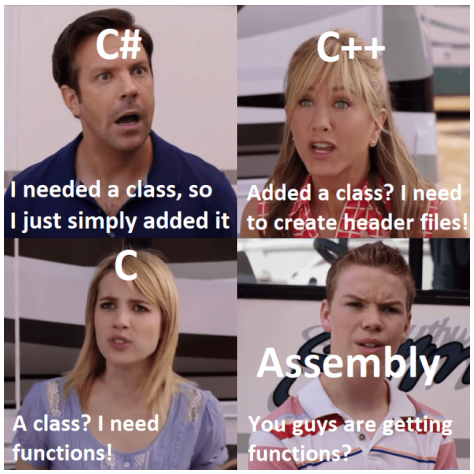
Sukoku Solver

CSEE 3827: Fundamentals of Computer Systems

Take Home Programming Test

Prof. Martha Kim

- All work is to be i laboration of any sort permitted.
- You may refer to al urse materials (i.e., slides, notes, problem sets and solutions, discussion board h
- You may also refe s (e.g., slides from other courses or semesters, the Harris and Harris textboo ls should serve as references and their content should never be presented as yo
- You may not solicit or consult solutions written by another human.
- AI-based coding assistants such as ChatGPT or Github Copilot are prohibited.
- Academic integrity violations will be reported directly to the Office of Student Conduct, and students found responsible are subject to academic penalties.
- You may pose questions to course staff via private post on ED. As on an in-person test, staff will answer clarifying questions about the prompt, but will not provide assistance writing or debugging code.
- Your code will be tested for both correctness and adherence to calling conventions.
- when possible, partial credit will be given.
- The submission deadline is Tuesday November 21 at 11:59pm. Be sure to leave sufficient time to upload before that deadline, as late submissions will not be accepted.
- To submit, upload a single file named sudoku.s to gradescope.



For this assignment you will implement parts of a Sudoku puzzle solver.

1 Background on Sudoku

Sudoku is a number puzzle played on a 9x9 grid. In a correctly solved puzzle, the digits 1 through 9 appear exactly once in each 3x3 box in the grid. Each puzzle starts with some of the cells completed, and the objective is to find the unique configuration of digits that fills the grid and satisfies the box constraints. For a more detailed explanation, see the Wikipedia page on Sudoku: https://en.wikipedia.org/wiki/Sudoku.

2 Solving Tactics

There are a number of Sudoko solving tactics that a solver might employ. The solver for this assignment implements only the most simple tactic: once a cell is solved (i.e., its digit is known) that digit can be ruled out of the row, column, and box containing the cell. The solver will apply this tactic to all solved cells in the board until all 81 cells in the board are solved.

3 Sudoku Cell Representation

We represent each cell as a null-terminated ASCII string in memory. The string is nine characters long, followed by the null character terminal, so each cell occupies ten bytes in all. The contents of this string list, in ascending order, the possible correct digits for that cell. So, a cell where any digit is possible is represented as:


```
123456789
```

When a digit is ruled out as a possible answer for a cell, the corresponding entry in the cell string is replaced with a period. So, if we were to rule out 3 as a possible answer to the cell above, the updated cell string would be:


```
12.456789
```

A cell is solved when all but one digit have been ruled out. For example, a cell that is determined to be 5 would be represented as:

```
....5....
```

4 Sudoku Board Representation

The 81 cells of the Sudoku board are arrayed in memory, one after the other. They are ordered in row-major order, meaning that all of the cells from the first row come first, followed by the cells in the second row, then the third, and so on until the 9th row. Within a row, cells are ordered from left to right. With each cell occupying 10 bytes, the whole board occupies 810 bytes of memory.

5 Functions to Implement

There are four functions to be implemented. Each of these four functions is delimited in the scaffolding by a separator line that begins

```
#### Do not move this separator.
```

Do not remove any c s from your submission. When implementing a function, make sure that all of you on appears between the separators for that function. If you call one of the provided u do not need to relocate those functions to between the separators.

num_candidates

Given a pointer to a cell, this function should return an integer between 0 and 9 (inclusive) indicating how many candidate digits remain for the given cell. The function should leave the string that represents the cell unmodified.

rule_out_of_cell

This function takes two arguments:

- a pointer to a cell

- an integer indicating the digit to be ruled out of the cell, guaranteed to be a valid Sudoku digit, i.e., 1-9 inclusive

The function should update the cell string to eliminate the given digit as a candidate. If the given digit has already been ruled out, the cell string should not change.

count_solved_cells

Given a pointer to a Sudoku board, this function should return an integer indicating how many of the cells in the board have been solved. The state of the board should be unmodified.

solve_board

Given a pointer to a Sudoku board, this function should solve the puzzle, iteratively applying a single solving tactic: for each solved cell, rule its digit out of the corresponding row, column and box. Apply this tactic until all cells in the board are solved.

Note that there are helper functions in the scaffolding. They are described in more detail in the following section, but rule_out_of_row, rule_out_of_col, rule_out_of_box will be particularly useful for solve_board.

solve_board should solve and modify the original board in place, so that when the function is complete, the solved board is found at the same address as the original board.

6 Helper Functions

In the scaffolding you will find a number of helper functions. You don't need to use all of them, but we describe all of them here for your convenience:

- print_int: given an int, prints it to the screen
- print_string: given a null-terminated ASCII string, prints the string to the screen
- print_newline: prints a newline to the screen
- print_space: prints a space
- print_hsep: prints a horizontal line of dashes to the screen
- print_vsep: prints a vertical pipe character to the screen
- print_board: given a pointer to a board in memory, prints it to the screen
- first_candidate: given a pointer to a cell, returns the value of the first possible digit; if there are no possible digits in the cell, returns 0
- get_row_base: given a pointer to a board and a pointer to a cell in that board, returns a pointer to the base of the row in the board that contains the given cell
- get_col_base: given a pointer to a board and a pointer to a cell in that board, returns a pointer to the base of the column in the board that contains the given cell
- get_box_base: given a pointer to a board and a pointer to a cell in that board, returns a pointer to the base of the 3x3 box in the board that contains the given cell
- This function will work once num_candidates is working:
  - is_cell_solved: given a pointer to a cell, returns 1 if cell is solved, 0 otherwise
- These three functions will work only once count_solved_cells and rule_out_of_cell are working:
  - rule_out_of_row: given a pointer to a board and a pointer to a solved cell in that board, rules the digit in the solved cell out of the row containing the solved cell
  - rule_out_of_col: given a pointer to a board and a pointer to a solved cell in that board, rules the digit in the solved cell out of the column containing the solved cell
  - rule_out_of_box: given a pointer to a board and a pointer to a solved cell in that board, rules the digit in the solved cell out of the box containing the solved cell

7 Testing and Advice

- As in PS4 and PS5, the provided main function makes test calls to the functions to be implemented. The expected output for each of these calls is specified in a comment in the body of main. If you are having trouble isolating an error, it helps to comment out or excise all but the one test invocation you are trying to debug.

- Additional helper functions beyond what is provided in the scaffolding are not advised; if you do write them, they should adhere to conventions and appear in between the same separators as the function that uses them.
- It is recommended that you implement solve_board only after the first three functions are working.