University of Leeds **School of Computing**
Procedural Programming (COMP1711)
2017-2018

Second Attempt Coursework

(40 Marks)

Submission Deadline:
16/July/2018 at 10:00 am

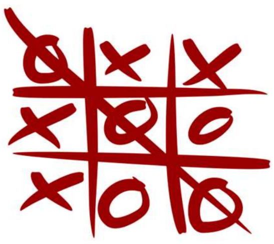
The Brief

You will write a C program for a computer 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.

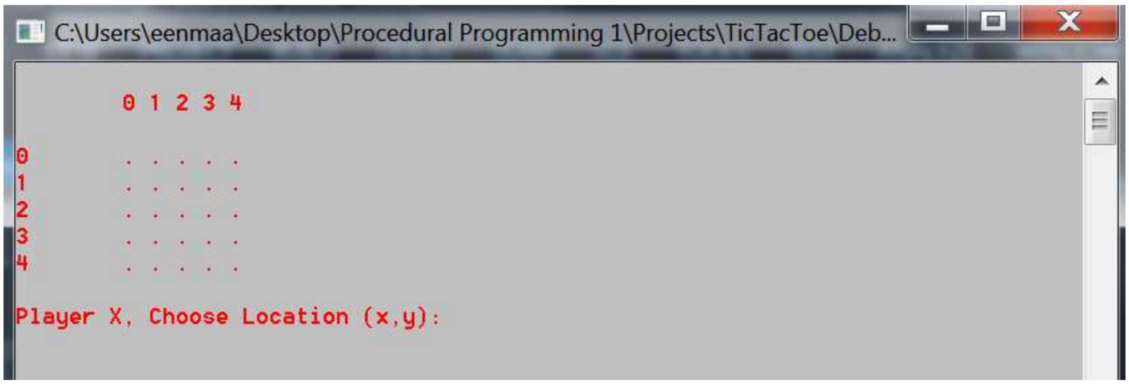
Using a computer can help people play this game more easily~~easier~~, by drawing the empty grid, organizing turns, and detecting a win as soon as it occurs.

The Task

Write a C program that allows two people~~rsons~~ 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:

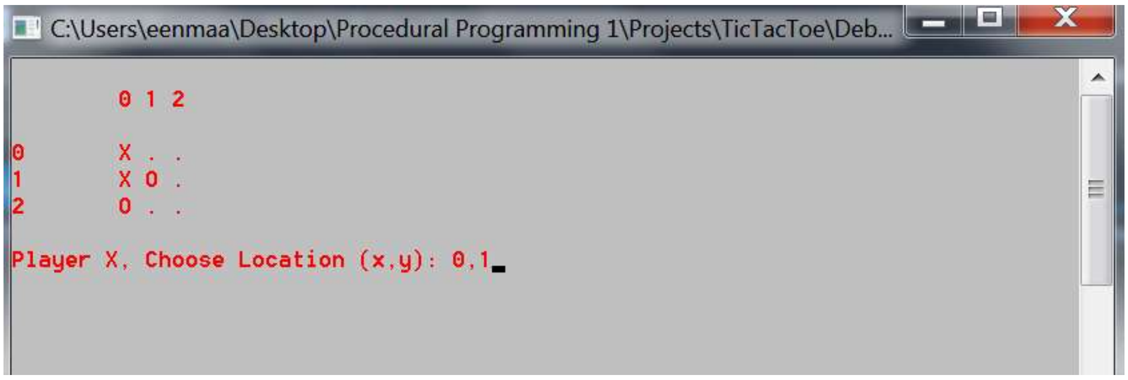

C:\Users\eenmaa\Desktop\Procedural Programming 1\Projects\TicTacToe\Deb...

```
        0 1 2 3 4

0       . . . . . .
1       . . . . . .
2       . . . . . .
3       . . . . . .
4       . . . . . .

Player X, Choose Location (x,y):
```

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:

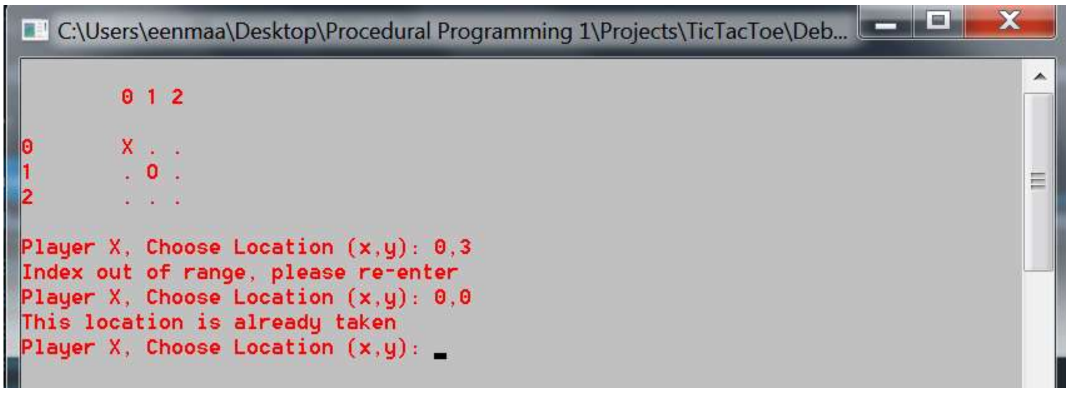

C:\Users\eenmaa\Desktop\Procedural Programming 1\Projects\TicTacToe\Deb...

```
        0 1 2

0       X . .
1       X O .
2       O . .

Player X, Choose Location (x,y): 0,1
```

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:

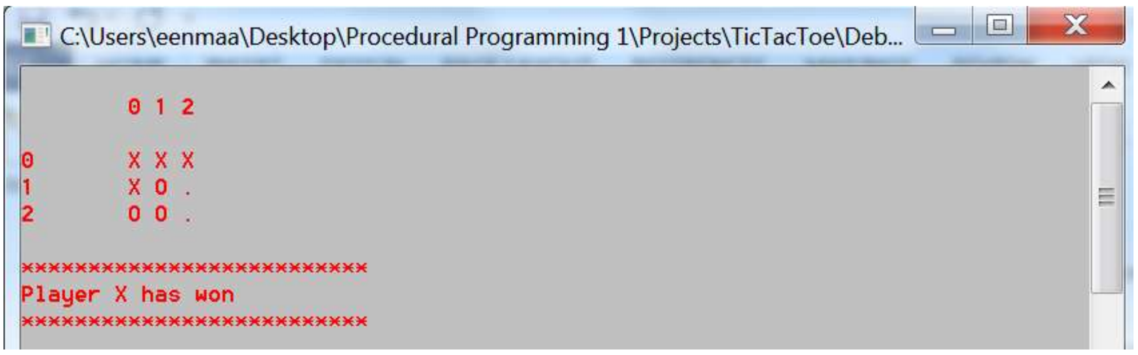

C:\Users\eenmaa\Desktop\Procedural Programming 1\Projects\TicTacToe\Deb...

```
        0 1 2

0       X . .
1       . O .
2       . . .

Player X, Choose Location (x,y): 0,3
Index out of range, please re-enter
Player X, Choose Location (x,y): 0,0
This location is already taken
Player X, Choose Location (x,y):
```

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.

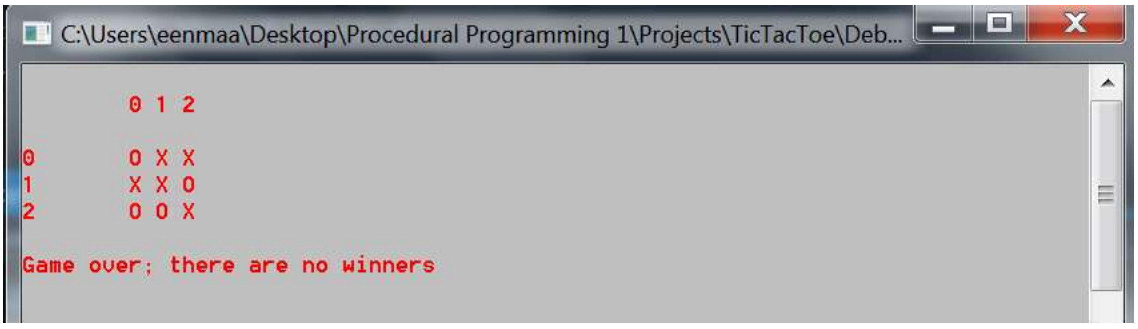

C:\Users\eenmaa\Desktop\Procedural Programming 1\Projects\TicTacToe\Deb...

```
       0 1 2

0      X X X
1      X O .
2      O O .

**************************
Player X has won
**************************
```

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


C:\Users\eenmaa\Desktop\Procedural Programming 1\Projects\TicTacToe\Deb...

```
       0 1 2

0      O X X
1      X X O
2      O O X

Game over; there are no winners
```

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 a zero mark 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 a software tool to detect plagiarism automatically, so if you do work with someone else, or submit someone else's work it WILL be detected.**

Unit testing is a testing procedure in which units of programs are tested in isolation, before they are integrated. We are implementing unit testing for this assignment, which will help you assess whether your code is working before submitting~~turning~~ it ~~in~~, and will also partially automate marking.

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". Declare the array in your program as follows:

```
#define MaxGrid 10
char grid[MaxGrid~~MAXGRID~~][MaxGrid~~MAXGRID~~];
```

This code declares a 10x10 array, large enough to contain the grid for the game. The functions specified below must update this array.

2. The character for the empty cell must be '.' .
3. Implement the function:

   ```
   int init_grid(int grid_size)
   ```

   which initializes the grid with size grid_size (from 3 to 10), preparing it for a new game. The grid must be filled with '.'. The function returns true (any non zero number, for instance 1) if grid_size is not valid, that is if it is not in the range [3,10]. The function returns false (that is, the number 0) otherwise.
4. Implement the function:

   ```
   int player_won(char letter)
   ```

   which returns true (that is, any non-zero number) if the player playing with character `letter` (either X or O) won, or false (that is 0) otherwise.
5. Implement the function:

   ```
   int make_move(int x, int y, char letter)
   ```

   which makes the player playing with character `letter` set the cell at (x,y) with his symbol (X or O). The function returns true if something goes wrong, for instance if cell (x,y) had already been used in a previous move (the player can't choose a cell that is not empty), or x and y are not in the correct range. The function returns false if it terminates without any error.

A file template.c is provided for your convenience, including all the necessary functions. You can rename this file and use it as a starting point to develop your program.

How to run the tests

Download the file test_harness.zip from Minerva, and unpack it in the same folder as your program. For the purpose of these instructions I will assume your file is called YOURFILE.c but please do use a better file name.

Your program will have its own `main`() function to play the game. However, the test file also has its own main function, and there can be only one. So, when you compile the tests, rename your main function into something else, for instance `main_mine`().

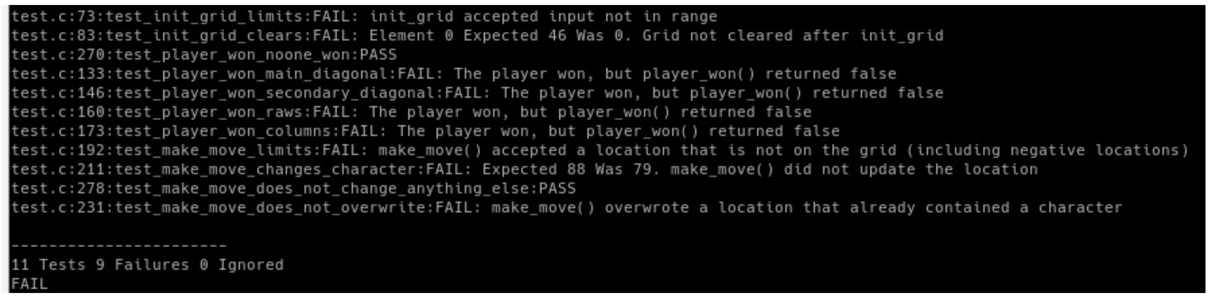
Compile the tests as follows:

```
gcc -o test --std=c99 test.c unity.c YOURFILE.c
```

This creates an executable called test, which you can run as usual with:

```
./test
```

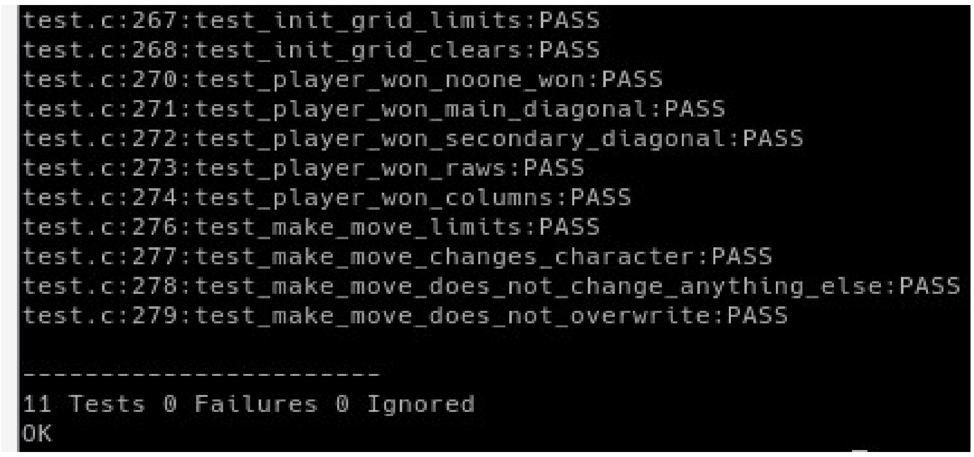
Compile and run the tests immediately, from the provided template, before doing any development. You will get the following output:



```
test.c:73:test_init_grid_limits:FAIL: init_grid accepted input not in range
test.c:83:test_init_grid_clears:FAIL: Element 0 Expected 46 Was 0. Grid not cleared after init_grid
test.c:270:test_player_won_noone_won:PASS
test.c:133:test_player_won_main_diagonal:FAIL: The player won, but player_won() returned false
test.c:146:test_player_won_secondary_diagonal:FAIL: The player won, but player_won() returned false
test.c:160:test_player_won_raws:FAIL: The player won, but player_won() returned false
test.c:173:test_player_won_columns:FAIL: The player won, but player_won() returned false
test.c:192:test_make_move_limits:FAIL: make_move() accepted a location that is not on the grid (including negative locations)
test.c:211:test_make_move_changes_character:FAIL: Expected 88 Was 79. make_move() did not update the location
test.c:278:test_make_move_does_not_change_anything_else:PASS
test.c:231:test_make_move_does_not_overwrite:FAIL: make_move() overwrote a location that already contained a character

-----------------------
11 Tests 9 Failures 0 Ignored
FAIL
```

As you can expect right now most of the tests fail, which is reasonable since you did not do any development yet! Now you can start implementing the functions. Run the tests regularly after you implement each new function, to see how the tests turn from fail to pass, and to have confirmation that your code is correct. If your code is correct, it will pass all the tests, and the output will look like this:



```
test.c:267:test_init_grid_limits:PASS
test.c:268:test_init_grid_clears:PASS
test.c:270:test_player_won_noone_won:PASS
test.c:271:test_player_won_main_diagonal:PASS
test.c:272:test_player_won_secondary_diagonal:PASS
test.c:273:test_player_won_raws:PASS
test.c:274:test_player_won_columns:PASS
test.c:276:test_make_move_limits:PASS
test.c:277:test_make_move_changes_character:PASS
test.c:278:test_make_move_does_not_change_anything_else:PASS
test.c:279:test_make_move_does_not_overwrite:PASS

-----------------------
11 Tests 0 Failures 0 Ignored
OK
```

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. There are 11 tests in the test suite and each one is worth 3 marks. Therefore, the automated part of the marking scheme is as follows:

Function `init_grid()` (2 tests)	(6 Marks)
Function `player_won()` (5 tests)	(15 Marks)
Function `make_move()` (4 tests)	(12 Marks)

The remaining of the marking will be assessed as follows:

The user interface works correctly	(4 marks)
The program is well structured, clear, and efficient	(2 marks)
The program uses comments properly	(1 mark)

Total: 40 marks.