Sudoku Puzzle Solver

05/13/2020

Question:

The objective of this final lab task is to develop one of two alternative data structures to support the solving of Sudoku puzzles, and to investigate their efficiency.

Data Structure used:

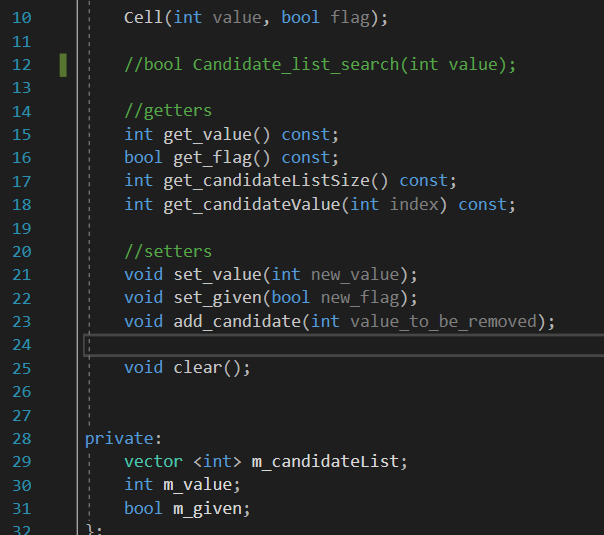
For my sudoku puzzle solver I used the second data structure that involved the use of an object array of pointer cells.

PC’s clock speed:

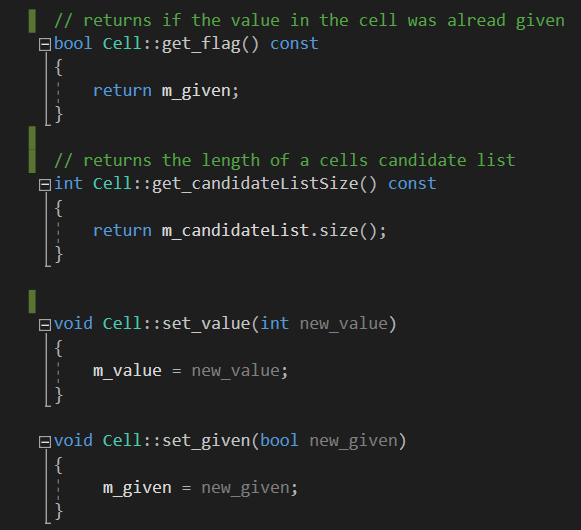
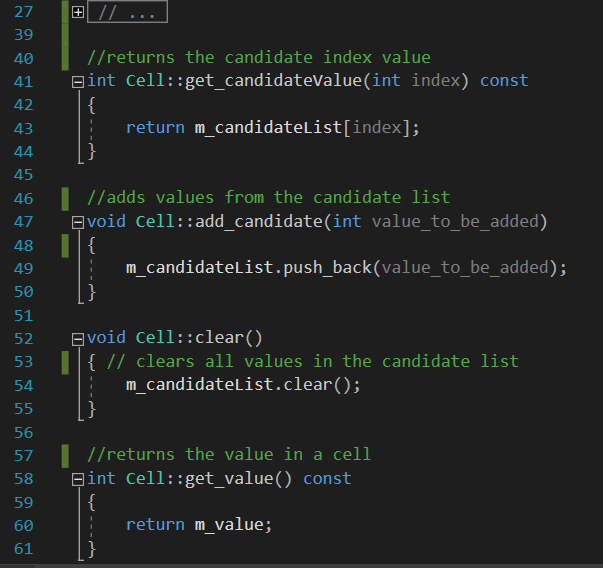
2.30GHz

Solution:

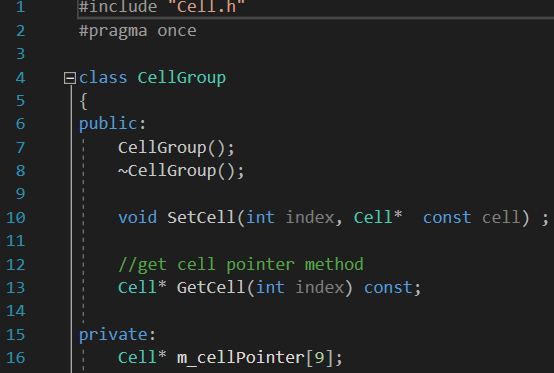
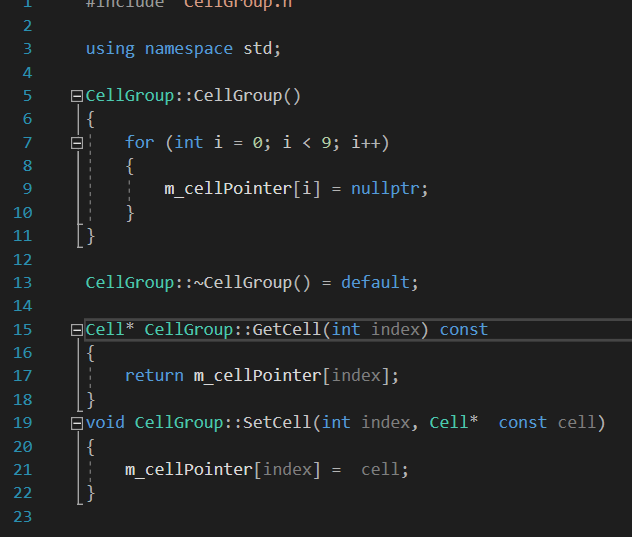
Cell object:



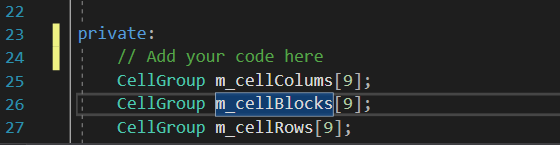
The first thing I did was create a class to hold the value of each cell value. The cell object would hold a data member for the integer value that it contains called m\_value, a Boolean member called m\_given to show true if the value in cell came from the puzzle being solved or if it was already given from the data file. Lastly a vector candidate list to hold the cells candidate list.



All member variables needed to be accessed by cell functions so I created mutators and accessors for all of them. I would also go on to create functions to get a cells candidate lists size and add values to a cell’s candidate list. On line 41 it shows my function that I would use to return the index of a candidate value

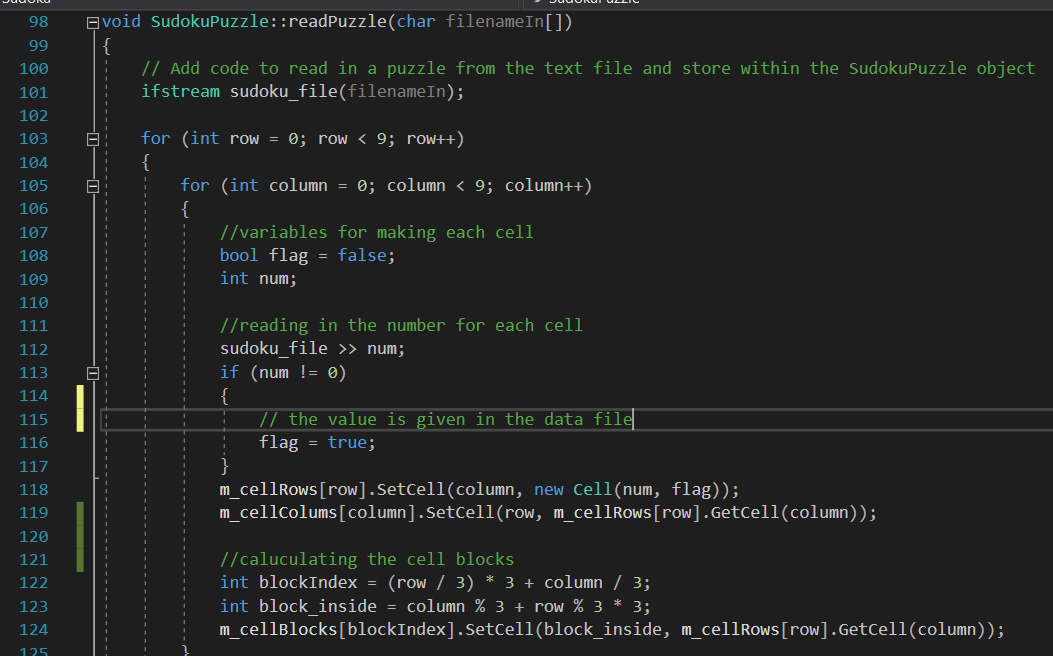
Cell group:

From the images above you can see my Cell Group class. On line 16 of the .h file you can see my cell pointer array that will be used to point to nine cells that are rows, columns and cell blocks. On the cpp file, I have to functions for getting and setting my cell pointer array. This allows me to use the pointer to point to the cell objects.

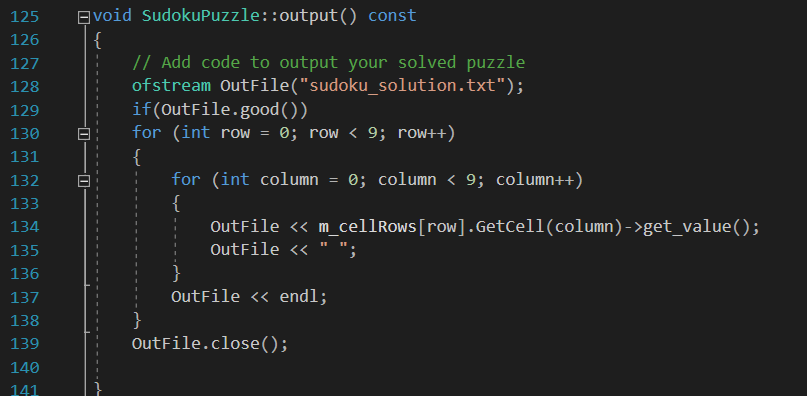


After I had made the cell group class, I made an object of cell group array to point to nine column cells, nine block cells, and nine cell rows.

Read and write file:

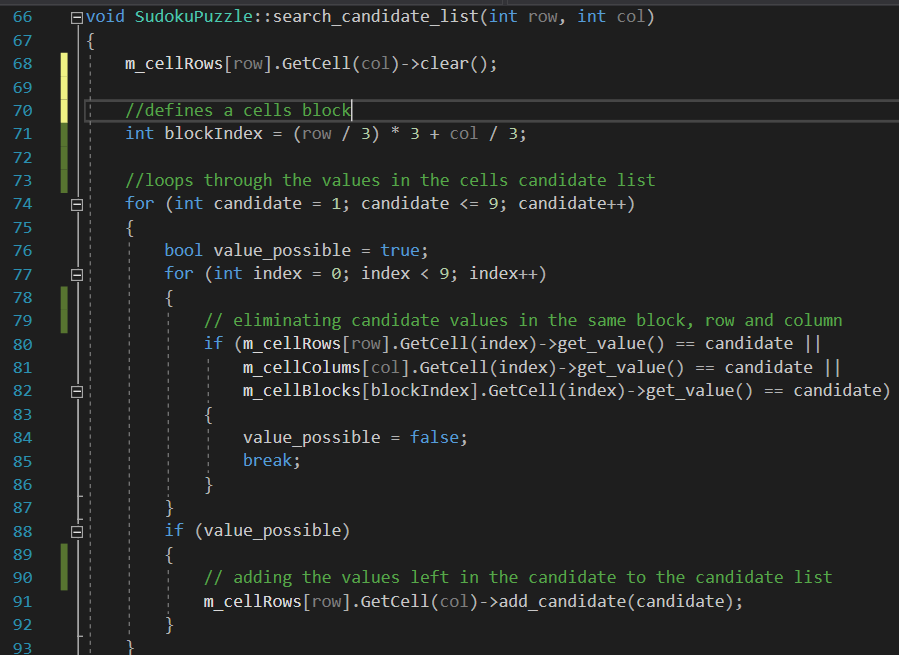


In the image above you can see how I read the unsolved puzzle file into the column row and block pointers.



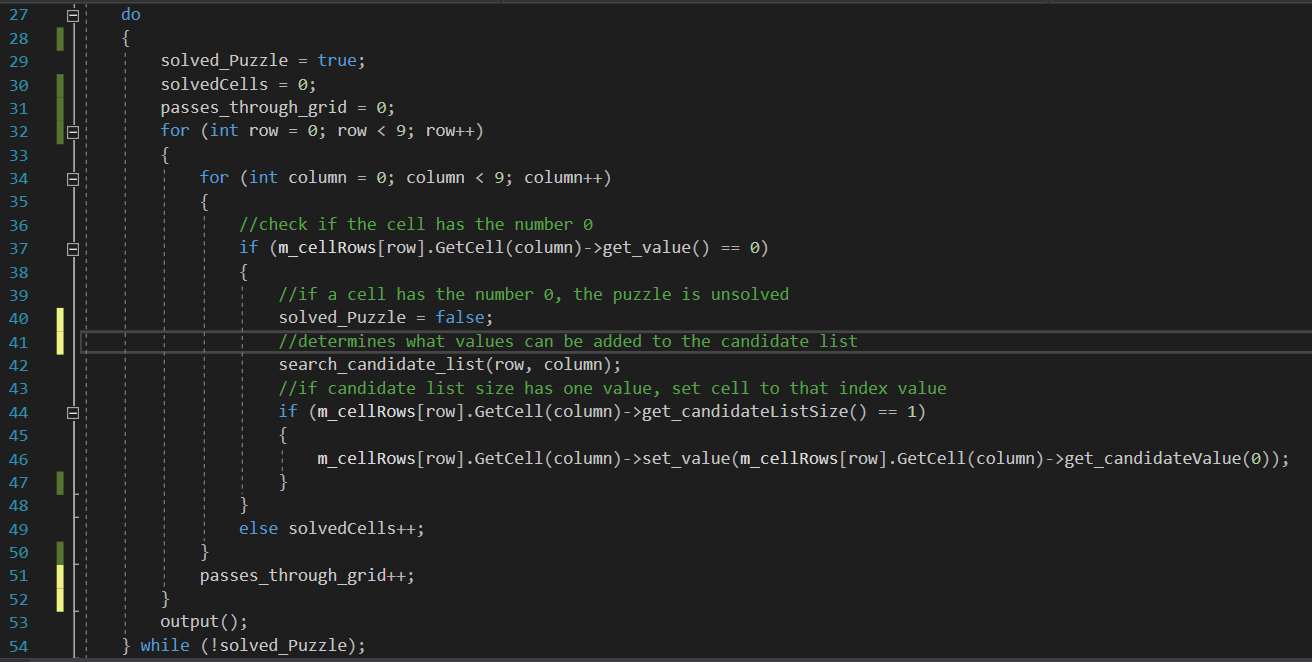
Inside the sudoku puzzle cpp file, in the output function, I wrote the code that would create a new text file to output completed puzzle solution.

Solving the puzzle:



From the screenshot above you can see the function that I used to check the triple context and calculate what values aren’t valid in a cells candidate list. This function works by initially clearing values the pointer array of rows is currently holding. It next calculates that cells block index. Next, it loops through the values of cells candidate list to see which values between one to nine occur in that cells row, column or block. The values are not in the cells row, column or block now get added to the

cells candidate list.



From the image above you can see the do-while loop I used to solve the puzzle. Inside the do-while loop, I created two for-loop that would loop through the whole puzzle and identify which cells need to be solved. For each unsolved cell in the puzzle, I used my “search candidate list” to narrow down their candidate list. If at any point in time, the candidate list of an unsolved cell had one value, I set the cell value as the candidate list value. After each unsolved cell is calculated, I add one to the counter of solved cells. I also add one to the counter after each iteration through the for loop to see how many times it passes through the grid. The do-while loop stops once there are no more zero’s in the puzzle.

Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Cells solved | Passes through grid | Average time in seconds for x86 | Average time in seconds for x64 |
| Puzzle 1 | 54 | 9 | Debug: 0.032587  Release: 0.021595 | Debug: 0.042422  Release:0.081892 |
| Puzzle 2 | 54 | 9 | Debug: 0.032523  Release: 0.022054 | Debug: 0.032924  Release: 0.022095 |
| Puzzle 3 | 54 | 9 | Debug: 0.042501  Release: 0.044150 | Debug: 0.028301  Release: 0.019292 |
| Puzzle 4 | 54 | 9 | Debug:0.030673  Release:0.016840 | Debug: 0.024506  Release: 0.016510 |
| Puzzle 5 | 54 | 9 | Debug: 0.051957  Release: 0.016663 | Debug: 0.018277  Release: 0.016187 |

Data Structure Evaluation:

For the implementation of my data structure, I went with the pointer approach instead of the 2D array. Pointer store the address of a variable while arrays store a list of the actual variable. Pointers are dynamic meaning their size can be changed as well as their contents a lot easier than arrays which are static and cannot be easily resized. Pointers make it easier to access memory directly. Since pointers are dynamic in size, it is likely to lead to reduced execution time when compared to using an array. Due to the nature of pointers, it provides multiple ways to access a specific cell and to change values in that cell. This

However, despite the fact that pointers are more memory efficient than arrays. They are harder to implement and can be more dangerous to work with. When designing my program, I often encountered situations early on where my pointers would update a cell with wrong values. This were extremely difficult to debug as I struggled to find where the memory mishandling was coming from in my program.