University of Leeds **School of Computing**
Procedural Programming (COMP1711)
2018-2019

Assignment 2

40 Marks

Submission Deadline:
3/12/2018 at 10:00 am

The main objectives of this assignment is to help you learn how to use arrays, structures, and functions.

This assignment is comprised of two tiers. Tier 1 is compulsory and must be completed by all students. If you satisfy all the requirements of Tier 1, you can earn the full mark of the project.

Tier 2 is optional, and you can choose to do it if you would like to stand out from the crowd. Tier 2 is not marked, but if you accomplish it correctly, your name will be written in the Distinction List on the Minerva page of the procedural programming module, and if your code is exemplary it will also be placed on Minerva for all your colleagues to see.

Tier 1 (Compulsory)

The Brief

You will write a C program for a game that enables two players to play the famous Tic-tac-toe game against each other.

The Details

The Problem

Tic-tac-toe, also known as noughts and crosses or Xs and Os, is a simple paper and pencil game for two players. A grid (matrix) of *nxn* cells is first drawn. Each player chooses one of two symbols (usually X or O) as his own. The two players then take turns in placing (writing) their symbols within the cells of the grid. The player who first succeeds in forming a complete horizontal, vertical, or diagonal line in their chosen symbol wins the game, as shown in the example below.

The most famous version of the game uses a 3x3 grid, but the game can be played using any size of grid. When the size of the grid is large, using paper and pencil has its problems. For a start, drawing the empty grid can be tedious. And, in many cases the players may get distracted and forget whose turn it is. Also, a player may form a line but this goes unnoticed for some time.

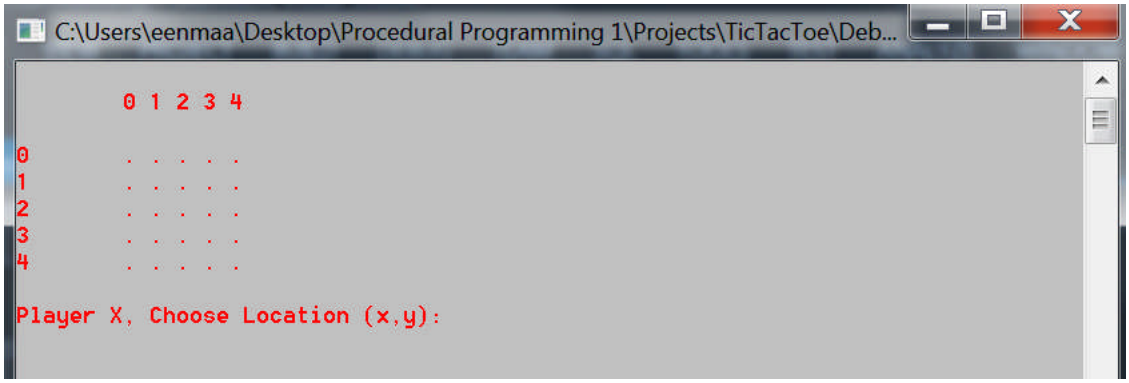
Using a computer can help people play this game easier, by drawing the empty grid, organizing turns, and detecting a win as soon as it occurs. The program also records all the moves made by the players during the game, and allows them to watch (play back) these moves once the game is finished.

The Task

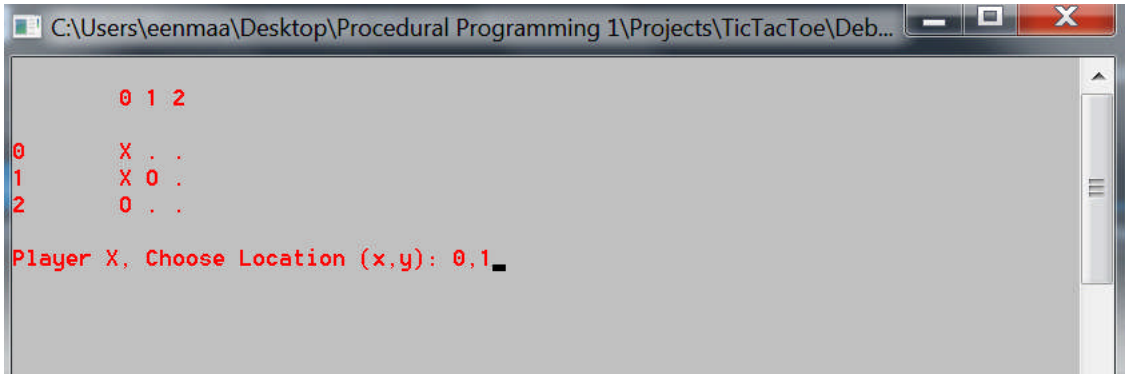
Write a C program that allows two persons to play Tic-tac-toe against each other. The program should support any size of grid from 3x3 to 10x10.

Program operation

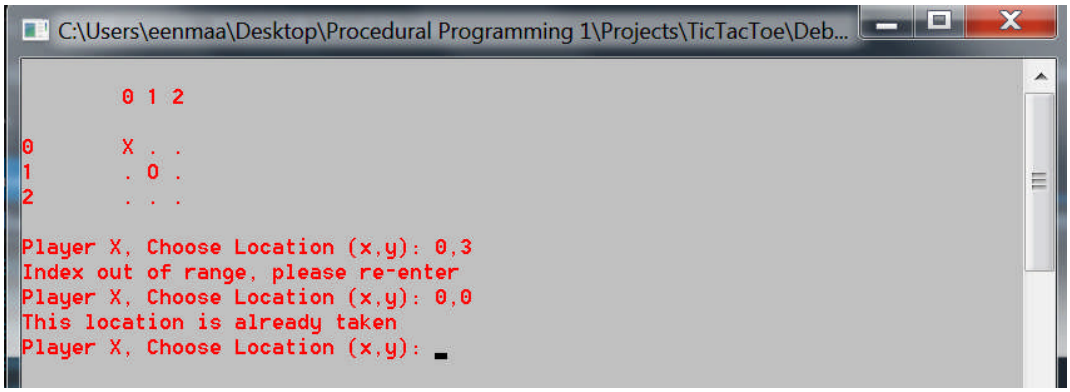
When the program starts, it prompts the user to enter the size of the grid (a number between 3 and 10). The program then displays the empty grid on the screen. Empty cells are shown as dots. To determine cell positions, the program also displays the row and column numbers (indices) of the cells as shown below:



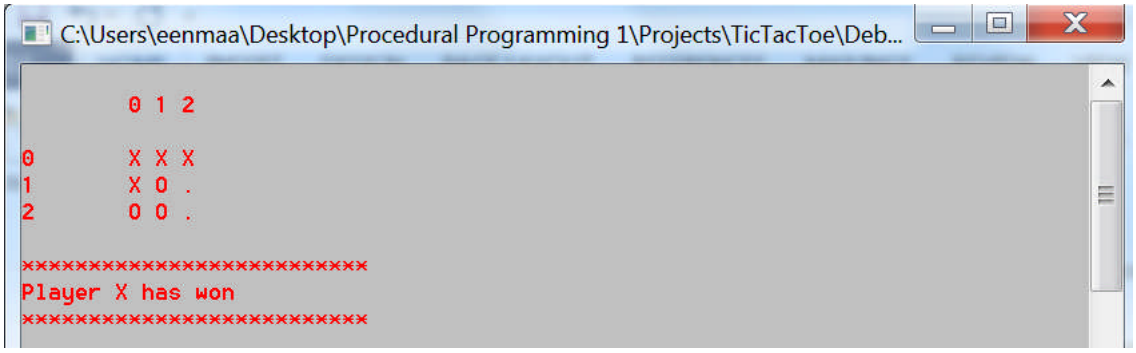
The program then alternately prompts the two players, called Player X and Player O, to place their symbols in the grid. The player enters the coordinates of the cell by giving its row and column numbers separated by a comma, as shown below:



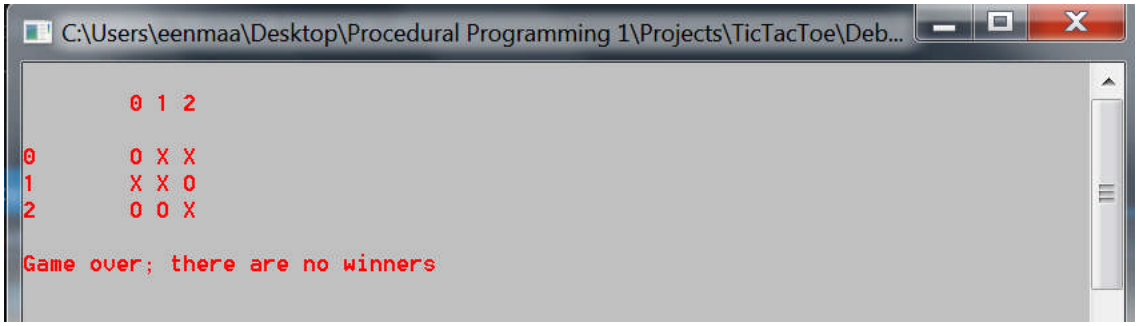
The program should check each input and make sure that the input coordinates are within the range of the grid, and that the intended cell is empty (i.e. a symbol is not being placed on top of an existing symbol), as shown below:



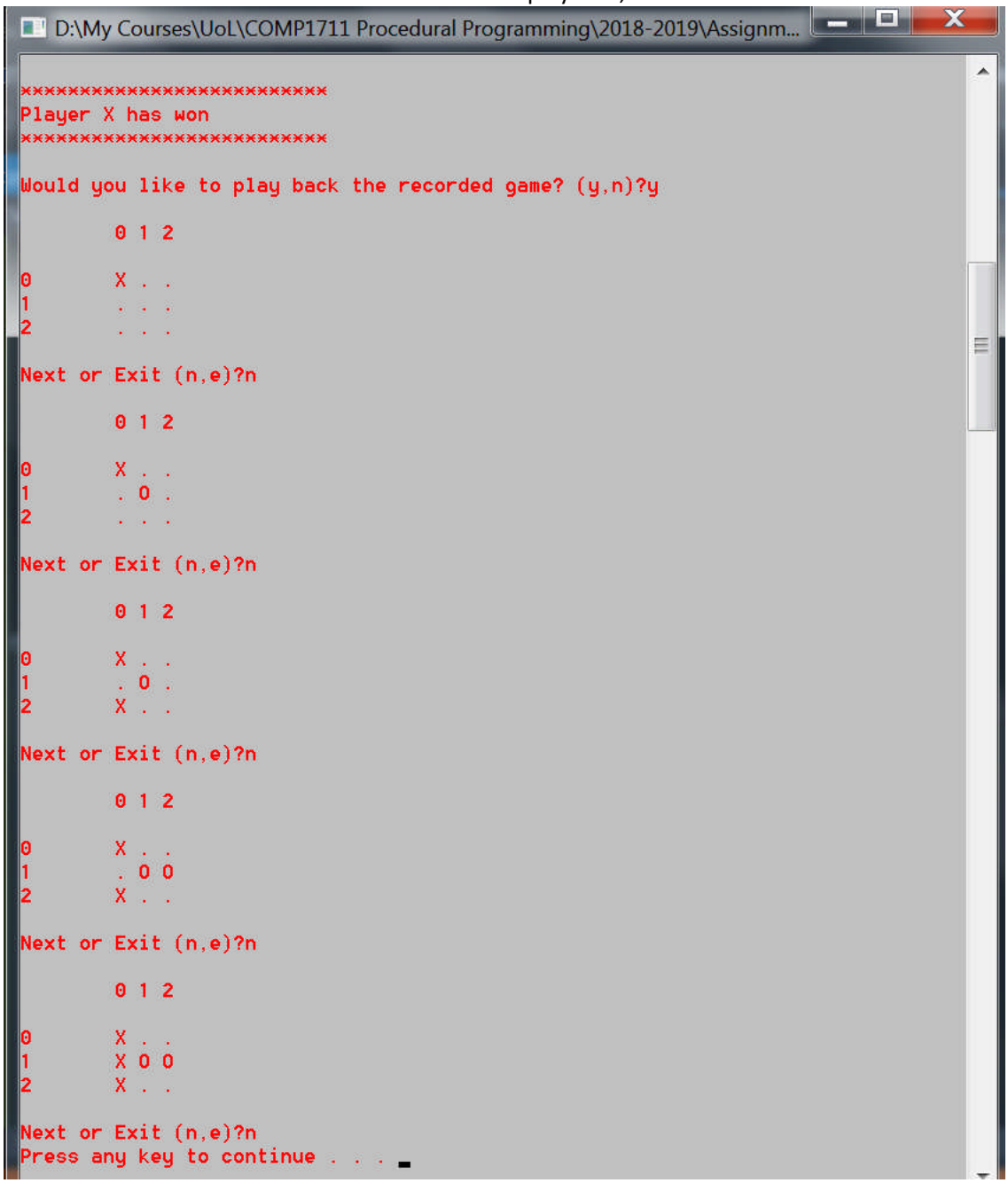
After each 'move' by any player, the program checks if this player has won (i.e. has made a horizontal, vertical, or diagonal line). And, if the player has won, the program declares this player as the winner and stops, as shown below.



If the grid becomes full and neither player has formed a line, the program declares a draw and stops, as shown below:



In all cases, once the game is over, the program asks the users if they want to watch the game, and if they respond with yes, the program displays the game's board after the first move, then pauses and asks the users if they want to view the board after the next move. This continues until all the moves were shown or the users decide to terminate the playback, as shown below:



Implementation Instructions

- Write the program in standard C. If you write your code in any other language, it will NOT be assessed and you will get a zero mark.
- Use the most appropriate data structure(s) to store and display game information.
- Use functions to avoid repeating similar code segments in the program, and to make your code well structured, clear, and concise.
- This is an individual project, and you are not supposed to work in groups or pairs with other students.
- Be aware that plagiarism in your code will earn you zero marks and will have very serious consequences. It is much better to submit your own partially finished work, than to fall into the trap of plagiarism.
- We use fairly sophisticated software which can identify whether two pieces of code are nearly identical (modifications such as renaming variables and reshuffling the positions of functions do not deceive it). If two (or more) students have large portions of their files nearly identical they will be accused of plagiarism or collusion. If found guilty, all parties involved will incur the penalty, regardless of who was the author of the code. For this reason, never show, or give access to, your code to anyone. Do not help a colleague by sharing your code, or you will both be found guilty of collusion.
- It is your responsibility to make sure that nobody has access to your code. Lock the session if you leave your computer unattended.

Submission

- Submit a single text file with extension *.c* through Minerva.
- Download and check the submitted file, to make sure that it is the correct version. We will not accept late submissions if you realise you uploaded the wrong file, or the file appears to be corrupted.
- Binary files (rather than text), and generally files that cannot be compiled, will earn zero marks.
- Even if the file is c code, but the file does not compile, the submission will earn zero marks.

Writing and testing your program

Part of the submission will be automatically evaluated by using the attached test harness. In order for the tests to work, your code must contain some predefined functions. You can find those functions along with their documentation, in the file *template.c*, which should therefore be your starting point. Rename this file, and implement the functions in it one by one. To run the tests, unpack the zip file containing the test harness in the same directory where your program is, and compile it with the following command:

```
gcc -lm -std=c99 –o tests unity.c test.c your_program.c
```

where *your_program.c* is the name of your program file. Each executable must only have a single *main*() function, therefore, when compiling the tests, rename your main function into *mymain*(). If the test program compiles correctly, you can then execute it by running:

```
./tests
```

Compliance with the unit tests is an implementation requirement for this assignment. For your program to be tested it must:

1. Use a 2-dimensional array of characters to implement the grid, and this array must be called "grid" (as already available in the template file).
2. The character for the empty cell must be '.' .
3. The struct called Move contains information about a single move. The struct contains three elements: the player's character (e.g. 'X' or 'O'), and the x and y coordinates of this move.
4. Implement all the core functions provided in the template file:
   int init_grid (int gridsize)
   int player_won (char letter)
   int make_move (int x, int y, char letter)
   struct Move replay_move (int sequence_number)
5. The interface and the core functions must be separate, that is, none of the core functions must contain prints or interactive components (such as scanf).

In addition to the above essential functions you may need to define other functions to make your program clear and well structured.

If you have any problem running the tests, please contact the instructors immediately.

Marking Scheme

With the use of unit testing part of the marking is automated. This is the value of each test:

test_init_grid_limits	(1 Mark)
test_init_grid_clears	(2 Marks)
test_player_won_noone_won	(2 Marks)
test_player_won_main_diagonal	(2 Marks)
test_player_won_secondary_diagonal	(2 Marks)
test_player_won_rows	(2 Marks)
test_player_won_columns	(2 Marks)
test_make_move_limits	(1 Marks)
test_make_move_changes_character	(2 Marks)
test_make_move_does_not_change_anything_else	(1 Mark)
test_make_move_does_not_overwrite	(2 Marks)
test_replay_moves	(5 Marks)

The remaining of the marking will be assessed as follows:

The interface is separate from the core functions, which have no interactive component	(2 Marks)
The user interface works correctly	(10 marks)
The program is well structured, clear, and efficient	(3 marks)
The program uses comments properly	(1 mark)

Total: 40 marks.

Tier 2 (Optional)

The above program allows two persons to play against each other using the computer to show the game board, detect wins, and record the moves they have made. Although this is useful, a much better program would allow one player to play against the computer.

Develop your program so that it will now have two options:

Option 1: Allow two players to play against each other (as before).

Option 2: Allow one player to play against the computer.