Reflective Report

Data Structure

For my data structure I used a simple pair of ints to represent the dimensions. Even though there can only be a maximum of 255 columns but an int-bounded number of rows it's simpler and allows larger inputs if so wished in the future
The grid data is stored in a 1-dimensional array (referenced as a pointer to a char). It has length of rows * cols.

Robustness

The principal method of this program is cell and therefore also cell_wrapc. These methods are used to access chars inside the data array of board_structure. The latter of these methods utilises a EUC_MOD (Euclidean modulo) macro to allow for "wrapping the column index value around" i.e., a negative value would wrap back around that many places from the end of the row. There is no support for row index wrapping as it's not needed. I chose to use a macro to remove an unneeded method call for a simple (though, admitted quite ugly) one-liner.

The wrapping of column values allows for very clean and elegant win-detection within the current_winner method. This method is used in is_winning_move, main.c and write_out_file (which admittedly with the current implementation of main.c is a redundant call most of the time). This reuse of code means behaviour is reliable and consistent.

I also re-use code in is_winning_move in that I effectively copy the given board, play the given move on it, check if there is a winner then free that memory. This will of course cause a spike in memory usage (noticeable on larger input files) but allows for consistent behaviour in all uses of the principal methods.

Memory-efficiency

First and foremost, the important element is the grid data stored in board_structure. I made sure to use the absolute minimum amount of memory required while also trying to optimise runtime. I do this by reading the input file twice. During the first pass I just count the rows and columns and use that data to allocate an exactly sized chunk of contiguous memory to store the grid in. Then on the second pass I actually write the characters to said memory. This is superior to a one-pass dynamically re-allocated solution in terms of memory waste (and considering the costliness of realloc; it's quicker too on smaller input files).

I also experimented with using an enum to store the symbols inside the grid but there is seemingly no way in C11 to change the underlying type of an enum from an int (which used 4 bytes as opposed to the single byte a char uses). I could have used a short or something similar but I feel this cuts too much into readability and simplicity.

I have also tried to optimise the data types I have used in given scenarios (like using char over int when feasible etc.).

Finally, sticking to theme of arrays. The only other array in the program lies within current_winner, holding exactly 4 elements which is the current line we are testing. This is re-used every cycle and never redeclared.

Improving robustness of provided code

1. I would remove the requirement of “capitalising the winning line” to be in write_out_file as it seems more fitting to have that functionality in current_winner where the winning is actually found. Also this method is ran before terminating the program in main.c anyway, it just means test.c will fail.
2. Use a global variable in main.c to keep track of the current player. This removes the need to constantly check the board with next_player (only need to do it once at load time)
3. I would index the columns and rows from 0 as opposed to 1 to improve readability.
4. *I’m not sure if I have the wrong files but it seems is_winning_move is never used in any file whatsoever so firstly I would remove that from connect4.h*