Curtin University — Computing Discipline

UNIX and C Programming (COMP1000)

Semester 2, 2019

Assignment

Due: Monday 8AM 21/10/2019

Weight: 20% of the unit mark.

1 Introduction

Your task for this assignment is to design, code **(in C89)**, test and debug a m-n-k tic-tac-toe game

In short, your assignment will:

- Interact with the user via a Terminal based menu.
- Extract the game settings from a file.
- Play a game of tic-tac-toe of a board size of m*n.
- Log all moves that have been executed.
- Save the logs to a file.

This assignment is separated into multiple sections. Make sure to read each section *carefully* before starting.

2 Code Design

You must thoroughly document your code using C comments (/* ... */). For each function you define and each datatype you declare (e.g. using struct or typedef), place a comment immediately above it explaining its purpose, how it works, and how it relates to other functions and types. Collectively, these comments should explain your design. (They are worth substantial marks - see Section 8)

Your code should also be separated logically into multiple c and h files that overall have a single purpose. Marks will be deducted if monolithic functions and/or files are used.

3 Task Details

Read this section carefully as there is a lot of detail to take in. You may need to read it several times to ensure that you do not miss anything. It is highly recommended that you spend some time coming up with a design template of your code before you start coding.

The individual tasks that are to be completed have been set up in a linear format. Therefore, before moving onto the next task, ensure that the current task you're working on works.

3.1 Input Files

Your program should accept one command-line parameter that will be the filename of the settings file. For example

```
[user@pc]$ ./TicTacToe settings.txt
```

Settings File

The settings file will contain the 3 values that are needed for your game. These values are:

- M – The width of the board
- N – The height of the board
- K – The number of matching tiles in a row to win

An example settings file is as follows:

```
M=5
N=4
K=3
```

The different settings can be placed in any order in the file and are case insensitive. Your code also needs to be able to handle any invalid settings file. This includes but is not limited to:

- File not existing.
- Not having all 3 required settings.
- Any settings having invalid values (cant have a negative width for example).
- Having duplicates of a specific settings.
- Invalid line format.

If in the instance where a file is found to be invalid for any reason, a meaningful prompt to the user is to be made and the program is to exit safely.

3.2 User Interface

Your program needs to have a menu that the user is able to interact with. Through this menu the user must be able to:

1: Start a new game

2: View the settings of the game

3: View the current logs

4: Save the logs to a file

5: Exit the application

Your menu should only require an integer input to select a menu and needs to be able to handle errors that may occur. ie, an input that isn't within range

When it comes to viewing the settings of the game, it is up to you on how it looks.

Playing The Game

The normal game of tic-tac-toe consists of 2 players taking turns to place a tile on a 3x3 board until someone gets 3 in a line or its a draw. m-n-k tic-tac-toe is very similar however the size of the board is dependant on the values of 'm' and 'n', and the required tiles in a line are dependent on the 'k' value.

When playing the game, the board is to be displayed graphically on the terminal. All tiles should represent where players have placed a tile and what cells are empty. An example of this is

```
======================
||   |   | O | X | O ||
||-------------------||
|| O |   |   | O | X ||
||-------------------||
||   | X |   | X | O ||
||-------------------||
||   | X | O | O | X ||
======================
```

How you represent the board is up to you, however you have to have a barrier between each column and row of the board.

For playing the game, a player is to input the coordinates of a tile that they wish to place their tile on. The coordinates of the tiles are to be that the top left tile is (0,0). If the tile is already occupied then a new cell must be selected otherwise the board is to be redrawn with the new tile.

For example if it was X's turn and they select the tile (1,0) then reprint the entire board. You are not expected to print a character on the existing board.

Note: The input of (1,0) is representing the (x,y) coordinates of the grid. Therefore you can treat is as go right 1 and down 0.

Your input doesn't have to be in the exact form (x,y), as long as it is intuitive and understandable.

```
======================
||   | X | O | X | O ||
||------------------||
|| O |   |   | O | X ||
||------------------||
||   | X |   | X | O ||
||------------------||
||   | X | O | O | X ||
======================
```

How you select a starting player is up to you. It can be random, selected at the start of the game, or hard coded to always be a certain player.

Note: For the time being don't implement the feature to detect if a player wins the game. This feature should be focused on later down the line after the other important features have been implemented.

Feel free for the time being to assume that the game always ends in a draw.

3.3 Logging

Your game is to keep a log of all actions performed by players during a game. When a player takes a turn an entry is to be added to a Linked List to be used later. Within this entry you must be able to store the tile that was placed and where it was placed.

Your Linked List must be a **Generic** Linked List and therefore be able to store any type of data. Substantial marks will be lost if it isn't properly generic.

Displaying Logs

When it comes to displaying the logs to the terminal, they must be done in the following format

```
SETTINGS:
   M: 4
   N: 3
```

```
   K: 3

GAME 1:
   Turn: 1
   Player: X
   Location: 0,2

   Turn: 2
   Player: O
   Location: 3,2

   ...

GAME 2:
   Turn: 1
   Player: O
   Location: 0,0

   Turn: 2
   Player: X
   Location: 0,1

   ...
```

Saving Logs

Logs are to be saved in the same format as they are displayed. The name of the log file is to be in the format:

`MNK_<M>-<N>-<K>_<HOUR>-<MIN>_<DAY>-<MONTH>.log`

Where the value of 'M', 'N' and 'K' are the settings of the game.

An example would be "MNK_3-4-2_18-20_09-07.log". If in the instance where a logfile already exists with this name you can overwrite it. Also take note that the hour in in 24 hours format.

Note: For assistance in getting the current date, these links may be of help:

- https://www.techiedelight.com/print-current-date-and-time-in-c/
- https://www.tutorialspoint.com/c_standard_library/c_function_localtime.htm

3.4 Finding a winner

A winner is to be determined once a player has 'K' tiles in a line, either vertical, horizontal or diagonal. Once this has been achieved a message is to appear to state the victor. In the case where it is a draw then a different message it to appear.

Return back to the main menu after the game is completed.

4 Conditional Compilation

You must provide a single valid makefile for your assignment. When testing your code, the makefile will be used to compile it.

Warning: Failure to provide a makefile will result in zero (0) marks being awarded for compilation and it will be assumed that your code does not compile.

Note that for a makefile to be valid, all 5 compilation flags are to be used. Any additional flags may be removed during demonstration and testing. The required flags are:

```
-Wall -ansi -pedantic -Werror -std=c89
```

The -std=c89 flag is used to enforce the compilation to be C89 compliant.

Condition Compilation

Normal Compilation - The compiled assignment as described above.

Secret - When this version is compiled, the user losses the ability to save logs to a file. Logs may still be displayed to the user, however the menu item to save the logs is to be deactivated (either hidden or error message when selected is fine)

Editor - When this version is compiled, the user now has the ability to edit the values of M,N and K at run time. Another feature is to be added to the menu to allow the user to do this. You may ignore having previous settings when it comes to the logs (Use the current values when displaying/saving)

You must use conditional compilation (preprocessor macros) to achieve the different functionalities. If you achieve this by having multiple c files for each version then you will not receive marks for this section.

5 Report

You must prepare a report that outlines your design and testing. Specifically:

1 For each file that you have, write a paragraph explaining the overall purpose of it.

2 Describe (in 1-2 paragraphs) how you implement the storing and printing/saving of the logs. Talk about any complications that you can into while designing this component.
3 Demonstrate that your program works and how to use it, including:
   - The command-line used to execute your program
   - The output of the board as shown on the screen
   - The input the user uses to interact with the menu
   - The contents of the input file
   - The contents of the log file

Your report should be professionally presented, with appropriate headings, page numbers and a contents page. You will loose marks for poorly written/badly formatted reports!

The report is to be *maximum* 3 pages long (this does not include cover page, contents page and references).

6 Submission

Once complete you are to submit your entire assignment electronically, via Blackboard, before the submission deadline.

Your submission is to be a single .tar.gz file containing:

A declaration of originality – A complete declaration of originality is to be filled out and submitted as a PDF file.
Your implementation – including all your source code (.c files, .h files, makefile, etc…). Note that all pre-compiled code will be removed before marking.
Your report – A single PDF file containing everything mentioned in Section 5

You are responsible for ensuring that your submission is correct and not corrupted. You may make multiple submissions, but only your latest submission will be marked.

No extensions will be granted. If exceptional circumstances prevent you from submitting an assignment on time, contact the Unit Coordinator ASAP with relevant documentation. After the due date and time is too late.

Late submission policy, as per the Unit Outline, will be applied.

Warning: Ensure that the filetype of all submission are correct. If your report is a docx or txt file it will be treated as a **non-submission**. Same applies for your submission. If you submit a .zip, .7z, .rar, etc…, then you will receive zero (0) for the whole assignment.

7 Demonstration

You will be required to demonstrate each of the worksheet submissions in your practical session. The final assignment submission will also be demonstrated in the practical session

following the weeks of the submission date.

If you cannot attend your registered practical session you must make arrangements with the lecturer as soon as possible. You may be asked several questions about your assignment in this demonstration.

Failure to demonstrate the assignment during your registered practical will result in a mark of ZERO (0) for the entire assignment. During the demonstration you will be required to answer questions about your design. If you cannot answer the questions satisfactorily, you will receive a mark of zero for the assignment.

8 Mark Allocation

Every valid and complete assignment will be initially awarded 100 marks. For an incomplete assignment, the mark will be applied proportionately. Marks will then be deducted for the following:

40 marks — Implementation
None of these marks will be lost if all required implementation is done.

40 marks — Coding practices, coding standard and Commenting.
You will not lose any of these marks if you have followed the coding standard and good coding practices (see Blackboard, resources section), your code is well-structured, including being separated into various, appropriate .c and .h files and if you have provided good, meaningful explanations of all the files, functions and structs/typedefs needed for your implementation.

20 marks — Report.
You will not lose any of these marks if your report is complete and professionally presented.

Marking Strategy — starting mark
The Marker will first assess the overall assignment, and determe the "starting mark", and then carefully go through the assignment awarding and/or deducting marks as appropriate (see below).

Mark Limit — Working Product.
You will not lose any marks if your program compiles, runs and performs the required tasks without unexpected error. The marker will use test data, representative of all likely scenarios, to verify this. You will not have access to the marker's test data before hand.

Things that will cause a deduction are, but not limited to:

(a) Quality of assignment overall

(b) Not following the assignment specification

(c) Breaking "zero mark" rules

(d) Monolithic files and/or functions

(e) Quality of makefile

(f) Quality of report

(g) Code that doesn't compile

(h) Omission of conditional compilation

(i) Occurrence of memory leaks/errors

(j) Error handling on files/input

(k) How much of your functionality is working

The deductions will reduce for maximum amount of marks that you can possibly get. For example if you had severe memory leaks then a deduction of 10 marks might be applied. This will cause your maximum mark to be 90.

Signoffs:

Once your mark has been calculated, the signoff results will be applied as follows:

Your total practical signoffs will be awarded a mark out of 10, which will have the following effect (multiplier) on the assignment mark.

Total	**Effect**
8 - 10	100%
7 - 7.9	90%
6 - 6.9	80%
5 - 5.9	70%
3 - 4.9	50%
0 - 2.9	30%

End of Assignment