

Gets location (y, x) from the user via console input. Part of the code has been provided. Your task is only to convert the `row` integer (1-N) and `col` character into zero-based indices `y` and `x` respectively for accessing the 2-D array of the target board. Returns false if the `cin` operation fails, and true otherwise (the return statements have been done for you).

`bool placeShip(char board[][N], int y, int x, char ship, bool vertical = false)` (To-do)

Places ship on board at (y, x) in specified orientation (vertical or horizontal). For the ship with the letter `ship` and size (or length) of n cells, place the first cell of the ship at (y, x) only if every of the n cells lays on top of a blank cell within the board area in the specified orientation. For example, refer to Figure 6 below, putting the ship D at F1 horizontally (i.e., the `vertical` argument = false) is valid because all its 3 cells are on the board and do not overlap with any other ship's cells. If it were placed at G2, it would be invalid because its third cell will be off the board. As another example, placing ship B vertically at H3 is valid. If it were at G3 instead, it would become invalid because one of its cells overlaps with the ship P at location G5. This function returns true if the ship placement is valid, and false otherwise.

```
Enemy Ocean          Your Ocean
   A B C D E F G H      A B C D E F G H
 1 . . . . . . . .    1 . . . . . D D D
 2 . . . . . . . .    2 . . . . . . . .
 3 . . . . . . . .    3 . . . . . . . B
 4 . . . . . . . .    4 C C C C C . . B
 5 . . . . . . . .    5 . . . . . P P B
 6 . . . . . . . .    6 . S S S . . . B
 7 . . . . . . . .    7 . . . . . . . .
 8 . . . . . . . .    8 . . . . . . . .
Enemy Fleet: CBDSP   Your Fleet: CBDSP
```

Figure 6: About ship placement

`void manuallyPlaceShips(char board[][N])` (To-do)

For each of the 5 ships, prompt the user to enter 'h' or 'v' which decides the ship orientation ('h' for horizontal, 'v' for vertical, case-insensitive), followed by the ship location, i.e., column letter + row id (case-insensitive). The prompt message is in this form *"Enter h/v and location for <ship name>"*. Print error message *"Invalid ship location!"* if the input location is invalid for placing the ship in the specified orientation. Keep prompting until the user enters a valid orientation-location pair. Hint: (1) call `placeShip()` in this function; (2) use `cin` here for the orientation input only; for the location input, you should call the `getCellFromConsole()` function. You may also look at the provided `randomlyPlaceShips()` function which has a similar flow of steps except that their ship orientations and locations are generated randomly instead of getting via the console.

`void printStatus(Fleet fleet1, Fleet fleet2)` (Provided)

Prints the status strings (health bars) of both the specified fleets.

`bool getHumanMove(int& y, int& x)` (Provided)

Prompts the user for shoot location (`y, x`) by calling `getCellFromConsole(y, x)`.

`bool getComputerMove(char board[][N], int& y, int& x)` (Provided)

Gets shoot location (`y, x`) by the computer player's strategy. The implemented strategy is trying to shoot the neighborhood of an existing hit first before making a shot at a random location. The function will loop for `TRIALS` (hardcoded to 1000) times of random pick of a cell on the board. If the location is a hit, it will look at all the 4 cells (north, east, south, west) around the hit in a clockwise manner but with a random direction to start with (e.g., it may go to the east cell first, then to the south cell, then west, and finally north). If one of these 4 neighboring locations is a valid cell (not off the board) and not shot before (i.e., neither a hit nor a miss), the location will be set to (`y, x`) and the function returns true. If no such a neighboring cell is found after the loop of searching is done, a random cell of the board is set to (`y, x`) and the function returns true.

`void randomlyPlaceShips(char board[][N])` (Provided)

Places all ships with random orientations and random locations on the specified board.

Program Flow

The program flow of the game is described as follows. You should call the functions above to aid your implementation in the client program (`battleshipgame.cpp`). Most parts have been done for you. Missing parts are marked with TODO comments.

1. **(To-do)** The program starts with prompting the user to enter a seed value for seeding the pseudo-random number generator, Y/N (case-insensitive) for playing in game mode, and Y/N (case-insensitive) for placing ships manually by the user. To make it simple and avoid looping here, if the user input for playing in game mode is 'Y'/'y', the Boolean flag (`gameMode`) is set to true, and to false for all other replies such as 'N', 'n', 'x', 'Y', 'z', ... (no validation is required). The same handling applies to user input for placing ships manually or not (the `manualSetup` flag).
2. Create two boards (2-D `char` arrays) and two fleets (`Fleet` type).
3. Place ships on the board for the human player manually or randomly accordingly. If manually, prompt the user for ship orientations and locations.
4. Place ships on the board for the computer player randomly.
5. Print the players' game boards with status bars.

6. If it is human's turn (human taking first turn), keep prompting the user to enter a shoot location until it is valid (or else print error message *"Invalid shot location!"*).
7. If it is computer's turn, keeping generating a computer move (shot) until it is valid.
8. Carry out the shot to update the board (and status bar if hit).
9. **(To-do)** If the hit results in all opponent ships sunk, set game over flag to true and set winner.
10. Swap player to take the next turn.
11. Repeat steps 5–10 until the game over flag becomes true.
12. **(To-do)** When a game finishes, print the final snapshot of the gameboards **with game mode set to false** (this is to reveal the secret ship arrangement of the computer player in case the human loses the game). Print the game over message and the winner.

Reminders on Testing

You may assume all user inputs won't have errors on the value's type and number of values required. For example, for a shoot location input, the program expects one character and one integer like A1, then the user won't enter something else like "AA", "@@", "1A", etc. But the values should still be validated, e.g., the user may enter "Z1" and the program should tell it is an invalid location. For a ship location input, the program expects one letter for orientation, then one character and one integer for the location, then the user won't input fewer or more than these three values. Also, inputs should be assumed case-insensitive, location "A1" and "a1" or orientation "h" and "H" should both be acceptable.

Note the **difference in random number sequences generated on Windows and on macOS** due to different library implementations. Even for the same seed, the two platforms may generate different sets of random integers. The sample runs given below were conducted on macOS. For Windows users, please use the sample program for Windows to perform your correctness checks.

Sample Runs

In the following sample runs, the blue text is user input and the other text is the program printout. You can try the provided sample program for other input. *Your program output should be exactly the same as what the sample program produces* (same text, symbols, letter case, spacings, etc.). Note that *there is a space after the ':' symbol in the user prompt text*. *The final line ends with a newline*.

Sample Run 1

```
Enter seed: 12345↲
Play in game mode (Y/N)? y↲
Manually set up ships (Y/N)? y↲
Enemy Ocean         Your Ocean
   A B C D E F G H     A B C D E F G H
 1 . . . . . . . .   1 . . . . . . . .
 2 . . . . . . . .   2 . . . . . . . .
 3 . . . . . . . .   3 . . . . . . . .
 4 . . . . . . . .   4 . . . . . . . .
 5 . . . . . . . .   5 . . . . . . . .
 6 . . . . . . . .   6 . . . . . . . .
 7 . . . . . . . .   7 . . . . . . . .
 8 . . . . . . . .   8 . . . . . . . .
Enter h/v and location for Carrier: h A10↲
Invalid ship location!
Enter h/v and location for Carrier: x A1↲
Invalid ship location!
```

```
Enter h/v and location for Carrier: h E1↵
Invalid ship location!
Enter h/v and location for Carrier: h A1↵
Enter h/v and location for Battleship: v E1↵
Invalid ship location!
Enter h/v and location for Battleship: v h2↵
Enter h/v and location for Destroyer: H b 8↵
Enter h/v and location for Submarine: H F3↵
Invalid ship location!
Enter h/v and location for Submarine: V   f3↵
Enter h/v and location for Patrol Boat: zC1↵
Invalid ship location!
Enter h/v and location for Patrol Boat: hC1↵
Invalid ship location!
Enter h/v and location for Patrol Boat: hC8↵
Invalid ship location!
Enter h/v and location for Patrol Boat: hC5↵
Round 1:
Enemy Ocean          Your Ocean
   A B C D E F G H     A B C D E F G H
 1 . . . . . . . .   1 C C C C C . . .
 2 . . . . . . . .   2 . . . . . . . B
 3 . . . . . . . .   3 . . . . . S . B
 4 . . . . . . . .   4 . . . . . S . B
 5 . . . . . . . .   5 . . P P . S . B
 6 . . . . . . . .   6 . . . . . . . .
 7 . . . . . . . .   7 . . . . . . . .
 8 . . . . . . . .   8 . D D D . . . .
Enemy Fleet: CBDSP  Your Fleet: CBDSP
Enter attack location: B3↵
Round 2:
Enemy Ocean          Your Ocean
   A B C D E F G H     A B C D E F G H
 1 . . . . . . . .   1 C C C C C . . .
 2 . . . . . . . .   2 . . . . . . . B
 3 . O . . . . . .   3 . . . . . S . B
 4 . . . . . . . .   4 . . . . . S . B
 5 . . . . . . . .   5 . . P P . S . B
 6 . . . . . . . .   6 . . . . . . . .
 7 . . . . . . . .   7 . . . . . . . .
 8 . . . . . . . .   8 . D D D . . . .
Enemy Fleet: CBDSP  Your Fleet: CBDSP
Round 3:
Enemy Ocean          Your Ocean
   A B C D E F G H     A B C D E F G H
 1 . . . . . . . .   1 C C C C C . . .
 2 . . . . . . . .   2 . . . . . O . B
 3 . O . . . . . .   3 . . . . . S . B
 4 . . . . . . . .   4 . . . . . S . B
 5 . . . . . . . .   5 . . P P . S . B
 6 . . . . . . . .   6 . . . . . . . .
 7 . . . . . . . .   7 . . . . . . . .
 8 . . . . . . . .   8 . D D D . . . .
Enemy Fleet: CBDSP  Your Fleet: CBDSP
Enter attack location: Z1↵
Invalid shot location!
Enter attack location: A10↵
Invalid shot location!
Enter attack location: E3↵
```

```
Round 4:
Enemy Ocean          Your Ocean
   A B C D E F G H     A B C D E F G H
 1 . . . . . . . .   1 C C C C . . .
 2 . . . . . . . .   2 . . . . . O . B
 3 . O . . X . . .   3 . . . . . S . B
 4 . . . . . . . .   4 . . . . . S . B
 5 . . . . . . . .   5 . . P P . S . B
 6 . . . . . . . .   6 . . . . . . . .
 7 . . . . . . . .   7 . . . . . . . .
 8 . . . . . . . .   8 . D D D . . . .
Enemy Fleet: CBDSP  Your Fleet: CBDSP

... (too long, skipped)

Enemy Ocean          Your Ocean
   A B C D E F G H     A B C D E F G H
 1 X O . . O . O .   1 X X X X X O . O
 2 X . O . . O . .   2 O O O O O O O X
 3 X O . O X X X X   3 . . . . O X O X
 4 X . O X O . O .   4 . O . . O X O X
 5 X . O X O . O O   5 . . P P O X O X
 6 X O X X . O . O   6 . O . O . O . O
 7 X . . O . . O .   7 . O O . O . . .
 8 X O . O . O . O   8 O X X X . . . .
Enemy Fleet: XXXXP  Your Fleet: XXXXP
Enter attack location: C7↵
Enemy Patrol Boat sank!
Enemy Ocean          Your Ocean
   A B C D E F G H     A B C D E F G H
 1 X O . . O . O .   1 X X X X X O . O
 2 X . O . . O . .   2 O O O O O O O X
 3 X O . O X X X X   3 . . . . O X O X
 4 X . O X O . O .   4 . O . . O X O X
 5 X . O X O . O O   5 . . P P O X O X
 6 X O X X . O . O   6 . O . O . O . O
 7 X . X O . . O .   7 . O O . O . . .
 8 X O . O . O . O   8 O X X X . . . .
Enemy Fleet: XXXXX  Your Fleet: XXXXP
Game over!
Your fleet wins!
```

Sample Run 2 (board size N = 5)

Note: N = 5 is the minimum board size for this game to work properly. And it is normal to see that the fleet status bars are not aligning with the game boards well for N < 8.

```
Enter seed: 1↵
Play in game mode (Y/N)? n↵
Manually set up ships (Y/N)? n↵
Round 1:
Enemy Ocean   Your Ocean
   A B C D E     A B C D E
 1 C C C C C   1 C S . D .
 2 . . S S S   2 C S B D .
 3 D B B B B   3 C S B D .
 4 D . . . P   4 C . B P P
 5 D . . . P   5 C . B . .
Enemy Fleet: CBDSP  Your Fleet: CBDSP
```

```
Enter attack location: a2↵
Round 2:
Enemy Ocean   Your Ocean
   A B C D E     A B C D E
 1 C C C C C   1 C S . D .
 2 O . S S S   2 C S B D .
 3 D B B B B   3 C S B D .
 4 D . . . P   4 C . B P P
 5 D . . . P   5 C . B . .
Enemy Fleet: CBDSP  Your Fleet: CBDSP
Round 3:
Enemy Ocean   Your Ocean
   A B C D E     A B C D E
 1 C C C C C   1 C S . D .
 2 O . S S S   2 C S B D .
 3 D B B B B   3 C S B D .
 4 D . . . P   4 C . B P P
 5 D . . . P   5 C . X . .
Enemy Fleet: CBDSP  Your Fleet: CBDSP
Enter attack location: b2↵

... (too long, skipped)

Round 47:
Enemy Ocean   Your Ocean
   A B C D E     A B C D E
 1 X X X X X   1 X X O D .
 2 O O X X S   2 X X X X O
 3 X X X X B   3 X X X X O
 4 X O O O X   4 X O X X X
 5 X O O O X   5 X O X O O
Enemy Fleet: XBXSX  Your Fleet: XXDXX
Enter attack location: e2↵
Enemy Submarine sank!
Round 48:
Enemy Ocean   Your Ocean
   A B C D E     A B C D E
 1 X X X X X   1 X X O D .
 2 O O X X X   2 X X X X O
 3 X X X X B   3 X X X X O
 4 X O O O X   4 X O X X X
 5 X O O O X   5 X O X O O
Enemy Fleet: XBXXX  Your Fleet: XXDXX
Your Destroyer sank!
Enemy Ocean   Your Ocean
   A B C D E     A B C D E
 1 X X X X X   1 X X O X .
 2 O O X X X   2 X X X X O
 3 X X X X B   3 X X X X O
 4 X O O O X   4 X O X X X
 5 X O O O X   5 X O X O O
Enemy Fleet: XBXXX  Your Fleet: XXXXX
Game over!
Enemy fleet wins!
```

Submission and Marking

- Your program file names should be `battleshipboard.cpp`, `battleshipgame.cpp`, and `battleshipboard.h`. Submit the *three files* in Blackboard (https://blackboard.cuhk.edu.hk/). If you do not submit the `.h`, we shall assume that it is the same as the provided one.
- Insert *your name*, *student ID*, and *e-mail* as comments at the beginning of your source file.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- Your program should *include suitable comments as documentation*.
- ***Do NOT share your work to others*** and ***do NOT plagiarize.*** Both senders and plagiarizers shall be penalized.