APS 105 — Computer Fundamentals
Lab #6: Battleship
Winter 2021

Important note: You must use the `submit` command to electronically submit your solution by the end of your lab session.

In this lab, you will write parts of a C program / application that plays a game of Battleship. By the end of this lab, you should be able to:

- Use two dimensional arrays.
- Use functions for efficiency.

1 Problem Statement

Your part of the program will be to create a function that sets up the boards for the game. In use one board will be set up according to user ("Player") input, another will be randomly generated for the opponent.

For our game, the board will be 9 squares down (rows denoted 1-9) and 11 squares wide (columns denoted 1-11). The game is played with two players, one of which will be the computer in our version. Each player places "ships" on non-overlapping areas, either vertically or horizontally on their board. Each player in turn calls out a square. If it is occupied by one of the other player's ships then it is a "hit" otherwise it is a "miss". Play continues until one player hits every square occupied by ships of their opponent.

We will play the game with one each of the following ships (one set per player):

Ship Name	Ship Size
Carrier	5
Battleship	4
Destroyer	3
Submarine	2
Patrol Boat	1

You can find out more about the game and example boards at

https://en.wikipedia.org/wiki/Battleship$_{(game)}$

The following are the requirements of the function you will write:

- Function will be able to populate a board either based on user input or using random placement. Which is done is selected by a parameter.
- When used: Inputs from the player for each ship size will be: Row, Column, Orientation. The Ship Size will be 1-5, Row will be 1-9, Column will be 1-11, Orientation will be 0 for horizontal (going to the right from the Row/Column position) or 1 (going down from the Row/Column position).
- Function must check that inputs are valid, but all inputs will be integer
- Function must not allow ship overlap or ships to extend past the edges of the board.

- Function to conform to the following declaration as given in the skeleton code provided:

```
void populateBoard(bool getUserInput,
    int board[BOARDROWS+1][BOARDCOLS+1]);
```

  where

  1. If getUserInput is true, populate the board using input from a player.
  2. If getUserInput is false, populate the board randomly.

  and board is the board to be populated. BOARDROWS is 9; BOARDCOLS is 11 (These are fixed values)

- Cells in the board will be assigned the following values: 0 if empty, otherwise the size of the ship covering the cell (so 3 if a Destroyer is covering that cell, since a Destroyer takes three cells in all and this is one of the three)

2 Examples

Please see the accompanying file of examples. The computer's playing is guided using the random numbers. See the note below on generating the computer board.

3 Notes

1. You can run your completed routines in the program shell provided to continue from the board generation into game play! Or you might want to program the whole thing – it's under 200 lines of code!! (But be sure to use the given game code for your submission.)
2. Suggestion: Make a chart of all the possible responses to any player input, including all the "problem" conditions, trace your code to make sure you've covered all these conditions, then test your code for each case.
3. You may want to play with the code given to make the development easier in the following ways. Make sure that you change the program back before you submit it!
   - Put a loop in to run your program repeatedly.
   - Cut out or comment out the part of the program to play the game so you only get the part that populates the board. This populating part is in main() until the comment

     ```
     // play starts here
     ```

     Note that you also can stop program execution at an input with a control-C.
   - You can temporarily set **dumpComputer=true;** (a boolean variable initialized to false) to cause the computer's board to be printed at the start of the game. You may be asked to do this by the TA. Be sure to remove this for testing and submission.

4 Grading / Submission and Automarking

In a file called `Lab6.c`, write your solution to the problem. There are ten (10) marks available in this lab.

Read the notes and other comments carefully to make sure your automarked version conforms to expectations.

The total of **10 marks** on this lab are marked in two different ways:

1. **By your TA, for 4 marks out of 10.** Once you are ready, show your program to your TA so that we can mark your program for style, and to ask you a few questions to test your understanding of what is happening.

   The TA will also ask you some questions to be sure that you understand the underlying concepts being exercised in this lab.

2. **By an auto-marking program for 6 marks out of 10**. You must submit all of your program files through the ECF computers for marking.

Long before you submit your program for marking, you should run the **exercise** program that compiles and runs your program and gives it sample inputs, and checks that the outputs are correct. Similar to previous labs you should run the following command:

```
/share/copy/aps105s/lab6/exercise
```

within the directory that contains your program.

This program will look for the file `Lab6.c` in your directory, compile it, and run it on a some of the test cases that will be used to mark your program automatically later. If there is anything wrong, the **exercise** program will report this to you, so read its output carefully, and fix the errors that it reports.

Once you have determined that your program is as correct as you can make it, then you must submit your program for auto-marking. This must be done by the end of your lab period as that is the due time. To do so, go into the directory containing your program and type the following command:

```
/share/copy/aps105s/lab6/submit
```

This command will re-run the exercise program to check that everything looks fine. If it finds a problem, it will ask you if you are sure that you want to submit. Note that you may submit your work as many times as you want prior to the deadline; only the most recent submission is marked.

The **exercise** program (and the **marker** program that you will run after the final deadline) will be looking for the exact letters as described in the output in this handout, including the capitalization. When you test your program using the exercise program, you will see that it is expecting the output to be exactly this, so you will have to use it to see if you have this output correct.

Important Note: You must submit your lab by the designated time and date. Late submissions will not be accepted, and you will receive a grade of zero.

You can also check to see if what you think you have submitted is actually there, for peace of mind, using the following command:

```
/share/copy/aps105s/lab6/viewsubmitted
```

This command will download into the directory you run it in, a copy of all of the files that have been submitted. If you already have files of that same name in your directory, these files will be renamed with a number added to the end of the filename.

---

5 After the Final Deadline — Obtaining Automark

Briefly after all lab sections have finished (after the end of the week), you will be able to run the automarker to determine the automarked fraction of your grade on the code you have submitted. To do so, run the following command:

```
/share/copy/aps105s/lab6/marker
```

This command will compile and run your code, and test it with all of the test cases used to determine the automark grade. You will be able to see those test cases output and what went right or wrong.