ECE 175 Computer Programming for Engineering Applications

Final Project: *Battleship*

Due Date: Wednesday November, 29th 2017 at 1PM, via D2L dropbox

See Page 4 for the late submission policy

Administrative details:

- The project is to be worked in a team of two students (maximum). You have the option to work by yourself. **Sign up for your team (groups of 2 or by yourself (work alone))**using the sign-up link below by 5 pm on Thursday November 9, 2017. After this date, you cannot change your team.
  https://docs.google.com/spreadsheets/d/1qphgzm5HRqTMWhrJFZTUzBC7uw2j_LqF4bkhP2An88A/edit?usp=sharing

- The final project demo will start at 2 pm on Wed November 29 in the lab. Use the sign-up sheet to choose your demo time. The demo sign up link will be available about a week before the project due date. At the demo/check-off time, your team (both students MUST be present) will be asked various questions (similar to your lab check-off but in more details). The person who does NOT show up for demo gets 0 point for the project. It is your or your team's responsibility to demo your project to the TAs/ULAs. Submitting code without demo in person results in 0 points.
- Honor code: Your team is expected to write and submit your own code. You may ask others for advice, and, in general, discuss the project, but your team should WRITE YOUR OWN CODE! If any part of the code submitted by different teams is identical, ALL involved parties will receive zero credit on the entire project and one letter reduction in their final grade. This policy will be very aggressively enforced. ALL submitted code will be checked with a plagiarism detection tool (e.g., http://theory.stanford.edu/~aiken/moss/).

The Game of Battleship:

Your task is to develop a program that plays a game of Battleship. The game is intended to be played between one player and the computer.

Basic Rules for the Game:

The game is played on four grids, two for each player (however, since one of the players is the computer, the computer's two grids are hidden from view). The grids are 10x10 square. The individual squares in the grid are identified by letter and number. On the major grid the player arranges ships and records the shots by the opponent. On the minor grid the player records his/her own shots. Before play begins, each player secretly arranges their ships on their major grid. Each ship occupies a number of consecutive squares on the grid, arranged either horizontally, vertically or diagonally (we will ignore Pythagoras for this project). The number of squares for each ship is determined by the type of the ship. The ships cannot overlap (i.e. only one ship can occupy any given square on the grid). The types and numbers of ships allowed are the same for each player.

Although the player may decide their own ship types and sizes, here are some example ship types and sizes

Ship Type	Board Symbol	Ship Size
Aircraft Carrier	A	5
Battleship	B	4
Cruiser	C	3
Submarine	S	3
Patrol Boat	P	2
Kayak	K	1
Row boat	R	1

Requirements:

In particular, your program should satisfy the following requirements:

1. To begin the game the computer will ask how many ships each side is given, or optionally, provide a file name to read in the information.
2. The ships are represented as variables of type *ship* and **stored as a linked list. Using linked lists is a requirement for this assignment.**

```
typedef struct ship {
  char shiptype[20];// e.g. Aircraft Carrier,
  int  shipsize;    // e.g.                 5,
  int  *X_Location; // Dynamically allocated array of length shipsize.
  int  *Y_Location; // Dynamically allocated array of length shipsize.
  int  *X_Hit;      // Dynamically allocated array of length shipsize.
  int  *Y_Hit;      // Dynamically allocated array of length shipsize.
  struct ship *pt;  // pointer to next ship on the list
} ship;
```

You are allowed to add attributes to this definition, but not remove any.

3. For each ship added a new node is created in the linked list.
4. The player will be given the option to fill out the structure information at the beginning of the game, or read in the information from a file. The information to be entered is:
   - ship type
   - ship size
   - X-location
   - Y-location
   - Other attribute information of your choice.
5. The computer will place the same type, and number of ships, at locations of his own choosing.
   - Use a random X,Y location for the bow of the ship.
   - The remaining X,Y values must be in a horizontal, vertical, or diagonal sequence.
   - No X,Y values of the ship may be off grid. Your program must check for this and not allow players to break this rule.
6. Ship positions may not overlap.
7. The player is given the first turn to shoot, then the computer takes his turn. The game continues in rounds until all ships of one contestant are destroyed. The player with at least one surviving ship (i.e. one un-hit X,Y location on at least one ship) is declared the winner.
8. Once the game begins each shot is recorded. Both 10x10 grids for the human player are updated with each shot. When a ship is hit the attributes X_hit and Y_hit are updated. These attributes are used to update the 10x10 grids during play.
   - On the minor grid a record of each shot is kept, using different symbols for a hit/miss.
   - On the major grid a record of the computer's shots are recorded. Again, using different symbols for a hit/miss.
9. The computer will choose his shot locations randomly.
10. After each turn the score is updated and displayed. The scoreboard should display (at a minimum)
    - Each player's name.
    - The number of hits each player has scored.
    - The number of missed attempts for each player.
    - The number of sunk ships each player has scored.
11. Once a ship is destroyed the node must be removed from the linked list. However, even though the ship is removed from the linked list, the major and minor grids continue to display the results of the sunken ship.
12. Your *C* program must keep track of both player's information. Thus two linked lists are needed, one for each player.
13. At the end of the game the major grids of both player's are revealed.

Optional Features for Extra Credit:

1. **Patterned Search.** The computer does not shoot randomly but uses a search pattern to determine his next shot. Upon a hit the computer searches the immediate area until a ship is sunk.
2. **Sonar.** If a submarine is in the game then a player may choose to use sonar. Upon choosing to *ping*
   - A single X,Y location of an enemy ship is shown on the player's major grid using a sonar location symbol 'O'. This symbol remains on the player's major grid for 1 turn only.
   - Since using sonar reveals position information, the player using sonar gives up a single X, Y location of his/her submarine.
   - Sonar does not reveal the orientation of the ships. It merely reveals a single X, Y location per player.
   - A sunk submarine cannot ping so the option is removed if no submarine is present.
   - A submarine may ping only once per game.
   - A ping costs the player 1 turn.
   - Your *C* program displays the sonar option only when it is an available choice.
3. **Abandon Ship.** If a large ship (size 3 or greater) is about to be sunk (i.e. has only one remaining un-hit X,Y location) a player may choose to *Abandon Ship*. Upon abandoning a ship
   - The ship with the remaining X,Y location is forfeit/sunk; since an abandoned ship cannot participate in the battle.
   - As a result of the sunk ship it must be removed from the linked list.
   - The abandon ship turns into two kayaks. Two Kayak nodes, each of size 1, are added to the player's linked list.
   - The X,Y location of the Kayaks are randomly placed on the player's major grid.
   - If the random position generator chooses an X,Y location that has already been used then that Kayak is automatically eliminated. i.e. If the X,Y location is already occupied by another ship, or if the X,Y location has been previously targeted then the Kayak is eliminated.
   - The opponent is informed that a player has used his/her turn to abandon ship.
   - The abandon ship option may only be used once per game.

- Your C program displays the abandon ship option only when it is an available choice.

4. **Torpedo.** An undamaged submarine may choose to fire a torpedo. This option can only be used one time, per player, per game.
   - The torpedo can only exit the bow of the submarine. Thus, in order to code this option you will need to determine how to distinguish, and keep track of, the submarine's bow and stern.
   - The torpedo travels in the same direction as the orientation of the submarine.
   - The torpedo can travel up to 2 squares away from the end of the submarine.
   - If the torpedo encounters a ship within 2 squares away from the submarine then a hit is registered. Note: The torpedo may damage friendly ships as well.
   - If a player chooses to use their turn to fire a torpedo, then the message

     `*** TORPEDO AWAY ***`

     must flash on the screen at least three times. This message flashes also when the computer chooses to fire a torpedo.
   - As the torpedo moves, the screen is updated to show the torpedo's location. You are free to choose any symbol to represent the torpedo. If the torpedo does not score a hit the symbol disappears from the screen after it has traveled two squares away.
   - Like sonar, using a torpedo reveals position information. The player using the torpedo gives up a single X, Y location of his/her submarine.
   - Your C program displays the torpedo option only when it is an available choice.

Below is an example game between John and the computer. Each player has had five turns. John's major grid shows a Submarine, Kayak, Battleship, and an Aircraft Carrier (which has been hit twice). John's minor grid shows two missed attempts and three successful attempts. The score is updated and displayed after each turn.

```
    A     B     C     D     E     F     G     H     I     J
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
1 |     |     |  S  |     |     |     |     |     |     |     |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
2 |  X  |     |  S  |     |     |     |     |     |  K  |     |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
3 |     |     |  S  |     |     |     |     |     |     |     |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
4 |     |     |     |     |     |     |     |     |     |     |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
5 |     |     |     |  A  |     |     |     |     |     |     |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
6 |     |     |     |     |  H  |  X  |     |     |     |     |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
7 |     |     |     |     |  X  |  H  |     |     |     |     |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
8 |     |     |     |     |     |     |  A  |     |     |     |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
9 |     |     |     |     |     |     |     |  A  |     |     |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
  |     |     |     |     |     |     |     |     |     |     |
10|     |     |     |     |     |     |  B  |  B  |  B  |  B  |
  |     |     |     |     |     |     |     |     |     |     |
  -----------------------------------------------------------
```

```
    A   B   C   D   E   F   G   H   I   J
  -------------------------------------------
1 | X |   |   |   |   |   |   |   |   |   |
  -------------------------------------------
2 |   |   |   |   |   |   |   |   |   |   |
  -------------------------------------------
3 |   |   |   |   |   |   |   | X |   |   |
  -------------------------------------------
4 |   |   |   |   |   |   |   |   |   |   |
  -------------------------------------------
5 |   |   |   |   |   |   |   |   |   |   |
  -------------------------------------------
6 |   |   |   |   |   |   |   |   |   |   |
  -------------------------------------------
7 |   |   |   |   |   |   |   |   |   |   |
  -------------------------------------------
8 |   |   |   |   |   |   |   |   |   |   |
  -------------------------------------------
9 |   |   |   |   |   |   |   |   |   |   |
  -------------------------------------------
10|   | H | H | H |   |   |   |   |   |   |
  -------------------------------------------
```

```
John          Computer
 Hit:  3       Hit:  2
 Miss: 2       Miss: 3
 Sunk: 1       Sunk: 0
```

```
Enter in the location of your next shot: row number, column letter
```

Helpful hint. You should use more than one structure. Maybe a player structure as follows?

```
typedef struct player_ {
  char name[20];//players name
  int ShotsBoard[10][10];//Keep track of where the shots were taken
  int ShipsBoard[10][10];//Keep track of current ships
  int OrigShipsBoard[10][10];//Remember the original configuration
  int Hits;//Count how many hits
  int Miss;//Cont how many shots
} player;
```

Late submission policy (submit after 1 PM on Wed November 29, 2017):

by 1 PM on Thursday November 30, 2017	deduct 10%
by 1 PM on Friday December 1, 2017	deduct 20%
by 1 PM on Saturday December 2, 2017	deduct 30%
by 1 PM on Sunday December 3, 2017	deduct 40%
by 1 PM on Monday December 4, 2017	deduct 50%
by 1 PM on Tuesday December 5, 2017	deduct 60%
by 1 PM on Wednesday December 6, 2017	deduct 70%

The last date of final project demo is by 5PM on Wednesday December 6, 2017. After this time the project will NOT be accepted/graded. If you submit your code late, it is your team's responsibility to demo your project to the TAs/ULAs. Submitting code without demo in person results in 0 points.

Grading Rubric (including extra credit) is forthcoming.