Mention again 1st project in c and feedback on implementation techniques regarding c would be appreciated

Difficulty getting box\_num calculations to work, eventually realised needed to do calculation as int to truncate decimals

memory allocation code adapted from (No name or date, Multidimensional arrays in C, <https://www.uio.no/studier/emner/matnat/ifi/IN3200/v19/teaching-material/multidimarrays.pdf>)

started by building and initialising grid.

Worked on code to display grid, made adaptive so just need to change SIZE in one place and the correct board would be created.

Split up box calculation and used to calculate when new box either horizontally or vertically.

Created partially\_complete method, managed to fill correct boxes with same int for all sizes of board.

Considered how to generate unique random numbers to fill each box, created doubly linked circular list “create\_box\_num\_list” method to store all numbers with idea to remove each number as selected. Used circular so do not need to worry about where the list pointer is pointing, and just move the pointer along in list by a random number to max number of elements in list -1 (no point looping through to point back at the same element). If-loop for moving pointer, only moves pointer when there are greater than 1 element remaining so not to waste time by moving pointer to point at itself.

Had difficulties with free(temp), it hung when it was in if loop for updating pointer.

Found formatting required adjustment for when double digits get inserted in the grid (grid size > 9)

After creating create\_box\_num\_list function and node structure, realised could create a structure to hold all the candidates for rows, columns and boxes. This could then be used to complete / solve the board. It would also remove the requirement for a check\_valid method, as everything inserted would have to be valid if its still a candidate (just look for matches between the row/box/column candidates). It would also reduce the time to compute as now not just trying any number, but only candidates.

Function created to populate candidate structures for each box/row/column

Rewrote partially\_complete method to use the candidate structures and remove candidates as entering each into grid.

Was using circular doubly linked list so didn’t need to worry where pointer was in list during shuffling for population of non conflicting boxes on the board. However when find\_match function, realised this meant it wasn’t possible to check if number in list was greater than number comparing to, as could already have started ahead of it. This would lead to more comparisons than necessary. So rewrote functions so candidates list was now a non circular doubly linked list. The candidates structure will now always be pointing at the smallest element in the list.

Wrote find\_match function, originally was calling set\_order\_to\_compare to find shortest, mid and longest lists from within function. However realised this will make it difficult to find ALL matches later on when trying to solve. Now set\_order\_to\_compare will be called externally prior to calling find\_match, and the lists can be passed to the find\_match function. This will allow the pointer to the shortest list to be advanced before passing so as to be able to utilise the function to find ALL matches.

Lots of “Fun” dealing with pointers and passing around functions.

Decided to use a structure with two arrays, and a variable to point to the top of the arrays similar to a stack to store empty cell grid references rather than a doubly linked list. Each empty cell does not need to know where the next or last is. Just need to get them in order. This can also be used to store references to populated cells when trying to remove values for the player\_grid, however the obtained cell reference in the list will be random between 0 and the top.

Stack of empty cells created.

Wrote functions for fill\_first\_empty and solve. Had an issue where set\_order\_to\_compare function was seemingly changing values within the empties list. After a lot of hunting, it turned out to be due to me not allocating memory for the empties stack. Once I fixed this, the grid was populated as expected, actually, a lot faster than expected. Next task, start removing numbers and check still solvable (that might slow things down)

Memory allocation for grid moved into function. Had trouble with pointers when moving it in.

Wrote function to copy grid (so as to have a solution and player grid, potentially later a 3rd to tell which are original values, and which are user values).

Considered logic of removing numbers from grid to create player\_grid. Decided against trying to do it symmetrically or to start in corners/centre, as want to check if solvable for any matches except the number just removed. If solvable, means there is more than 1 unique solution. Due to needing to check this after each number removed, means there isn’t much point in starting with corners/centre, might as well remove randomly from entire grid.

Was considering using structure to store empties to also store populated cells. However wanted to obtain the grid reference at random. Storing like this would mean having to shift all the elements up each time removed one. After lots of consideration, decided to use a similar method to partial\_complete. Will store elements as a circular doubly linked list where each node contains the row and col reference. These can be stored in another structure containing the first access node and the number remaining nodes. This allows the list to be shuffled by updating the pointer a random amount (up to the number of nodes).

Will also store frequency of clues remaining in an array. Index is key-1 (ie for grid size 9, keys are numbers 1-9 minus 1 to obtain index). Value stored will be the frequency of numbers remaining. This can be used to ensure that at least 1 clue remains for SIZE-1 (all except 1 number) at all times.

Rem\_grid\_nums function written. Works fine up until around 50 numbers to be removed for 9x9, then becomes very very slow, and gets slower for each number. Found was slow due to a lot of backtracking in some cases. Put in a timer to leave numbers in place in the grid and remove from the list of grid references to check when taking too long to check if after removal, there is an alternate solution. Tested various timings with 9x9 and 16x16 grids.

Noticed when trying to remove maximum amount of numbers, was outputting a grid with multiple solutions. Discovered issue with how I was updating the empties top in the rem\_num function (needed to reduce original top by 1 when solvable and not use the empties list from solve function)

Rem\_grid\_nums potential slow performance areas:

Updates access node – comparison check on each number removed

Wrote user interface program to generate and parse menus.

Had difficulty linking the programs together, discovered I had #include “sudoku.c” instead of “sudoku.h” at top of sudoku file.

Added functionality to get and check user input then set grid values.

Added colour aid setting which will display correct values in green and wrong values in red when on by checking values against the original player grid.

Added functionality to check when game is complete by keeping a counter starting at the value of removed numbers from the grid. Once all cells have been filled, the original player grid is checked against the player grid. If completely matching, a Boolean solved is set to true to exit a while loop.

Found intermittent bug which outputs a non solvable grid. The solution grid does not get fully populated and the function fill\_first\_empty fails (which it should never do as there should always be a solution). Managed to reproduce the error when setting the random seed to 500 for a 4x4 board. Not found a fix yet, so permanently set seed to 1 instead of time(0), meaning grids will of the same size/difficulty will be identical. Cannot reproduce when seed is set to this. Empties cell\_ref struct looks good. Further investigation required. Random numbers used in rem\_grid\_nums and partially\_complete functions only. Suspect partially\_complete function or memory allocation issues due to fill\_first\_empty failing to find a match when building solution grid.

Was using following in play\_game function in user\_interface to view status of all grids (when also setting a solved=true flag to jump over the actual solving of the grid).

printf("solution:\n");

display\_board(solution\_grid);

printf("original:\n");

display\_board(original\_player\_grid);

printf("num to remove: %d\n", num\_to\_remove);

printf("empties: %d\n", grid\_empties);

OR add to top of generate\_board and set seed to 500 to test locally just in that .c program:

// global variables

int SIZE= 4;

bool timer\_setting = false;

bool colour\_aid\_setting=true;

clock\_t timer;

int \*\*solution\_grid;

int \*\*player\_grid;

int \*\*original\_player\_grid;

int BOX\_ROWS; // rows per box - 9x9 board will be 3, 6x6 will be 2

int BOX\_COLS; // cols per box - 9x9 board will be 3, 6x6 will be 3

int num\_to\_remove= 6;

bool solved=false;

int grid\_empties;

int main(void){

    generate\_grids();

    display\_board(solution\_grid);

    return(0);

}

**TO DO**

Still need to put in Timer functionality,

Access to menu during game (and change whats accessible)

Save game

Save completed games

Load game and continue

Watch playthough

Need c makefile

Order functions in all programs to make logical sense when looking through. Review variable names.

Review header files (move things back to original files??)

Compile with:

gcc sudoku.c user\_interface.c generate\_board.c -o sudoku

run with:

.\sudoku.exe

Ideally would like to simplify following functions. Due to lack of time, was unable to go back and clean up the code.

Rem\_grid\_nums

Solve

Fill\_first\_empty\_cell

All three functions are similar, but do things slightly differently. Was working on functionality first and was planning on going back and extracting methods for identical sections. Would also be nice to split up program files even further to produce cleaner and more readable/understandable code.