Unix and C Programming

Benjamin Le

19798734

Contents Page

Menu.c

The purpose of my menu.c file is to act as the overall structure of the program. It contains a case statement which gives the user a choice between 5 options. If the program is compiled with EDITOR, an additional choice is unlocked, making it 6 options. I decided to make my menu act as my main instead of having a separate main that calls the menu because I found that this was easier for testing purposes.

Fileio.c

My fileio.c opens the file and reads it. If the file is null, it'll print out an error message. If the file was read, it'll loop through the file until it reaches the end and will scan a character followed by an integer. The character is assigned to a variable I created called tempLetter and the integer is assigned to tempNum. My if statements check if tempLetter is the upper and lowercase version of the letters that I am looking for, in this case M N and K. If it meets the criteria, it sets tempNum to either M N or K depending on what it is to a struct which I have created. I chose to create a struct because it made it easier to refer to M N and K through the struct instead of having to refer to them separately. My file only checks for if the file does not exist and if the M N and K are in any order and if they're upper or lower case. I left the rest of the checks until last because I was short on time and had to move on with the rest of my program. Towards the end I still didn't find the time to but an idea of how I'd implement a duplication check is to have a count which increments if there is an M, N or K in the file. If the count is greater than 1, then a duplicate is found and an error has occurred.

Game.c

This file contains functions relating to the game of Tic Tac Toe. Starting off with gameBoard, this function acts as a "mini main" and calls the other functions within the file. My initialiseBoard is where I malloc the rows and columns of my board, allocating enough space depending on the size of the board generated from the settings file. I also initialise my board to start off as empty which is achieved by initialising my board to ' '.

The next function drawBoard is where I made the design of the board. If I had more time to work on the assignment, I would have implemented a quality of life design which included the coordinate numbers along the top and side of the board so that it allowed the user to choose where they wanted to put their X or O easier. My board also does not print the bottom board. These are some changes that would improve the overall aesthetic of the design.

TicTacToe function is where the game is "played". I have a do while loop which creates a log, logs the players turn, re-draws the board and checks if there is a winner. This loop continues if there is no winner yet or if a draw occurs. I included them in the loop because after every turn, the user is prompted with a message to make their move by choosing a coordinate they wish to place their piece and the board is reprinted with either an X or O in the respective coordinate. My "draw" check which determines if a draw has occurred does not work as intended. It did work at first but after changing lots of code, I found out that it no longer worked and I was short on time to fix it. The way I thought of this was that I knew a draw occurs when the board is completely filled and a win hasn't been found. In order for the board to fill up, it must be the size of the width and height of the board, thus a draw must be width * height.

My playerTurn function is where I have my prompt to ask the user where they want to make their move. I have an integer variable named count which starts at 1 and by using the modulus operator, I am able to differentiate players with either an X or O. If I mod the count by 2 and it is equal to 0, the player is O, otherwise the player is X. This means that the beginning of every game, the first player to make their move will always be X. Subsequent games after the first will switch between the X and O starting next. Eg. Game 1, X starts. Game 2, O starts. Game 3, X starts. Game 4, O starts. I made my count a static variable because I ran into the issue where the count would make it so that X would always be the selected player. By making count a static variable, it retains the count value instead of resetting back to 1. In this function I also use the current players coordinates as pointers so that I can use them to do my win condition checks later on in the code.

Winchecks.c

This function is where I keep all my win checking conditions. Originally I had them all stored in my game.c file, as it is considered a huge part in playing the game of Tic Tac Toe, but I decided to separate it into a different file so that a single file wouldn't have more than 200 lines of code. This file has another "mini main" which calls the rest of my functions and will return an integer named win. If win is 1, a win has occurred otherwise by default it is 0. I used integers 0 and 1 because C does not use boolean. My checkRow, checkColumn, checkDiagonal and checkAntiDiagonal all use the same idea.

First I have streak, count and win as integer variables. Streak is the amount required for the game to be considered won. In this case it is the K value in the settings file. I have a while loop which calls my traverse method, and this method is where it checks for wins. The traverse method uses pointers to keep track of where the current player is and then adds on xChange and yChange which are fixed numbers that I used to traverse the board depending on where the current player is. Assume the board is a 3x3 board and the player starts off in the centre of the board (1,1). Using the checkRow function as an example, I set the xChange and yChange imports to -1 and 0 because in the X coordinate, I want to "traverse" up one square but stay in the same y coordinate. This keeps updating until the current players piece is no longer the same piece. Once that happens, it will increment 1 because it needs to count the current spot and then traverses down the board, so coordinates 1 and 0 are then given to the function. If the count is then equal to the streak, a won has been determined.

Originally I thought of looping though the entire board and searching for the win but thinking about where the current player is and traversing the area around the current player is more efficient.

LinkedList.c

This file is a generic linked list created from the practicals. This differs from other linked lists made in the pracs because it involved function pointers set to void, which is then typecasted to the actual data so that it can be used for anything instead of just for one thing. Within this function other than the generic linked list functions, I also have a printLinkedList function which simply just prints the contents of the linked list.

Logs.c

This function contains all the log functionality. I have a createLogs function which simply declares and mallocs the logs. I chose to make my logs a struct because I found it easier to refer to the variables in the struct as a whole instead of individually. My displayLogs function simply prints out what the current game, turn, player and location is which I set in my game.c file, in playerTurn function. I kept a game counter in my main which increments each time the user presses the first option because this starts a new game and by incrementing it each time a game has started, a new game will occur. My writeToLogs function is similar to displayLogs but this time it prints it into a file.

Implementing and storing of logs

My logs was created as a struct in my logs.h file. Within this struct I have int variables as game, turn, x, y and a char value as the player. I set my x, y and player values in my playerTurn function in game.c because this is where the game determines which player is which piece (X or O). The coordinates x and y are also set here because this is where the user is prompted to input the location they wish to place their player piece. My turn is set in ticTacToe function in game.c because each time a turn is made, there is a counter which increments and keeps track of the amount of turns there are. My game count is set in my menu where I have a counter which increments each time the first option (start game) is chosen. With these variables set, I simply print them out to the terminal using displayLogs function and then write them to a file using a combination of writeToLogs and saveLogs functions. My saveLogs functions implements code from the websites listed in a comment above the function. My logs are stored in a linked list and is then printed into a file.

An issue I had which took me a while to debug was that my linked list would continuously print out the very last move made by the player instead of each individual move. I fixed this by malloc'ing the log in my loop so that it instead of having a size of 1 log and it gets overwritten each time, there would be space for more logs since I've looped it.

Another issue I had was my player kept returning the value 88 instead of either an X or O. After a while I figured that it was returning the ASCII value of X instead of the actual digit. I changed this by changing my %c into a %d.