Assignment 6:

Mine or battle!

- Design Due: Tuesday, June 4, 2019, 11:59pm
- Design Reviews Due: Thursday, June 6 2019, 11:59pm
- Code Due: Sunday, June 9, 2019, 11:59pm
- Code Reviews: *No code reviews for this assignment!*

Aim:

By working on this assignment you should be able to:

- Create and use dynamic arrays
- Create, manipulate and store input in c-style strings
- Accept and parse command-line arguments
- Implement recursion

About this Assignment:

For this assignment, you will design and implement a game. You can pick between minesweeper or battleship. If you turn in both, you will get whichever scores the highest points. If you submit the design for one option, but code for another, you would lose all the points on the design.

Points:

The point distribution is as follows for any option that you choose:

- **Design for the problem as described in Design section: 50** points
- **Code** to implement the problem statement and requirements: **80** points.
- **Program Style & Comments** (as described in this doc) **: 20** points.
- **Extra Credit:** As described for each problem.

You will be graded without a demo for this assignment i.e. TAs will compile and execute your program for a grade and also evaluate your design in their own time without you being present.

Option A: Minesweeper

If you choose this option, name your code file as optiona_minesweeper.cpp and design file as optiona_minesweeper.pdf.

Problem statement:

This game was made popular by the Windows operating system in the early 1990s, and it has continued to be a pre-installed game until Windows 8.
You can:

- watch a video of how to play minesweeper on YouTube https://www.youtube.com/watch?v=Z0EAysRluJk
- OR read about it on Wikipedia: https://en.wikipedia.org/wiki/Microsoft_Minesweeper

The objective of the game is to open/reveal every cell on the board without detonating a mine. Every cell contains a number or a mine. The numbers tell you how many mines surround the cell in the *horizontal*, *vertical*, and *diagonal* directions, e.g. at most, 8 possible mines surrounding a cell. You can flag a cell, which helps you remember where you think there is a mine to not detonate. If you select to open/reveal a cell that has a mine, then you automatically lose the game!

When the user runs the game program, they would pass command-line arguments telling the number of rows, columns and mines that they want in the game. The game would randomly distribute the mines on the board and setup the numbers describing how many adjacent mines are to each cell. After that, the game will display a blank board and ask the user to flag or open a cell on the board until the user selects a cell that contains a mine (losing the game) or selects all the cells free of mines (winning the game).

After the user wins or loses, you will ask the user if they want to play again. If so, you must get the number of rows, columns, and mines for the new game and create a new board for a new game.

When the user decides to quit the game, you would display their score as number of mines the user stepped on and number of mines cleared(the ones they didn't step on) taking into consideration all the games the user played since the program was run.

Tip: You can display row and column numbers on board to make it easier for the user to read/know which row and column to select!

Example run:

```
a.out -r 9 -c 9 -m 10
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------

Flag(1) or Open(2): 2
Enter row, col: 0 4
```

```
You stepped on a mine!!! You lose!
-------------------
|1|1|0|1|*|1|0|0|0|
-------------------
|*|2|1|2|2|2|0|0|0|
-------------------
|2|*|2|2|*|1|0|0|0|
-------------------
|1|1|3|*|3|1|0|0|0|
-------------------
|1|1|3|*|2|0|0|0|0|
-------------------
|1|*|3|2|2|0|0|0|0|
-------------------
|1|1|2|*|1|0|0|0|0|
-------------------
|1|2|3|2|1|0|0|0|0|
-------------------
|1|*|*|1|0|0|0|0|0|
-------------------

Do you want to play again (1-yes, 2-no): 1
How many rows, cols? 9 9
How many mines? 10
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
```

```
Flag(1) or Open(2): 2
Enter row, col: 0 0
-------------------
|1| | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
Flag(1) or Open(2): 2
Enter row, col: 8 0
-------------------
|1| | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
| | | | | | | | | |
-------------------
|1| | | | | | | | |
-------------------
```

```
Flag(1) or Open(2): 2
Enter row, col: 5 5
-------------------
|1| | | | |1|0|0|0|
-------------------
| | | | | |2|0|0|0|
-------------------
| | | | | |1|0|0|0|
-------------------
| | | | |3|1|0|0|0|
-------------------
| | | | |2|0|0|0|0|
-------------------
| | | | |2|0|0|0|0|
-------------------
| | | | |1|0|0|0|0|
-------------------
| | | |2|1|0|0|0|0|
-------------------
|1| | |1|0|0|0|0|0|
-------------------

Flag(1) or Open(2): 1
Enter row, col: 6 3
-------------------
|1| | | | |1|0|0|0|
-------------------
| | | | | |2|0|0|0|
-------------------
| | | | | |1|0|0|0|
-------------------
| | | | |3|1|0|0|0|
-------------------
| | | | |2|0|0|0|0|
-------------------
| | | | |2|0|0|0|0|
-------------------
| | | |!|1|0|0|0|0|
-------------------
| | | |2|1|0|0|0|0|
-------------------
|1| | |1|0|0|0|0|0|
-------------------
```

```
...
Flag(1) or Open(2): 2
Enter row, col: 7 1
-------------------
|1|1|0|1|!|1|0|0|0|
-------------------
|!|2|1|2|2|2|0|0|0|
-------------------
|2|!|2|2|!|1|0|0|0|
-------------------
|1|1|3|!|3|1|0|0|0|
-------------------
|1|1|3|!|2|0|0|0|0|
-------------------
|1|!|3|2|2|0|0|0|0|
-------------------
|1|1|2|!|1|0|0|0|0|
-------------------
|1|2| |2|1|0|0|0|0|
-------------------
|1|!|!|1|0|0|0|0|0|
-------------------

Flag(1) or Open(2): 2
Enter row, col: 7 2
```

```
Congratulations!!!
-------------------
|1|1|0|1|*|1|0|0|0|
-------------------
|*|2|1|2|2|2|0|0|0|
-------------------
|2|*|2|2|*|1|0|0|0|
-------------------
|1|1|3|*|3|1|0|0|0|
-------------------
|1|1|3|*|2|0|0|0|0|
-------------------
|1|*|3|2|2|0|0|0|0|
-------------------
|1|1|2|*|1|0|0|0|0|
-------------------
|1|2|3|2|1|0|0|0|0|
-------------------
|1|*|*|1|0|0|0|0|0|
-------------------

Do you want to play again (1-yes, 2-no): 2

Total number of mines cleared: 9
Total number of mines you stepped on: 1
```

Requirements:

- You must provide a message telling which command line arguments are acceptable and how to pass them, if the user enters incorrect command-line arguments. Include an example!
- The user should be able to specify command line arguments in any order.
- You must not have functions over 15-20 lines long (-10 automatically)
- You must not use global variables (-10 automatically)
- After the user wins/loses, you must ask if they want to play again
    - If no, then end the game while displaying the total number of mines stepped and cleared for all the attempts the user played the game
    - If yes, then prompt for rows, cols, and mines for new game.
- You must not have any memory leaks (-10 automatically)
- You must detect these errors:
    - Invalid row or column to flag or open
    - Opening a cell that has already been opened

- Flagging a cell that has been already flagged

Extra Credit (10 points)

Recursively open all cells adjacent to a cell that has 0 mines:

- If the user selects a cell that has no mines surrounding it, then you will help the user by recursively opening all adjacent cells to empty cells, until you reach cells with a surround mine.

Design (50 points):

The more thought and time you give for designing, the less you will spend on debugging the buggy code!

To understand how to design, you can look at the Example of a good Design Document available on this Week's Canvas module.

Step 1: Understand and restate the problem (15 points)

Do you understand everything in the problem? List anything you do not fully understand, and make sure to ask a TA or Instructor about anything you do not understand. Based on your understanding, also list:

- What are the user inputs?
- What are the program outputs?
- What assumptions are you making about the user input?
- What are all the tasks and subtasks in this problem?

Step 2: The Design (20 points)

What does the overall picture of your program look like?

Draw a flowchart for your program which answers the following questions:
If you want, you can also list out all the steps in form of pseudocode.

- How do you create and store the the gameboard in the memory?
- How do you keep track of which cells the user has explored and which cells are still unexplored?
- How do you display the gameboard stored in the memory?
- How do you keep track of the mines adjacent to each cell - vertically, horizontally and diagonally?
- When do you read input from the user?
- What tasks are repeated? How long are they repeated?
- How do you implement the flag operation?
- How do you implement the clear operation?
- What command line arguments is the user passing?

Be very explicit and make sure your flowchart/steps list out all the details in your design.

Step 3: Program Testing (10 points)

List out **all** the inputs that you are accepting in your program.
For each of these, list out what are the good inputs, bad inputs and the maximum and minimum limits of the expected input.

Option B: Battleship

If you choose this option, name your code file as optionb_battleship.cpp and design file as optionb_battleship.pdf.

Problem statement:

You will write a program that plays the game Battleship. In this program, you will be graded on having functions, as well as the ability to play the game correctly.

The game is described at:
https://www.hasbro.com/common/instruct/Battleship.PDF

This is traditionally a 2-person game, where each player picks where to put their ships on a 10 x 10 matrix. Ships can touch each other, but can't both occupy the same spot. Your computer game will simulate this process by first asking player 1 where he/she wants to put the ships, and then asking player 2.

Each player is given the following ships, and each ship takes up a specific number of spots on the 10 x 10 grid:

- 1 Aircraft Carrier, 5 spots
- 1 Battleship, 4 spots
- 1 Destroyer, 3 spots
- 1 Submarine, 3 spots
- 1 Patrol Boat, 2 spots

After you determine where each player wants to put their ships, then you can prompt each player to choose a position on the opponent's board. If there is a boat in the corresponding position on the opponent's board, then it was a hit, and you can use whatever you want to symbolize that it was a hit, i.e. X or 1.

In the case of a hit, the other player must mark the hit on his setup board containing the ships. Whenever a ship is sunk, the game must announce to your opponent that their ship has sunk. If there isn't a boat in the opponent's position, then it is a miss, and you can use whatever you want to symbolize a miss on your board, i.e. N or 0.

The player who sinks all of their opponent's ships first is the winner!

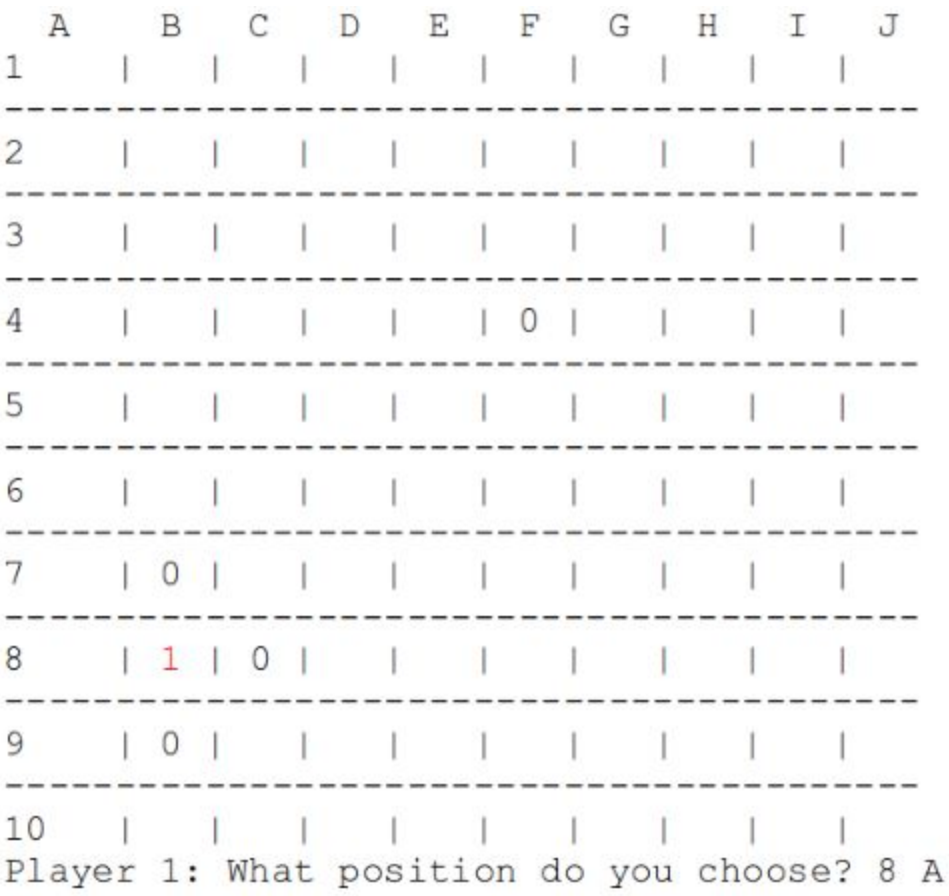
Tips

1. You can clear the screen after each player chooses their positions for their ships, by using `system("clear")` available by including the `stdlib.h` header file.
2. Some example functions you might want to include are:
    a. initialize_board(), which initializes the boards to spaces,
    b. determine_player_choice() that allows players to pick their spot on their opponents grid,
    c. fill_board(), which fills the board with the player's choice
    d. print_board() that prints the board to the screen after each user's turn,
    e. check_for_winner(), which checks to see if there is a winner, and a
    f. print_winner_results() that prints the results of the game to the screen.
3. You can print the X's or 1's red by using the following code:
   cout << "\033[22;31m X \033[01;37m"; \033[22;31m turns your text red, and \033[01;37m turns your text back to white.

Requirements:

- You must provide a message for each action and result of that action.
- Ships can touch each other, but can't both occupy the same spot.
- Ships can only be arranged horizontally or vertically on the board, and they can't hang over the grid!
- You should use functions.
- You must not have functions over 15-20 lines long (-10 automatically).
- You must not use global variables (-10 automatically).
- At the end of the game display for each player:
    - Ships still standing:
- You must not have any memory leaks (-10 automatically)
- You must detect and handle these invalid inputs by asking the user to enter again:
    - Invalid position to move the ship on:
        - If a ship is already at that position
        - If the position does not exist on the matrix
        - Any other possibility

Example run:


A B C D E F G H I J
1
2
3
4 0
5
6
7 0
8 1 0
9 0
10
Player 1: What position do you choose? 8 A

Hit!!!

You sunk my ship!!!

```
   A    B   C   D   E   F   G   H   I   J
1     |   |   |   |   |   |   |   |   |
-------------------------------------------
2     |   |   |   |   |   |   |   |   |
-------------------------------------------
3     |   |   |   | 0 |   |   |   |   |
-------------------------------------------
4     |   |   | 1 | 1 | 0 |   |   |   |
-------------------------------------------
5     |   |   |   | 0 |   |   |   |   |
-------------------------------------------
6     |   |   |   |   |   |   |   |   |
-------------------------------------------
7     |   |   |   |   |   |   |   |   |
-------------------------------------------
8     |   |   |   |   |   |   |   |   |
-------------------------------------------
9     |   |   |   |   |   |   |   |   |
-------------------------------------------
10    |   |   |   |   |   |   |   |   |
```

Player 2: What position do you choose? 4 C

```
Hit!!!
You sunk my ship!!!

  A    B   C   D   E   F   G   H   I   J
1    |   |   |   |   |   |   |   |   |
-----------------------------------------
2    |   |   |   |   |   |   |   |   |
-----------------------------------------
3    |   |   |   |   |   |   |   |   |
-----------------------------------------
4    |   |   |   |   | 0 |   |   |   |
-----------------------------------------
5    |   |   |   |   |   |   |   |   |
-----------------------------------------
6    |   |   |   |   |   |   |   |   |
-----------------------------------------
7    | 0 |   |   |   |   |   |   |   |
-----------------------------------------
8  1 | 1 | 0 |   |   |   |   |   |   |
-----------------------------------------
9    | 0 |   |   |   |   |   |   |   |
-----------------------------------------
10   |   |   |   |   |   |   |   |   |
Player 1: What position do you choose? 2 B

Miss!!!
```

```
Miss!!!

  A   B   C   D   E   F   G   H   I   J
1   |   |   |   |   |   |   |   |   |
-----------------------------------------
2   |   |   |   |   |   |   |   |   |
-----------------------------------------
3   |   |   |   | 0 |   |   |   |   |
-----------------------------------------
4   |   | 1 | 1 | 1 | 0 |   |   |   |
-----------------------------------------
5   |   |   |   | 0 |   |   |   |   |
-----------------------------------------
6   |   |   |   |   |   |   |   |   |
-----------------------------------------
7   |   |   |   |   |   |   |   |   |
-----------------------------------------
8   |   |   |   |   |   |   |   |   |
-----------------------------------------
9   |   |   |   |   |   |   |   |   |
-----------------------------------------
10  |   |   |   |   |   |   |   |   |
Player 2: What position do you choose? 7 H

Miss!!!
```

Extra Credit (10 points)

Implement a one player Battleship, where the user can play the computer. The algorithm for the computer play that you use doesn't have to be smart, it just needs to work!

Design (50 points):

The more thought and time you give for designing, the less you will spend on debugging the buggy code!

To understand how to design, you can look at the Example of a good Design Document available on this Week's Canvas module.

Step 1: Understand and restate the problem (15 points)

Do you understand everything in the problem? List anything you do not fully understand, and make sure to ask a TA or Instructor about anything you do not understand. Based on your understanding, also list:

- What are the user inputs?
- What are the program outputs?
- What assumptions are you making about the user input?
- What are all the tasks and subtasks in this problem?

Step 2: The Design (20 points)

What does the overall picture of your program look like?

Draw a flowchart for your program which answers the following questions:
If you want, you can also list out all the steps in form of pseudocode.

- How do you create and store the the game boards in the memory?
- How do you implement the hit operation? What happens to the ship which get hit? What happens to the ship which hits?
- How do you process the position that the user enters to actually move the ship?
- What does moving a ship mean for the single variables/arrays in your memory?
- How do you keep track of the various ships ?
- How do you display the gameboard stored in the memory?
- How do you keep track of the hits and misses?
- When do you read input from the user?
- What tasks are repeated? How long are they repeated?

Be very explicit and make sure your flowchart/steps list out all the details in your design.

Step 3: Program Testing (10 points)

List out **all** the inputs that you are accepting in your program.
For each of these, list out what are the good inputs, bad inputs and the maximum and minimum limits of the expected input.