Computing Machinery I
Project
(40% of your final score)

Objective
You will create a simple single-player game both in C and ARM assembly. This is a two-part project.

Overview
This game is inspired by retro minesweeper game, but it is not the same. The game consists of a 2D board of hidden rewards or bombs. These are represented as floating point numbers, positive for rewards and negative for bombs. The board is randomly populated by these numbers. The player starts with a score of zero and a time duration on the clock. S/He chooses a cell to uncover. The value of the uncovered cell is added to the score. Hence, a bomb reduces the score and a reward increases the score. The game ends in one of three cases: when the score becomes less than or equal to zero (except for the very initial move), when the player uncovers every cell on the board, or the allocated time is up. There is no need for a graphical interface; however, graphical interfaces will be rewarded with bonus marks.

Details
The user starts the game by specifying a player name and the dimensions of the game board. These will be N×N grids; hence, it is sufficient for the user to enter the value of N. The minimum value of N is 5 and the maximum is 20.

```
./mygame.o minesweeper 5
```

The grid is then initialized with random floating numbers with two precision points, ensuring that no more than 20% of the cells are negative. These numbers should be non-zero with a maximum absolute value of 15. You must use bitwise arithmetic to calculate the modulus of the random numbers. The game board is then displayed. If you are using 5×5 grid and a text-based interface, it will look like the following:

```
XXXXX
XXXXX
XXXXX
XXXXX
XXXXX
Score: 0
Time: 60
```

The user then chooses a cell by entering its coordinates, such as:

```
Enter your move (x, y): 0 1
```

The countdown starts as soon as the coordinates of the initial move are read. The move is applied, and the resulting game state is shown again. For example:

```
Uncovered a reward of 10.23 points
X+XXX
XXXXX
XXXXX
XXXXX
XXXXX
Score: 10.34
Time: 58
```

A bomb is represented as a negative (–) symbol:

```
Bang!! you lost 8.11 points
X+XXX
XXXXX
XX–XX
XXXXX
XXXXX
Score: 2.23
Time: 58
```

In addition, the game utilizes surprise packs. These can double or half the score. You must use bitwise shifting to implement these changes. A nice surprise (double) is indicated by $ on the game board. Use ! for a bad surprise.

The timer should be initiated to a minute for the smallest value of 5 and it increases as N increases. It is left to you to decide on the actual values based on what you see reasonable to play the game. As a bonus, you can implement additional surprise packs in the form of extra time added to the remaining time. Use the symbol @ for these surprises.

The timer must be shown every time the board is displayed.

The user can choose at any time to exit the game. When the game ends with a win or a loss, the player's name, score, and time are recorded in a log file. A user can also ask to display the top *n* scores before or after any game, including player names and duration.

Modularity
Your code must divided into functions as appropriate. At a minimum, you must define the following functions (we are not showing all necessary arguments):

- `initializeGame(*board)`
- `randomNum(n,m,neg);` `n` and `m` are the lower and upper bounds for the random number; the generated number is negative if `neg` is `true`.
- `displayGame(*board)`
- `calculateScore()`, returns overall score
- `logScore()`
- `exitGame()`
- `displayTopScores(n)`

Part 1 (20% of your final score, due on June 1st @ 11:59 MST)
Create the game entirely in C.

Part 2 (20%, of your final score, due on the final exam day)
Re-create the same game entirely in ARMv8 assembly. You must use a text-based interface for Part 2. For this part, we suggest that you implement the game first without floating-point numbers. Then, you can add this feature at the end of the semester. If your code is modular enough. It should only require your to replace a few subroutines.

Submission
Submit your work to the appropriate dropbox on D2L. The TA will provide further submission instructions.

Late Submission Policy
Late submissions will be penalized as follows:
-12.5% for each late day or portion of a day for the first two days
-25% for each additional day or portion of a day after the first two days
Hence, no submissions will be accepted after 5 days (including weekend days) of the announced deadline.

Academic Misconduct
This project is to be done by individual students: your final submission must be your own original work. Teamwork is not allowed. Any similarities between submissions will be further investigated for academic misconduct. While you are encouraged to discuss the project with your colleagues, this must be limited to conceptual and design decisions. Code sharing by any means is prohibited, including *looking* at someone else's paper or screen. The submission of compiler generated assembly code is absolutely prohibited. Any re-used code of excess of 5 lines in C and 10 lines in assembly (10 assembly language instructions) must be cited and have its source acknowledged. Failure to credit the source will also result in a misconduct investigation.

D2L Marks
Marks posted on D2L are subject to change (up or down).

Computing Machinery I
Project Rubric

Student:_____

Item	Max Points	Points
Code compiles	5	
Code runs	5	
Game logic	20	
Use of floating-point numbers	10	
User interface (input validation, implementing all features)	10	
Random numbers	10	
Use of binary arithmetic (mod with random numbers, surprise packs)	5	
Timer	10	
Modularity and macros (macros: part 2)	10	
Command-line arguments	5	
Passing array parameters by reference	5	
Code readability (documentation)	5	
Time pack bonus	**10 (bonus)**	
Total Points	**100**	