Programming Assignment 6: Basic Game of Battleship

Assigned: Wednesday, March 11, 2014
Due: Monday, April 7, 2014 by midnight

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Apply and implement pointers 2-dimenional arrays
- Define and apply structs in C

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Apply and implement pointers in C
- Pass output parameters to functions
- Analyze a basic set of requirements and apply top-down design principles for a problem
- Apply repetition structures within an algorithm
- Construct while (), for (), or do-while () loops in C
- Compose C programs consisting of sequential, conditional, and iterative statements
- Eliminate redundancy within a program by applying loops and functions
- Create structure charts for a given problem
- Open and close files
- Read, write to, and update files
- Manipulate file handles
- Apply standard library functions: fopen (), fclose (), fscanf (), and fprintf ()
- Compose decision statements ("if" conditional statements)
- Create and utilize compound conditions
- Summarize topics from Hanly & Koffman Chapter 8 including:
    - What is an array?
    - Distinguishing between single dimensional and 2-dimentional arrays
    - What is an index?

III. Overview & Requirements:

Write a program that simulates the game of Battleship. The game will be completely text-based (see Sample Execution). Battleship is a two player Navy game. The objective of the game is to sink all ships in your enemy's fleet. The Player to sink his/her enemy's fleet first wins. Both players' fleets consist of 5 ships that are hidden from the enemy. Each ship may be differentiated by its "size" (besides the Cruiser and Submarine) or number of cells it expands on the game

board. The Carrier has 5 cells, Battleship has 4 cells, Cruiser has 3 cells, Submarine has 3 cells, and the Destroyer has 2 cells.

The program should be built such that the user is Player1 and the computer is Player2. Two boards exist within the game. Hint: each board should be implemented as a 2-dimensional array. Each 2-dimensional array should be 10 X 10. One represents Player1's board and one represents Player2's board. At the beginning of the game each Players' game board should be initialized to all '-' indicating that no ships have been placed on either board. Before the game starts, Player1 should have the option to either manually place each of the 5 ships in his/her fleet or to have them randomly placed on the board. If Player1 decides to place the ships manually, then he/she should be prompted to place the Carrier first, Battleship second, Cruiser third, Submarine fourth, and the Destroyer last. Note that ships cannot be placed diagonally on the board, they can only be placed vertically or horizontally. You program must check to see if the user tries to place a ship outside the boundaries of the board or on top of a ship that has already been placed. Each cell on the board that contains part of the ship must be indicated by 'c' for Carrier, 'b' for Battleship, 'r' for Cruiser, 's' for Submarine, or 'd' for Destroyer. For example, if the Carrier was placed then the board should contain 5 'c' s for each cell on the board that has a piece of the Carrier, etc. Once Player1's ships have been placed, Player2's ships must be randomly placed. Note that the placement of Player2's ships must be unknown. Thus, Player2's board will only display '-' in each cell after the placement of each ship. The program should randomly select Player1 or Player2 to go first.

Once it has been decided on which player goes first, the game starts. Whenever it's Player1's turn, a prompt should be displayed asking for a position to target (specifying where to "shoot") on the enemy's (Player2's) board (2-dimensional array). The position should be specified in terms of a row and a column on the board. The row and column should always be displayed along with the board. If the position specified happens to hit a ship, then a '*' should replace the '-' on Player2's board. If the positioned specified misses any one of the ships in the fleet, then a 'm' should replace the '-' on Player2's board. Note that from turn-to-turn each player should NOT be allowed to enter the same position. Also, between turns clear the screen (system("cls")). In one turn, a player can only take one shot at the enemy's (other player's) fleet. When Player2 takes a shot at Player1's board, each hit should be specified with a '*' and each miss with a 'm' on Player1's board. The game is over win Player1 or Player2 has sunk all of the ships in the fleet of the enemy.

For each move made by Player1 and Player2, the results should be echoed to a file called "battleship.log". In this file, you should log the targeted position by each player on each move and whether it was a hit on one of the ships in the fleet. Also, if one of the ships happens to sink, then note this in the log file. For more information about the rules of Battleship visit: Rules of Battleship.

At the end of the game, Player1's and Player2's statistics should be written to "battleship.log". The stats include total number of hits, total number of misses, total number of shots, hits to misses ratio (as a percentage), and won or lost the game. Note that the statistics should be placed into a structure called Stats. You need two variables of type Stats, one for Player1 and one for Player2. Once the game has ended you should write the contents of each struct variable to the "battleship.log" file.

Functional Decomposition

First step is to draw a structure chart to help you understand the decomposition of functions for this program. Remember to start with the overall problem and break it down into inputs, computations, and outputs. One possible functional decomposition includes the following **(Note: you are NOT required to apply these functions in your program!)**:

- Create a function `welcome_screen()` that displays an initial program welcome message along with the rules of Battleship.
- Create a function `initialize_game_board()` that sets each cell in a game board to '-'.
- Create a function `select_who_starts_first()` that determines if Player1 or Player2 goes first in the game.
- Create a function `manually_place_ships_on_board()` that allows the user to place each of the 5 types of ships on his/her game board.
- Create a function `randomly_place_ships_on_board()` that randomly places the 5 types of ships on a given board.
- Create a function `check_shot()` that determines if the shot taken was a hit or a miss.
- Create a function `is_winner()` that determines if a winner exists.
- Create a function `update_board()` that updates the board every time a shot is taken. '*' indicates a hit and 'm' indicates a miss.
- Create a function `display_board()` that displays a board to the screen. Note that Player1's board should be displayed differently than Player2's board (see above).
  Hint: pass in a flag (int) that stores whether or not you just passed in Player1's or Player2's board.
  Then perform the different logic for Player1's board versus Player2's board.
- Create a function `output_current_move()` that writes the position of the shot taken by the current player to the log file.
  It also writes whether or not it was a hit, miss, and if the ship was sunk.
- Create a function `check_if_sunk_ship()` that determines if a ship was sunk.
- Create a function `output_stats()` that writes the statistics collected on each player to the log file.
- Other functions that you think are necessary!
- A main function that does the following:

  - Opens an output file battleship.log for writing;
  - Simulates the game of Battleship
  - Outputs data to logfile
  - Outputs stats to logfile
  - Closes logfile

Sample Execution

The following sample session demonstrates how your program should work (user input is shown in bold).

***** Welcome to Battleship! *****

Rules of the Game:

1. This is a two player game.

2. Player1 is you and Player2 is the computer.

3. Etc. (You need to list the rest of the rules here.)

Hit enter to start the game!

Enter

(clear the screen)

Please select from the following menu:

1. Enter positions of ships manually.

2. Allow the program to randomly select positions of ships.

1

Please enter the five cells to place the Carrier across:

0 2 0 3 0 4 0 5 0 6

Please enter the four cells to place the Battleship across:

3 4 4 4 5 4 6 4

Etc.....

Player2 (Computer's) board has been generated.

Player1 has been randomly selected to go first.

Player1's Board:

```
   0 1 2 3 4 5 6 7 8 9
0  - - c c c c c - - -
1  d d - - - - - - - -
2  - - - - - - - - - s
3  - - - - b - - - - s
4  - - - - b - - - - s
5  - - - - b - - - - -
6  - - - - b - - - - -
7  - - - r r r - - - -
```

8 - - - - - - - - - -

9 - - - - - - - - - -

Player2's Board:

0 1 2 3 4 5 6 7 8 9

0 - - - - - - - - - -

1 - - - - - - - - - -

2 - - - - - - - - - -

3 - - - - - - - - - -

4 - - - - - - - - - -

5 - - - - - - - - - -

6 - - - - - - - - - -

7 - - - - - - - - - -

8 - - - - - - - - - -

9 - - - - - - - - - -

Enter a target: **2 3**

(clear screen)

2,3 is a hit!

Player1's Board:

0 1 2 3 4 5 6 7 8 9

0 - - c c c c c - - -

1 d d - - - - - - - -

2 - - - - - - - - - s

3 - - - - b - - - - s

4 - - - - b - - - - s

5 - - - - b - - - - -

6 - - - - b - - - - -

7 - - - r r r - - - -

8 - - - - - - - - - -

9 - - - - - - - - - -

Player2's Board:

0 1 2 3 4 5 6 7 8 9

0 - - - - - - - - - -

1 - - - - - - - - - -

2 - - - * - - - - - -

3 - - - - - - - - - -

4 - - - - - - - - - -

5 - - - - - - - - - -

6 - - - - - - - - - -

7 - - - - - - - - - -

8 - - - - - - - - - -

9 - - - - - - - - - -

Player selects: 9 9

9,9 is a miss!

Hit enter to continue!

Enter

(clear screen)

Player1's Board:

0 1 2 3 4 5 6 7 8 9

0 - - c c c c c - - -

1 d d - - - - - - - -

2 - - - - - - - - - s

3 - - - - b - - - - s

4 - - - - b - - - - s

5 - - - - b - - - - -

6 - - - - b - - - - -

7 - - - r r r - - - -

8 - - - - - - - - - -

9 - - - - - - - - - m

Player2's Board:

```
   0 1 2 3 4 5 6 7 8 9

0 - - - - - - - - - -

1 - - - - - - - - - -

2 - - - * - - - - - -

3 - - - - - - - - - -

4 - - - - - - - - - -

5 - - - - - - - - - -

6 - - - - - - - - - -

7 - - - - - - - - - -

8 - - - - - - - - -

9 - - - - - - - - -
```

Etc...

Player1 Wins!

Statistics outputted to logfile successfully!

IV. Submitting Assignments:

1. Using the Angel tool https://lms.wsu.edu submit your assignment to your TA. You will "drop" your solution into the provided "Homework Submissions" Drop Box under the "Lessons" tab. You must upload your solutions as <your last name>_pa6.zip by the due date and time.
2. Your .zip file should contain one project workspace. Your project must have at least one header file (a .h file), two C source files (which must be .c files), and project workspace. Delete the debug folder before you zip the project folder.
3. Your project must build properly. The most points an assignment can receive if it does not build properly is 65 out of 100.

V. Grading Guidelines:

This assignment is worth 100 points. Your assignment will be evaluated based on a successful

compilation and adherence to the program requirements. We will grade according to the following criteria:

- 95 pts for adherence to the instructions stated above (see above)
- 5 pts for appropriate top-down design of functions and good style