EEEE1042 - **Coursework 2**

Total marks: 100

In this coursework assessment you are to create `.c` programs that accomplish the tasks described in the problem statement below. This coursework assessment is made available to the students from Moodle on Thursday Nov 25th and is due back uploaded on Moodle by 5pm, Friday Nov 17th. Please submit your programs on Moodle by the due date. This assessment is worth 10% of your final grade.

Marking Rubric

- 0-25%: Codes developed but did not compile properly.
- 26-50%: Codes developed and compiled, but not achieving expected result, programs are buggy or inefficient.
- 51-75%: Codes developed and compiled with correct bug-free and efficient results but modularity/clarity/commenting practices are not well followed.
- 76-100%: Codes developed and compiled using taught best practice standards, with clear, well-commented code and modularity rules being demonstrated/followed.

The codes you submit must be developed by yourself. Any codes which are **significantly** similar to each other, or to code found on the internet will be penalized as plagiarism. The comments in your code will be contributing evidence that the codes submitted are your own.

Problem statement

In this coursework, you are to design and build a program in C that is able to play tic-tac-toe. Tic-tac-toe is a simple game on a 3x3 board in which players take turns placing an `x` or an `o` onto anyone of the available (unoccupied) squares. A player wins by getting 3 of his pieces in a line, either horizontally, vertically or diagonally. Play alternates between the two players until one player wins, or all 9 squares are filled with no winner, in which case the game is a tie.

You are to tackle this coursework in stages:

1. In part 1, you will work on building a program that is able to print the tic-tac-toe board, accept legal moves from the user and determine when a player has won. In part 1, the user makes the moves for both sides as the computer does not yet have any intelligence to make a move. When you have completed part 1, the program will be able to serve as a tic-tac-toe board with 2 human players taking turns to make their moves, i.e. it operates in 2-player mode.

2. In part 2, you will work on modifying your program from part 1 such that it is able to make moves to play against the user. In this part you will add some capability to your program to make moves on the tic-tac-toe board. You will develop and implement two algorithms to make moves in this part. When you have completed part 2, the program will be able to play against a human player, i.e. it operates in a single-player mode.

3. In part 3, you will statistically compare the performance of the two algorithms developed in the previous part by comparing the #wins, #draws and #losses for each when pitted against the other. Then you will writeup your findings in a short report.

1 2-player mode

In the 2-player mode you should setup your program and, at a minimum, define functions that can

1. Initialize the tic-tac-toe board

2. Print the tic-tac-toe board

3. Clear the tic-tac-toe board

4. Check if a player has won

In the main function iterate over a for-loop that asks for a move from the user, checks it is legal and enters the move onto the tic-tac-toe board. It should then print the board and check whether a player has won, ending the loop if he has. The moves on the board should be a number from 1 to 9 according to the Figure 1: A sample game between two players

```
     |     |
  1  |  2  |  3
-----------------
  4  |  5  |  6
-----------------
  7  |  8  |  9
     |     |
```

Figure 1: The numbers representing each possible move by a player in the tic-tac-toe game.

using the program could look as in Figure 2

```
1 Enter move for Player o: 5
     |     |
     |     |
-----------------
     |  o  |
-----------------
     |     |
     |     |
```

```
2 Enter move for Player x: 3
     |     |
     |     |  x
-----------------
     |  o  |
-----------------
     |     |
     |     |
```

```
3 Enter move for Player o: 7
     |     |
     |     |  x
-----------------
     |  o  |
-----------------
  o  |     |
     |     |
```

```
4 Enter move for Player x: 6
     |     |
     |     |  x
-----------------
     |  o  |  x
-----------------
  o  |     |
     |     |
```

```
5 Enter move for Player o: 9
     |     |
     |     |  x
-----------------
     |  o  |  x
-----------------
  o  |     |  o
     |     |
```

```
6 Enter move for Player x: 8
     |     |
     |     |  x
-----------------
     |  o  |  x
-----------------
  o  |  x  |  o
     |     |
```

```
7 Enter move for Player o: 1
     |     |
  o  |     |  x
-----------------
     |  o  |  x
-----------------
  o  |  x  |  o
     |     |
```

```
Player o wins!
```

Figure 2: A sample two-player tic-tac-toe game output using the program

Save your program to the file: `coursework2_EEEE1042.1.c` and submit it to Moodle by the due date. For full marks be sure to adequately comment your code.

2 Single-player mode

In single-player mode, the computer must be able to play moves for the opponent. You should re-use the functions that you created in the 2-player mode, such as printing and checking for winner. Modify your program above such that play alternates between taking input from the user, and the computer making a move.

For the computer's moves (the AI), write the code such that there are 2 types:

1. Random AI: in this case the computer makes random (though legal) moves.
2. Smart AI: In the second scenario, the computer checks for 2 things in determining its move:
   (a) If the computer can immediately win on its current move, it plays the winning move ending the current game.
   (b) If the opponent is about to win on the next move, the computer sees this and blocks the opponent's winning move.
   (c) If neither of the above 2 scenarios occurs, the computer falls back to the random AI, and chooses a random legal move.

It should be clear that the second AI is going to perform better than the first, but if you've played tic-tac-toe before, you should know that the second AI is still not perfect. Save your program to the file: `coursework2_EEEE1042.2.c` and submit it to Moodle by the due date. For full marks be sure to adequately comment your code. Put extra time into commenting the AI portion of your code

3 Computer-vs-computer mode

In computer-vs-computer mode, we are going to let the computer play against itself and the human is only the observer. The computer will play very quiclky and you should modify your program to keep the statistics on how well the two sides perform. As we have two

AI's defined above you should collect the statistics (#wins/#draws/#losses) for each of the following 3 situations:

1. Smart AI vs Smart AI.
2. Smart AI vs Random AI.
3. Random AI vs Random AI.

Gather your statistics for a suitably large number of games. You should choose this number yourself based on your experimentation with your program, but it is recommended to be at least 1000. Write your findings up in a short (2-3 pages) document which reports what you did and the results of your experiments for the 3 scenarios above. Save your program to the file: `coursework2 EEEE1042.3.c` and submit it together with your report to Moodle by the due date, again being sure to have adequately commented your code. Please ensure your report is in `.pdf` format.