CS 4323 Design and Implementation of Operating Systems I

Assignment 00: Full Marks 100
(Due Date: 09/01/2021, 11:59 PM CDT)

The objective of this assignment is to make you familiar with C programming language. We will be using C to study operating system concepts and to be able to complete the future assignments successfully, you will need to be familiar with the following C programming concepts:

- Basic data types such as int, float/double and char
- Familiarization with Arrays (Strings as array of char), Structures (Very important)
- Basic use of printf() to print text to the terminal and scanf(), fgets() to read from the keyboard
- Read and write from and to the text file
- Control structures: if-then-else, switch
- Loops: while, for and do-while
- Functions and function calls
- Pointers (Very important)
- Pointers to structures (Very important)
- Call by reference
- Dynamic memory allocation using malloc, calloc and free (Very important)
- Basic use of the #define directive
- Separate compilation of individual C files

So, please do not just focus on completing the assignment. You need to understand the concepts. In order to get the full credit on the assignment, you must fulfill all the specifications specified below.

All these assignments must be compiled and run on CSX machine. Logging on to the CSX account and other relevant information can be easily accessed in: http://www.cs.okstate.edu/loggingon.html

Game Description:

You are going to make a single-player menu-driven **Battleship** game in C. The game starts by calling a function menu(), which displays the option menu as follows:

1. Enter player information
2. Initialize the board
3. Populate the board
4. Play the game
5. Print the board
6. Score board
7. Exit

Once each option is completed, the game displays this main option menu again.

Before choosing any other option (except option 7), the player must select option 1, which would then allow the player to enter following information through the function playerInfo():

- Name of the player
- Country of the player
- Date of the play

After the player has entered this information, the game needs to display the option menu again.

The player can again choose option 1 and reenter his/her information. In that case, the latest information should replace any previous information for the rest of the game.

Before the player can proceed with the game, the board must be first initialized. This is done by calling the function initializeBoard(), which creates the board of default size 10-by-10 i.e. the board has 10 rows and 10 columns.

Only after the board is initialized, the player can choose option 3, which will populate the board with different types of battle ships. The game has following 5 types of battle ships:

- Carrier with 5 grids
- Battleship with 4 grids
- Cruiser with 3 grids
- Submarine with 3 grids
- Destroyer with 2 grids

The board is randomly populated with these 5 battle ships by function populateBoard(). Each of these battle ships have different size and they take different number of grids in the board. For example, carrier ship takes 5 grids on the board. Similarly, battleship takes 4 grids, cruiser ship and submarine take 3 grids and destroyer takes 2 grids.

Each of these ships can be placed only horizontally or vertically in the board as shown in fig (1). You cannot place these ships at an angle or diagonal as shown in fig (2).

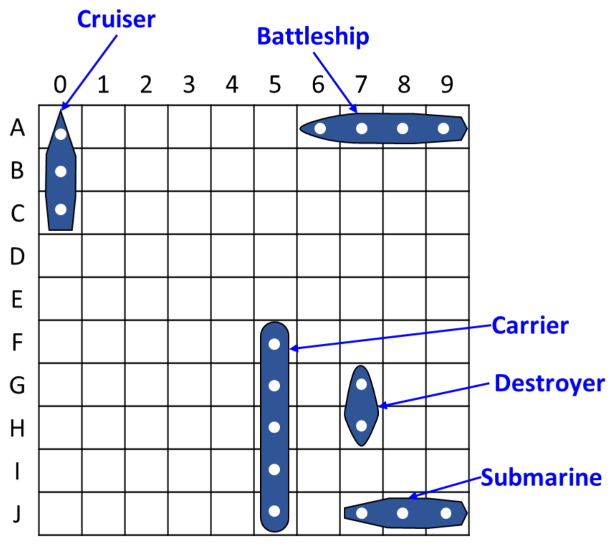

Fig. (1) Valid Configuration: Ships can be placed in the straight grid (either horizontally or vertically)

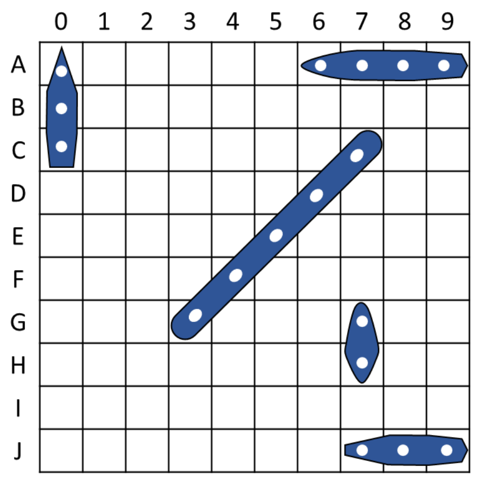

Fig. (2) Invalid Configuration: Ships cannot be placed in the diagonally

Once the board is populated, the main option menu is displayed. The player can then choose option 3, 4, 5, 6 or 7.

If the player chooses option 4, then the player starts playing the game. This option is described in the section “How to play the game” below. Once the player chooses option 4, the ships cannot change its position.

If the player wants to see how the board is populated in option 3, then the function printPopulatedBoard() in the option 5 allows the player to display the populated board. This step is very important for TA to check if your board is populated correctly or not. If the board is populated as shown in fig (1), then option 3 would display the board on the screen as shown below:

	0	1	2	3	4	5	6	7	8	9
A	X						X	X	X	X
B	X									
C	X									
D										
E										
F						X				
G						X		X		
H						X		X		
I						X				
J						X		X	X	X

Fig. (3) Board printed, for the populated board as shown in fig. (1)

The player can repopulate the board by choosing option 3. So, for each option 3, the ships position should be randomly selected.

The player can print the list of all players information and score stored in the file, using option 6. This option uses the function scoreBoard(…).This is discussed below.

The player can exit the game by choosing option 7 at any point while choosing option from the main option menu. The function exit() is used for this.

How to play the game:

Once the player choses option 4 in the option menu, the game starts by calling the function play(), which takes the last populated board in option 3, as one of the argument. The populated board, however, should not be displayed on the screen. Otherwise, the player will know the position of the ships and there is no point of playing the game. So, the board that should be displayed to the player should be as follows:

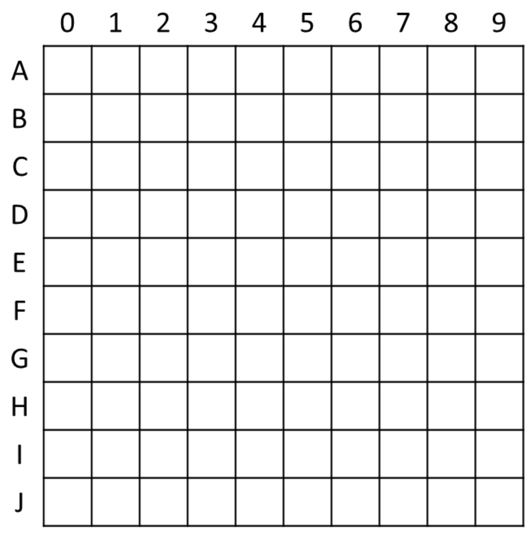

Fig. (4) Board at the start of the game

The game starts with the initial score of 100.

<u>Call your shot:</u>

The player will try to sink all those 5 ships by guessing where those ships are located in the board. So, the player will target the location by letter and number. Each target is a grid on the board, designated by the corresponding letter on the left side of the grid and the number on the top of the grid. For example, if the player guesses the ship is located at the intersection of row E and column 6, then the shot location is given by E6, as shown below.

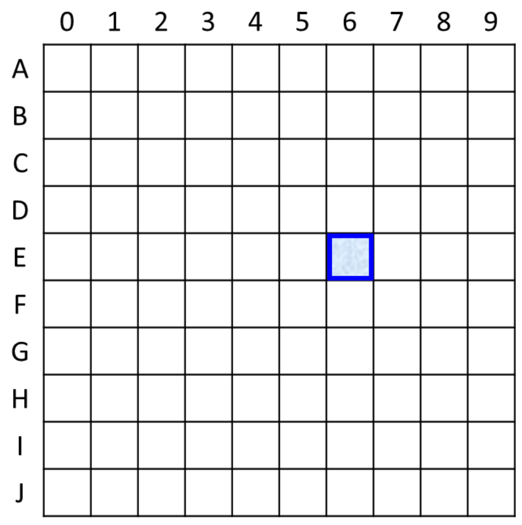

Fig. (5) Player guesses the target of the ship at the location E6

When you call a shot, you will be notified whether it's a hit or miss, by updating the table.

<u>The shot is a miss:</u>

If you call a shot location that is not occupied by a ship, then your shot is a miss. For example, for the above shot E6, you have encountered a miss. Each miss is designed by a letter M on the board, as shown below. This board will be printed on the screen.

	0	1	2	3	4	5	6	7	8	9
A										
B										
C										
D										
E							M			
F										
G										
H										
I										
J										

Fig. (6) The shot is a miss.

For each missed shot, the score decreases by 1.

<u>The shot is a hit:</u>

If you call a shot location that is occupied by a ship, then your shot is a hit. For example, if the shot location is J9, then it is a hit, which is designated by letter H on the board, as shown below. This board will be printed on the screen

	0	1	2	3	4	5	6	7	8	9
A										
B										
C										
D										
E										
F										
G										
H										
I										
J									H	

Fig. (7) The shot is a hit.

For each hit shot, the score does not decrease. Also, for each hit, the program needs to specify what type of ship you have hit. For the above example, it is a submarine.

You need to continue the game in this manner, until you sink all the ship.

When will the ship sink:

Once all the grids occupied by the ship is hit, then that ship will be sunk. For example, to sink submarine, you have to hit location J7, J8 and J9. Once the ship is sunk, the program needs to specify the message saying that the particular ship is sunk. In this case, the message would be:

Submarine is sunk.

Once all the ships have sunk, the game is over. The program then displays the final score. This score needs to be saved in the file along with the player information (name, country and date), by calling the function save(). Once this information is saved on the file, the program displays the main option menu. If another player plays the game, then this new player information and the corresponding score should be appended at the next line on the same file.

However, you need to consider the following two cases, while saving the player's information and score in the file:

- If the player chooses option 5, before playing the game, then the player information and the score should not be recorded in the file. This would lead to the cheating in the game, as the player already knows the position of the ships and can easily make the highest score.
- If the player has first chosen option 5, then option 3 and finally option 4, then the player information and player's score should be recorded in the file. In this case, the player does not know the ships' position in the board. So, there is no cheating in this case.

Programming requirements:

Your program should extensively use the following concepts:

1) Files should be used to store the player's information and the score.

2) The program should have at least the following functions:
    a. main(...): calls all other functions
    b. menu(...): displays all the options in the game
    c. playerInfo(...): enters player's information
    d. initializeBoard(...): creates an empty board of size 10-by-10
    e. populateBoard(...): randomly populates the board with 5 battle ships.
    f. printPopulatedBoard(...): print the populated game
    g. play(…): starts the game
    h. save(…): save the player information and score in the file
    i. scoreBoard(…): Prints the score board from the file
    j. exit(…): exits the game

Note: Each function should be in separate C file.

3) You need to use dynamic memory allocation to create a 2D array to create the initial state of the game. You need to pass the array via pointers to all functions.

   Note: Whenever dynamic memory is allocated, you need to free that space before exiting the program.

4) Use Structure to contain play information. The fields of the structure should account for following information:
   a. Player's name
   b. Country
   c. Date
   d. Score

   This structure should be used for all file operation i.e. to write into the file and read from the file. You need to use pointer to the structure to display the game summary.

Grading Rubrics:

The grading will be as follows:

• Correct working of menu options ○ Failure to call functions from main() will lead a penalty of -3 points	[5 Points]
• Correct working of playerInfo() ○ Not using or incorrect use of structure for player's information will lead to a penalty of 3 points	[5 Points]
• Correct working of initializeBoard() ○ Not using or incorrect use of dynamic array (using malloc) to create the board will lead to a penalty of 5 points ○ Incorrect or failure to deallocate memory will lead to a penalty of 1 point	[10 Points]
• Correct working of populateBoard() ○ Not using pointer or incorrect usage of pointer in the function parameter will lead to a penalty of 5 points	[15 Points]
• Correct working of printPopulatedBoard() ○ Not using pointer or incorrect usage of pointer in the function parameter will lead to a penalty of 5 points ○ Incorrect or not using pointer to print the board will lead to a penalty of 5 points	[10 Points]
• Correct working of game play() ○ Incorrect working of hit scenario lead to a penalty of 10 points ○ Incorrect working of miss scenario lead to a penalty of 10 points ○ Incorrect working of ship sink scenario to a penalty of 5 points ○ No display of which ship being hit or/and sunk will lead to a penalty of 2.5/5 points	[30 Points]
• Correct working of save() ○ Incorrect or not using file operation (writing) with structure or/and pointer will lead to a penalty of 5/7.5 points	[10 Points]
• Correct working of scoreboard() ○ Incorrect or not using file operation (reading) structure or/and pointer will lead to a penalty of 5/7.5 points	[10 Points]
• Correct working of exit()	[5 Points]

Note:

- Failure to follow standard programming practices will lead to points deduction, even if the program is running correctly. Some of the common places where you could lose points are:
  - Program not compiling successfully: -20 points
    - The TA will run the program using the input file state.txt in CSX machine.
  - No comments on code (excluding function): -5 points
  - No comments on function: -5 points
  - Not writing meaningful identifiers: -5 points
  - Failure to submit files as specified: -10 points

Submission Guidelines:

- You need to submit all the required C file with an extension .c. For example:
    - assignment00_lastName_firstName_ main.c
    - assignment00_lastName_firstName_ playerInfo.c

- All the C files should be submitted in the pdf format as well. Please make sure that you do not screen shot any code or save the program in the image form. All the codes need to be copied and pasted in the text form. For example:
    - assignment00_lastName_firstName_ main.pdf
    - assignment00_lastName_firstName_ playerInfo.pdf

- You need to include readMe.txt file which should include how to run your program.
    - readMe.txt

- In the assignment00_lastName_firstName_main.c file, use the following header information:
    - Author Name:
    - Email: <Use only official email address>
    - Date:
    - Program Description:
- Use comments throughout your program.

- Each function should be properly commented:
    - Mention each argument type, purpose
    - Function description
    - Return type of the function