1 Introduction

Your task for this assignment is to design, code (**in C89**), test and debug a modified battleships game.

In short, your assignment will:

- Extract the game settings and values needed from a file
- Interface with the user via a Terminal based menu
- Play an adapted game of battleships (single player)
- Be able to do conditional compilation
- Generate new input files.

2 Code Design

You must thoroughly document your code using C comments (`/* ... */`). For each function you define and each datatype you declare (e.g. using struct or typedef), place a comment immediately above it explaining its purpose, how it works, and how it relates to other functions and types. Collectively, these comments should explain your design. (They are worth substantial marks - see Section 8)

Your code should also be separated logically into multiple c and h files that overall have a single purpose. Marks will be deducted if monolithic functions and/or files are used.

3 Academic Integrity

This is an assessable task, and as such there are strict rules. You must not ask for or accept help from anyone else on completing the tasks. You must not show your work to another student enrolled in this unit who might gain unfair advantage from it. These things are considered plagiarism or collusion.

Staff can provide assistance in helping you understand the unit material in general, but nobody is allowed to help you solve the specific problems posed in this document. The purpose of

the assignment is for you to solve them on your own. Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry. In addition, your assignment submission may be analysed by systems to detect plagiarism and/or collusion.

4 Task Details

4.1 Input Files

Your program should accept two (2) command-line parameters that will be the filenames of the input files. For example:

```
[user@pc]$ ./battleships board.txt missiles.txt
```

Error checking is required on both files and in the case that either of the files is deemed to be invalid the program is to safely exit. Your program should **not** continue in the case of an error being detected but should print out a meaningful error to the user on **stderr**.

More details on each file are explained below.

4.1.1 Board File

The 1st filename is the name of the settings file for the game board. This file details the size of the board and the location of the ships. The first line is the size of the board and each proceeding line is a ship that belongs on the board. The format is as follows:

```
<width>,<height>
<location> <direction> <length> <ship name>
<location> <direction> <length> <ship name>
...
```

Details on each field:

Width + Height :

What is it? : The size of the board in width and height

Restrictions : Has to be a positive Integer between 1 and 12 (inclusive)

Location :

What is it? : This is the location of the front (head) of the ship

Restrictions : Has to be within the board size

Direction :

What is it? : The direction that the ship is facing. All proceeding blocks will be placed behind it

Restrictions : Can only be the values N,S,E,W **and** n,s,e,w (case insensitive)

Length :

What is it? : The amount of Blocks (including the head) that the ship takes up.

Restrictions : Has to be a positive number. Has to fit within the board.

Ships Name :

What is it? : The name of the ship.

Restrictions : Can be any sequence of characters (as long as it has at least 1 character)

A full example of the board file and the expected game board:

```
5,4
D4 E 3 NullByte Sub
A1 N 3 SSASR Popoto
C2 W 2 CPN "Rogers" Steve
```

```
:) || A | B | C | D | E |
---++===+===+===+===+===+
 1 || 0 |   |   |   |   |
---++---+---+---+---+---+
 2 || 0 |   | 0 | 0 |   |
---++---+---+---+---+---+
 3 || 0 |   |   |   |   |
---++---+---+---+---+---+
 4 ||   | 0 | 0 | 0 |   |
---++---+---+---+---+---+
```

The layout of the board is the same as normal Battleships game. On the 'X' axis we have the values from A,B... and on the 'Y' axis the values from 1,2... (More details further down)

4.1.2 Missile File

The 2nd file that is to be read is the missiles file. This file will contain the list of missiles that are available to use for that game. While in a normal game of battleships you only having a missile that hits a single tile, this one is slightly different as we now have missiles that are special. In this game there are 4 types of missiles:

single : Hits a single tile

v-line : Hits an entire column (Vertical Line)

h-line : Hits an entire row (Horizontal Line)

splash : Hits a 3x3 square (centered on the selected tile)

These missiles are to be read from the file and put into a **generic linked list**.

Example:

```
single
splash
single
V-Line
h-line
```

```
Single
```

The missile can be in any order and are case-insensitive. If a line consists of anything but one of the above 4 then assume the file is invalid.

4.1.3 Mandatory Function Pointer

To demonstrate your understandings of function pointers, one is required to be used on the missiles. The goal is that a function pointer is assigned to each missile when it is read from the file and when it comes to playing the game the assigned function pointer can be used instead of having to check the missile type again.

4.2 Menu

After the 2 input files have been read and appropriately stored in memory it is time to have a menu for the user. This menu is to have the following options initially and is to be controlled via integer input:

1 Play the Game

2 List all missiles

0 Exit

4.3 Playing the Game

Now that the files have been read its time to play the game. The game will consist of a number of rounds and will finish when there are either no missiles left or all ships have been destroyed.

For each turn you must display the following:

- The board showing all shots (hits and misses) made. (Similar to above)
- The amount of missiles left (Not including the current one)
- The name of the currently loaded missile (for example "v-line")
- A prompt for the user to input the next coordinate to target

This part has many components and as such as been broken down to make it simplier.

4.3.1 Displaying the Board

Tile that hasnt been shot yet

X Tile that has been shot but nothing there

0 Tile that has been shot and contains a ship

Note: For the time being, for debugging purposes, you might want to print a tile that contains a ship that hasnt been shot with a different character

Example: of a board (same file as above)

```
:) || A | B | C | D | E |
---++===+===+===+===+===+
 1 || # | X | # | # | # |
---++---+---+---+---+---+
 2 || # | # | 0 | 0 | # |
---++---+---+---+---+---+
 3 || 0 | X | X | X | X |
---++---+---+---+---+---+
 4 || # | # | # | # | # |
---++---+---+---+---+---+

Missiles Left: 2
Current Missile: v-line

Enter Next Target:
```

Your board doesn't have to look *exactly* like this, however you must have the axis on both sides (top and left) and be able to differentiate between each cells.

Note: We will be adding colour later to make it look better

4.3.2 Entering Coordinates

The input that is expected from a user is in the following format "<Column><Row>" eg "B4"

To make it easier you can assume that the user always inputs a character followed by an integer, however you must still do some error checking on it.

- Check that you are within the correct range (row and column)
- Allow the use of lower case character eg 'b4'

4.3.3 Ships Destroyed

When a ship is completely destroyed (all tiles have been hit) you are to print a message to the user saying that a ship has been destroyed followed by the name of the ship.

4.3.4 Game Ending

As said before a game ends when the player runs out of missiles or when all ships are destroyed. When a game ends you are to go back to the main menu and allow the use to play the game again (Same board and missiles)

4.4 Add Some Colour

To make the board look nicer were going to print the tiles in different colours depending on what their status is.

For an example board:

```
:) || A | B | C | D | E |
---++===+===+===+===+===+
 1 || # | X | # | # | # |
---++---+---+---+---+---+
 2 || # | # | 0 | 0 | # |
---++---+---+---+---+---+
 3 || 0 | X | X | X | X |
---++---+---+---+---+---+
 4 || # | # | # | # | # |
---++---+---+---+---+---+
```

In order to print colour youre going to need the following macros:

RED "\033[0;31m"

GREEN "\033[0;32m"

BLUE "\033[1;34m"

RESET "\033[0m""

To use them you wrap your wrap your printf string with the colour you want to print followed by the RESET colour (turns the colours back to normal)

```
char ch = 'X'
printf("%s%c%s", RED, ch, RESET);
--- OR ---
printf(RED "%c" RESET, ch);
```

Heres another example if you want to add more colours to your program: Click here

4.5 Remove Some Colour (Conditional Compilation 1)

Now that we have colour lets create a way to disable it.

Your goal is to add a condition compilation called "MONO" that when compiled with your program no longer prints any colours.

4.6 Find Your Ships (Conditional Compilation 2)

Lets add another conditional compilation to help us find our ships.

Add another conditional compilation called "DEBUG". With this one defined your board is to now print the '#' in a magenta colour if a ship is there and hasnt been shot yet. For example:

MAGENTA "\033[1;35m"

```
:) || A | B | C | D | E |
---++===+===+===+===+===+
 1 || # | # | # | # | # |
---++---+---+---+---+---+
 2 || # | # | # | # | # |
---++---+---+---+---+---+
 3 || # | # | # | # | # |
---++---+---+---+---+---+
 4 || # | # | # | # | # |
---++---+---+---+---+---+
```

4.7 Need Help?

So by now you should be able to shoot missiles at the board and the tiles will change character and colour depending on what was there. However what if a player didnt know how to play the game.

The challenge with this part is to allow the user to type in "help" when they are prompted to enter a coordinate to shoot at. When entered you are to print a description of what that missile does. Remember though that you cant do any checks outside of FileIO on which missile it is.

4.8 Creating Files

Add another 2 options to your Menu

3 Create Board File

4 Create Missile File

The goal of these options is to allow the user to create new input files for the program. As such they must save to the same format as above. These functions should ask the user for all the required information and save them in a file that is named by the user.

How you decide to setup these function and get user input is completely up to you. And to test that your function works, just use the newly created file as the next input file.

5 Report

You must prepare a report that outlines your design and testing. Specifically:

1. For each file that you have, write a paragraph explaining the overall purpose of it. (Hint: If you cant explain in 1 simple paragraph then its doing to much)
2. Describe (in 1-2 paragraphs) how you implement the storing and using of the ships on the board and the board. Talk about any complications that you can into while designing this component.
3. Demonstrate that your program works and how to use it (only need to show valid input files), including:
    - The contents of the input files

- The command-line used to execute your program
- The output of the board as shown on the screen
- The input the user uses to interact with the menu.
- The input the user used to play the game.

Note: Demonstrating your program is very important as it shows us if your code is meant to be working or not.

Your report should be professionally presented, with appropriate headings, page numbers and a contents page. You will loose marks for poorly written/badly formatted reports!

6 Submission

Once complete you are to submit your entire assignment electronically, via Blackboard, before the submission deadline.

Your submission is to be a single .tar.gz file containing:

A declaration of originality – A complete declaration of originality is to be filled out and submitted as a PDF file.

Your implementation – including all your source code (.c files, .h files, makefile, etc...). Note that all pre-compiled code will be removed before marking.

Your report – A single PDF file containing everything mentioned in Section 5

You are responsible for ensuring that your submission is correct and not corrupted. You may make multiple submissions, but only your latest submission will be marked.

No extensions will be granted. If exceptional circumstances prevent you from submitting an assignment on time, contact the Unit Coordinator ASAP with relevant documentation. After the due date and time is too late.

Late submission policy, as per the Unit Outline, will be applied.

Warning: Ensure that the filetype of all submission are correct. If your report is a docx or txt file it will be treated as a **non-submission**. Same applies for your submission. If you submit a .zip, .7z, .rar, etc..., then you will receive zero (0) for the whole assignment.

7 Demonstration

You will be required to demonstrate your assignment to a tutor the week after the submission date during your practical time. Due to the current unforeseen circumstances that are currently on though, how we do demonstrations may change.

During the demonstration, you will just be asked to show us your code working and talk to us about what you did. The idea is to help us staff with using your assignment.

8 Mark Allocations

To Be Determined

End of Assignment