UCP Assignment Report

Mahmudul Hossain
Student ID : 19303235

The assignment required us to implement a tic tac toe game with variable dimensions and win conditions. For the game to run, I have included board.h, display.c, file.c, game.c, LinkedList.c, logger.c, main.c, myTime.c, operation.c and win.c where each .c file has its own header file except main.c.

Inside board.h, I have included all my typedef struct aliases.For the entire game, I have used three typedef structs which are Board, log and Settings. Board struct is used to create a 2D array platform for the tic tac toe game to be played which consists of symbol(String) and state(Integer). symbol is used to store the character which can be either "X", "O" or " " for every co-ordinate, which is then used to display the state of the current board to the user. state is used to determine whether the co-ordinate is occupied or not which is initially set to 0 and after a symbol is entered it is set to 1 meaning that specific co-ordinate is occupied. Struct log is used to store each move made by the player which consists of gameNum(Integer) the number of games from the start, turn(Integer) records the move number made by the player, player(String) to determine whether the move is made by player "X" or "O", x(Integer) and y(Integer) to store the co-ordinates entered for that particular move in the board. Struct Settings is used to store the settings of the game which includes the dimensions of the board as rows(Integer) meaning height and columns(Integer) meaning width of the board and win(Integer) representing the number of matching symbols to win the game. Struct Settings was quite useful throughout the program since it enclosed multiple variables in a single user-defined struct which greatly reduced the number of imports for functions and allowed multiple returns.

Inside display.c, I have included all the functions that are used to display specific output to the user which includes the current settings, main menu, current state of the board and winner of the game except printing out the logs which I have included in log.c.

The board of the game is displayed in this format with M=3, N=2 and K=3

```
=====================================================
||   |   |   ||
---------------------------------------------------
||   |   |   ||
---------------------------------------------------
=====================================================
Player X's turn, enter position:
<x>, <y>
```

If 1,0 is entered the output of the board would be

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

Inside file.c, I have included all file input/output functions carried out in the game. readFile() imports invalid height, width and win along with filename from the main where the filename is used to open the

file for reading. I have used the #define directive TRUE AND FALSE, with num initially set to FALSE, to carry out checks if the read operation was successful or not. Once the file is opened for reading, it expects the file to be in a format like:
M=<Integer> //Width of the board (columns)
N=<Integer> //Height of the board (rows)
K=<Integer> //Consecutive matching symbols of a winner (win)
where MNK can be in any order. For a successful read operation, I have read till EOF where I used value(Character) and temp(Integer) to store values temporarily. I have also used a readCount variable to store the number of successful assignment to each varibale rows, columns and win which determines if a duplicate setting was read or not. Once value and temp are assigned, I carried out checks comparing variable value with 'M','N','K' and assigning temp variable to rows, columns and win after a match. Finally, after reading till EOF, I made final checks to determine if rows, columns and win had valid values where rows and columns had to be at least 1 or greater in order to allocate a 2D array of board, win to be non-negative and readCount to be exactly 3. Once all the conditions are met num is set to TRUE meaning a successful read operation and returned to main. writeFile() is used to write the current settings and logs into a specific filename. It imports a Linked List of logs and a Settings struct containing the current settings of the game. Current time is generated using using the calcTime() situated in myTime.c which is incorporated with the filename. The filename is then used to open the file for writing where at the top, the current settings of the game are saved followed by writing each and every logs that are stored in the Linked List by traversing through the Linked List recrodLog, adhering to the format mentioned in the assignment specification. Initially, I tried to removeFirst from the Linked List but I realised that the previous log entries were not available for future saving of current logs, so then I decided to traverse and access the data instead of removing. Any invalid read/write operations are caught and handled to my best practice with the help of perror and ferror functions.

Inside game.c , I have stored the functions that are related to the running of a single tic tac toe game incorporating functions situated in win.c to determine winner. play() imports an empty board,current settings of the game and Linked List of logs to intitiate the game. Before starting the game, I have made a check that if the winning condition is equal to 0, I have considered the match to be a draw because it is illogical to initiate a game without knowing the winning factor. If the win is greater than 0, the variable winner is set to = -1 indicating no winner has been determined. While winner = -1 the game will keep on continuing until winner = 0 (meaning a draw), 1 (meaning "X" is the winner) and 2 (meaning "O" is the winner). Initially I have hard-coded the game to start with player "X" and used a function within game.c makeMove() for each player to make a valid move inside the board and return a winning value. makeMove() iterates the user to enter valid co-ordinates in <x>,<y> form where the state must be equal to 0 beforehand and when valid move is entered, the board gets updated and a checkWin() situated inside win.c, is carried out to determine if that move was a winning move and finally the winner value is returned to play.After a valid move, the move is recorded and stored in the Linked List of logs using insertLast() and a check function checkDraw() inside game.c determines if the game is draw which checks if the total number of moves carried out in the game exceeds the total number of entries in the board.If exceeded it returns 0 to the winner meaning the game is a draw. After all checks are carried out for player "X", player "O" is then given the option to play in the board where exactly same checks are carried out.
After a winner is determined, winner is displayed to the user using displayWinner() from display.c and memory allocated for current board is freed using freeBoard() function in game.c.
It also contains a function changeSettings() which is only used when Editor is defined during conditional compilation where the user enters valid input for each variable in Settings struct which then updates the settings following the new dimensions and win condition for the game.

LinkedList.c is a generic LinkedList which is declared in function operation operation.c to store the log entries of the game until the user exits the program.

logger.c contains all the functions related to log entries entered by the user with the help of struct log. It allocates each log entry using the createLog(), deallocates memory using freeLog() and has printLog() which outputs the contents of log entries. freeLog() and printLog() are used as function pointer imports in LinkedList's freeLinkedList() and printLinkedList() which are called inside operation() since Linked List is generic. Hence I did not define printLog() in display.c because I want printLog() to be specific to handling log entries only making logger.c specific to handling log entry structs.

Inside main.c, I have the main method which starts the program.I have allowed exactly two arguments to be passed in which is ./TicTacToe <settings_file>, if it is not exactly two arguments, I have outputted an error message along with usage information. I have initialised rows, columns, and wins to be invalid and passed it onto readFile function in file.c to get updated. If the readFile returns TRUE, then operation() is called passing in valid settings else an error is outputted to the user mentioning that the input file is invalid.

myTime.c contains only a single function called calcTime() which assigns the current time to its imports and return it back to the calling function writeFile() in file.c.

operation.c has two functions validChoice() and operation(). validChoice() determines if the user input from terminal is within range and in the right format. To obtain a correct user input from terminal, I have used additional variables c(Integer) and check(Character) which are also used in functions changeSettings() and makeMove() in game.c where check determines if a single integer value is entered by making sure that the integer ends with '\n' which is determined with variable check, and if an invalid user input has occurred, c is used to clear the input buffer by receiving characters from terminal until stdin is empty to ensure that the next user input does not get affected by the previous input. I tried to clear the input buffer using fflush(stdin) but it failed to clear stdin properly since fflush(stdin) is said to be undefined.
operation() is used to direct specific operations to be done in the program according to the user input, it imports valid rows,columns and win from main and allocate memory for a Settings struct which will be used throughout the running of the program. The operations to be carried out varies depending on the conditional compilation like: when Editor is defined, an additional option is displayed to the user to change the settings of the game and when Secret is defined, the user gets an error output when they select to save logs onto a file. Other than that, operation() directs to any other function the user chooses. When the user selects to start a game, memory is allocated for the board by calling createBoard() in game.c that creates a 2D array of Board structs and initialises all entries to be empty.The user can also choose to display the current settings of the game also with the help of displaySettings() from display.c.

win.c file checks whether the updated board contains a matching row, column, diagonal or antidiagonal by iterating through every co-ordinates and using strcmp() to carry out checks for consecutive matching symbols of either "X" or "O", where the number of matching co-ordinates are stored in variable count. Each comparison on rows, columns, diagonals and anti-diagonals are carried out in specific functions checkHorizontal(), checkVertical(), checkDiagonal() and checkAntiDiagonal().Inside these functions, if count equals to the winning condition, then a winner is decided using checkPlayer() which returns the winner depending on the player.

Overall, there is no possible memory leaks and the program works correctly with appropriate settings file. If the compilation is carried out without defining Secret, the log files are saved with a MNK_<M>-<N>-<K>_<HOUR>-<MIN>_<DAY>-<MONTH>.log filename and stores in exactly the same format as they are displayed logs to the user which is :

```
SETTINGS
        M : <Integer>
        N : <Integer>
        K : <Integer>

GAME : <Integer>
        Turn : <Integer>
        Player : <String> // X or O
        Location : x<Integer>,y<Integer>
```

Finally when the user chooses to exit the program, appropriate free functions are called to free the Linked List and deallocate the memory for Settings struct in operation().

.