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 potential slow performance areas:

Updates access node – comparison check on each number removed