Hanna Foerster
Username: lzhc88
Student-ID no.: 000816884

Reflective Report: Systems Programming Coursework

When I started programming the connect4.c programming, I wanted to make the data structure to store the grid into one that was intuitive to call every time I was using it. For me this was a two-dimensional character array. In C, a two-dimensional character array that does not yet have a definite size is the same as a pointer to a pointer to a character, which is how I initialized my grid. With the input file, the program would then allocate via dynamic memory allocation (malloc) the size of a character array times the number of rows the input file has, to this pointer to a pointer to a character. Then, every character array (pointer to a character) was allocated storage space of the size of a character times the number of columns the input file has. In order to obtain the size of a row and column of the input file, I checked this first using fgetc() multiple times to count the characters in a row and fscanf() multiple times to get the row size. In this way if for example my two-dimensional array was called array, I could access an element in the grid by simply calling array[i][j], where i specifies the row and j specifies the column.

I have made my code robust by trying to be paranoid about any error that my code could encounter especially from user inputs or memory allocation. For example, for the input of the initial text file that contains the grid, I conditioned to check that this input was comprised only of the characters '.', 'x', 'o', 'X' or 'O' and to throw an error if not. Furthermore, I have a check in place to count that the input has at least 4 rows, 4 columns and at most 512 columns. Moreover, my code also checks the input to ensure that all rows have the same length. Lastly, the initial grid is checked on whether gravity has worked. If not, my code will fix the grid, so that all tokens are at the bottom, how they would be placed, had gravity worked.

Next, for the move input the players give for every round, there is a check in place in the function read_in_move to check whether this input is an integer. After testing the program, it worked very strangely with character inputs, hence I have made my program throw an error and exit if a non-integer input is given.

In terms of memory allocation, I have made sure to free all dynamically allocated memory not only when the program finishes, but also when it exits because of an unexpected error. If an error occurs in the read_in_file function, I have also made sure that only memory that was already allocated is freed, which is why I did not use the clean_up function in this part. Moreover, in the read_in_file function, the File infile is still open, so when exiting I made sure that the program closes infile first. In this way no memory error is detected by termination when checking with valgrind. In addition to that an appropriate error message that indicates the reason of termination is output for any exit out of the program due to errors.

Another focus I had when writing my code was to write it cleanly. For this purpose, I implemented extra functions for some repeated processes, such as checking whether 4 tokens from one player were in a row in the four_in_a_row function. In this way this function could be called a few times throughout the program and avoided the program from having repeatedly written code. To be more specific, in case of the four_in_a_row function, it is comprised of three additional functions, namely check_horizontal, check_vertical and check_diagonal. These functions check whether a player has four tokens in a row, horizontally, vertically or diagonally, respectively.

Additionally, I commented the most important processes to be able to understand my code later and to facilitate the reading of my code for anybody else. I took special care regarding the pointers in my

Hanna Foerster
Username: lzhc88
Student-ID no.: 000816884

two-dimensional array because the rows and columns where indexed differently in my array from the user-input.

In terms of memory-efficiency, I used dynamic memory allocation for any array and freed it right after the variable had finished serving its purpose. For example, in the is_winning_move function I had to make a copy of the original board to test whether the given move was a winning move or not. I made sure to free all memory used in this process right after the copy of the board had served its purpose.

There are a few things that could be improved in robustness in the main.c program. One thing I noticed for example is that opening the input file and output file could fail. Therefore, it might be a good idea to handle exceptions when opening a file i.e., by enforcing a try-catch block structure. Additionally, I observed that for the move structure, since it was checked for validity in a different function than the user-input was read, the non-integer input could only be handled as an error which exits the program. This is because if a non-integer input is given when an integer is expected, the behaviour of the program is unpredictable. Had the move instead been validity checked and read in the same function, then the user-input could have been read as a character, then validity checked and only if the input were valid, casted as an integer. Additionally, if the character were not valid, the user could have been asked again and again, instead of the program terminating every time a non-integer input was given.

For the connect4.h header file, robustness could be improved by wrapping the header into a conditional include. This allows the header to be included many times and the definitions are only set if the file has not been seen before, skipping if the header has been seen before. In this way the header is not redundantly read many times in case it is used in multiple files. Moreover, it is not good practice to define a structure in a header. If the definition had only been made in the connect4.c file, then the main file would have no access to the elements in the move structure. However, because the move structure is constructed with user-input from the keyboard, I think it is not necessary to call or define any element of the move structure in the main file.