Assignment No. 6
EECS 210
Discrete Structures
Due: 11:59 PM, Thursday, November 9, 2023
Submit deliverables in a single zip file to Canvas
Files in other formats (e.g., .tar) will not be graded
Name of the zip file: FirstnameLastname_Assignment6 (with your first and last name)
Name of the Assignment folder within the zip file: FirstnameLastname_Assignment6

Deliverables:

1. Copy of Rubric6.docx with your name and ID filled out (do not submit a PDF).
2. Source code.
3. Screen print showing the successful execution of your code or copy and paste the output from a console screen to a Word document and PDF it.

Assignment:

- You may use any language you want, but if you want help from me or one of the SIs, you should probably use C++, Python, or Haskell.
- Topic: Recursion
- Sudoku is a number puzzle in which a 9x9 grid is subdivided into 9 3x3 sub-grids, or blocks.
- A typical puzzle is shown here (image credit: Wikimedia Commons):

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

- Each of the cells in the puzzle must be filled in with a single digit 1-9, subject to the following constraints:
    1. Each digit from 1 to 9 must appear exactly once in each row.
    2. Each digit from 1 to 9 must appear exactly once in each column.
    3. Each digit from 1 to 9 must appear exactly once in each block.
- Published puzzles are usually constructed to have a unique solution. This requires that at least 17 cells be filled in (and may require more; depending on the puzzle and which cells are empty, as many as 78 filled cells can be required to guarantee uniqueness). In general, the number of cells that are already filled in is not a particularly good indicator of how difficult a Sudoku puzzle is to solve.

- The Sudoku problem, then, is this: Given a partially completed Sudoku puzzle, find the (or at least a) solution; if no solution exists, prove that.
- It is possible to solve a Sudoku puzzle entirely through logic. For example, in the above puzzle, consider the bottom right block, top right cell. Because of other numbers in the same row, column, or grid, that number cannot be 1, 2, 3, 5, 6, 7, 8, or 9, and so must be 4. By proceeding in this fashion, using these and more advanced deduction rules, it's possible to gradually fill in the entire grid.
- That is not the approach we're going to use. Instead, we're going to implement a brute-force search, in which we exhaustively try possibilities until we find a solution. This will be slower to run but is easier to solve. (And by "slower to run," we mean it'll finish in a few tenths of a second rather than a few milliseconds.)
- Here's the idea: Suppose we have a Sudoku puzzle of which k of the 81 cells are unfilled. We identify an empty cell and deduce a list of candidates for what might go into that cell. If there are no such values (all digits are accounted for elsewhere in the same row, column, or grid), then we must have made a mistake in filling in the k cells and return a no-solution-found result. Otherwise, we try putting the first candidate value into the empty cell, and see if we can find a solution with the k+1 cells filled (i.e. a recursive call). If we get a no-solution-found to the first candidate, then we try the next, and so on, until we either find a solution (k == 81) or determine that no solution exists (all candidates report no solution, which again implies we took a wrong path somewhere earlier).
- This approach, by the way, is called depth-first search with backtracking. For each empty cell we make an attempt and see how far we can get with it. If we fail to find a solution, we back up until we find something else to try, and so on. If a solution exists, we are guaranteed to find it, because we exhaustively try all possibilities.
- You are given five files (puzzle1.txt through puzzle5.txt), each containing a Sudoku puzzle. Digits already filled in have been placed, and an underscore character (_) is used to indicate a blank square.
- You may read the files in or just hard-code them in the program.
- Write a **recursive** program to find the solution to the five puzzle files.
- The output for each solution, should include:
    1. The puzzle file name (e.g., puzzle1.txt)
    2. The puzzle from the file printed out.
    3. The solution to the puzzle printed out or if no solution is found, "No solution found" printed out.
- The solution may not be unique; if there is more than one solution to the puzzle, any valid solution is acceptable.
- To help you debug your program, solution1.txt is a solution to puzzle1.txt.

Provide comments that explain what each line of code is doing. See rubric below.

| Rubric for Program Comments | | |
|---|---|---|
| **Exceeds Expectations (90-100%)** | **Meets Expectations (80-89%)** | **Unsatisfactory (0-79%)** |
| Software is adequately commented with prologue comments, comments summarizing major blocks of code, and comments on every line. | Prologue comments are present but missing some items or some major blocks of code are not commented or there are inadequate comments on each line. | Prologue comments are missing all together or there are no comments on major blocks of code or there are very few comments on each line. |

Adequate Prologue Comments:

- Name of program contained in the file (e.g., EECS 210 Assignment 3)
- Brief description of the program, e.g.:
    - Python code for demonstrating operations on relations and properties of relations.
- Inputs (e.g., none, for a function, it would be the parameters passed to it)
- Output, e.g.,
    - Print out of the name of each exercise, followed by the exercise's output.
- All collaborators
- Other sources for the code ChatGPT, stackOverflow, etc.
- Author's full name
- Creation date: The date you first create the file, i.e., the date you write this comment

Adequate comments summarizing major blocks of code and comments on every line:

- Provide comments that explain what each line of code is doing.
- You may comment each line of code (e.g., using //) and/or provide a multi-line comment (e.g., using /* and */) that explains what a group of lines does.
- Multi-line comments should be detailed enough that it is clear what each line of code is doing.
- Each block of code must indicate whether you authored the code, you obtained it from one of the sources listed in the prolog, or one of your collaborators authored the code, or if it was a combination of all of these.

Collaboration and other sources for code:

- When you collaborate with other students or use other sources for the code (e.g., ChatGPT, stackOverflow):
    - Your comments must be significantly different from your collaborators.
    - More scrutiny will be applied to grading your comments in particular explaining the code "in your own words", not the source's comments (e.g., ChatGPT's comments).
- Failure to identify collaborators or other sources of code will not only result in a 0 on the assignment but will be considered an act of Academic Misconduct.
- Students who violate conduct policies will be subject to severe penalties, up through and including dismissal from the School of Engineering.