ENGG1110 Problem Solving by Programming
Fall 2023

Project Specification (Version 1.3)

Due Date: 11:59 PM, 15 December (Friday), 2023

Changelog:

v1.3 Describe what to do if loadPuzzle() returns 0. Emphasize the player will only input an alphabet/digit (not space/newline) when inputting a guess (inputBoard()). Specify writing your program explanation in the source code file. Removed a extra space in textInput_3.txt

v1.2 Coordinate (x, y) is to be displayed with space after the comma. Full stop is displayed at the end of all congratulation messages.

v1.1 Fix sample output in p.8. [main.c] Removed the commented (Line 58) in inputBoard().

1 Introduction

Sudoku is a number-placement logic-based puzzle. The objective is to fill a 9×9 grid with digits $1, 2, \ldots, 9$. Each row, column, and nine 3×3 neighboring subgrids forming the 9×9 grid must compose all digits 1 to 9 without repetition. A Sudoku puzzle is a grid partially filled with clues. Figure 1a and 1b shows a puzzle with 24 clues and its solution. In this project, the given puzzles are assumed to be proper, i.e., puzzles that have *one* unique solution.

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

(a) Sudoku puzzle with 24 clues.

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

(b) The only valid solution for Figure 1a .

Figure 1: A Sudoku Puzzle

In this project, you are going to make a single-player Sudoku Game in C programming language with additional features.

You are required to complete the given source file **main.c** without modifying any existing code (unless otherwise specified) or introducing new libraries. Marks will be deducted for each modification.

2 Project Part 1: the Barebone (60%)

In Project Part 1, your program should hold a 9×9 Sudoku game with the following rules:

1. Start with a 9×9 game board filled with clues (a puzzle is given).

2. In a finished puzzle, each row, column, and the 9 neighboring 3×3 grids must contain numbers $1, \ldots, 9$ without repetition.
3. The player wins the game if Condition 2 is met for the current game board.
    - In Easy mode, the player has an unlimited number of trials and is told whether the guess is correct.
    - In Hard mode, the player loses when it has inputted a number violating Condition 2 for 3 times or causing a board locking situation (Project Part 2).

The program will:

1. allow the player to select Easy or Hard mode.
2. read the puzzle & solution from source:
    - The puzzle is assumed to have only a unique solution (you do not need to check they have a unique solution).
    - The solution is assumed to be valid for the corresponding puzzle.
3. display the Sudoku puzzle.
4. read the player's input and update the game board accordingly:
    - The player will select a cell by inputting (row, column).
    - The selected cell will display an 'X'.
5. tell the player if it inputs a wrong number (comparing to the solution) in Easy mode. In Hard mode, the program:
    - will store the number of chances left (starting at 3).
    - will only check if the number collides with other numbers in that row, column, and subgrid. If so, it consumes one chances,
    - does not support edits (and eventually loses) if an incorrect number is inputted.
6. help fill a cell in the Easy mode.
7. check if the player wins or (in Hard mode) loses.

You must start with the template code provided, and only modify the designated areas.

3 Program Flow

The flowchart of the game running in easy mode is provided in Figure 2.

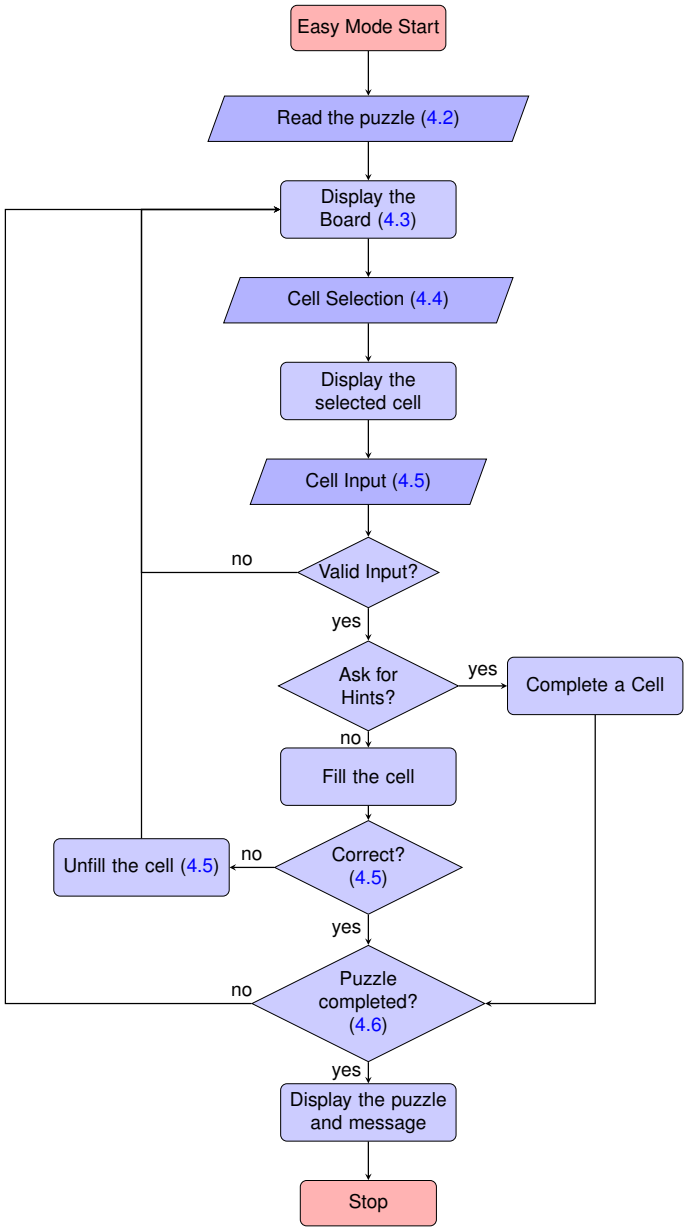

Figure 2: A flowchart depicting the easy mode.

4 Detailed Design

The player's inputs are underscored in the sample outputs.

4.1 Header Files and Libraries

No new header file or library should be added. **You should not modify the function parameters.**

4.2 Reading Game Mode and Puzzle (12%)

The **main()** function starts with game mode selection. In this stage, the player first enters the difficulty level:

- If the choice is 0, the game runs in Easy mode.
- If the choice is 1, the game runs in Hard mode.
- Else[1], the program prints "Invalid Input." and starts over the game mode selection.

```
Enter the game mode [0: Easy. 1: Hard]: 4
Invalid Input.
Enter the game mode [0: Easy. 1: Hard]: 0
You have selected Easy mode.
```

The program reads from the source the puzzle and solution. The puzzle and solution are initialized as two 2D arrays:

- myPuzzle is a 9×9 integer array storing the clues of the given puzzle.
- mySolution is a 9×9 integer array storing the puzzle's solution (the completed puzzle).

Format of Puzzle and Solution

```
   012 345 678
  +---+---+---+
0|5  | 63|4  |
1|   |7  |   |
2|1  | 5 |83 |
  +---+---+---+
3|   | 18|  7|
4|  6|9  |   |
5| 43|   |9  |
  +---+---+---+
6|   |   |   |
7|   | 7 | 2 |
8|32 |64 |5  |
  +---+---+---+
```

Code 1: The board starts counting from top-left $(0, 0)$ to bottom-right $(8, 8)$

A puzzle and its solution are two 2D integer arrays of 81 integers corresponding to the 9 × 9 cells arranged from left to right, top to bottom.

- Integer 1 to 9 represents a clue and a 0 represents an empty cell to be filled by the player.
- For example (the puzzle in Code 1):
  - The 2D integer array for the puzzle is { {5, 0, . . . , 4, 0, 0}, . . . , {3, 2, . . . , 5, 0, 0} }.
  - The 2D integer array for the solution is { {5, 9, . . . , 4, 7, 2}, . . . , {3, 2, . . ., 5, 9, 8} }.
- In both game modes, the same puzzle and solution are used.

You need to complete **initGameBoard()** to initialize the board with the selected puzzle.

[1] If the player inputs a character/string, scanf() would act "strangely" (why?). You may assume the player only inputs integer(s) separated by space and end with newline '\n'.

4.3 Display Sudoku Puzzle (16%)

The program will initialize a 9 $\times$ 9 grid (or the board) following the format (calling **initGameBoard()**), using the puzzle stored in the source. Then, the board will be printed by invoking **printGameBoard()** with the following format (Code 1):

- The board is displayed with row and column numbers.
- The given clues are displayed.
- The 3 $\times$ 3 subgrids are separated by borders.
- The unfilled cells (storing 0) are displayed with the space character ' '.
- The selected cell that stores a specific integer (if exists) is displayed with the character 'X'.

The coordinate of a cell is (x, y) where x is the row number and y is the column number.

4.4 Selecting a cell and inputs (12%)

After a puzzle is selected and displayed, the player inputs row and column number to select a cell. The selected cell is displayed with a character 'X'.

```
Select a row and column: 1 1
  012 345 678
 +---+---+---+
0|5  | 63|4  |
1| X |7  |   |
2|1  | 5 |83 |
 +---+---+---+
3|   | 18|  7|
4|  6|9  |   |
5| 43|   |9  |
 +---+---+---+
6|   |   |   |
7|   | 7 | 2 |
8|32 |64 |5  |
 +---+---+---+
```

Code 2: The player selected cell (1,1) displays an 'X'.

The program will check if the input is valid[1], if it is invalid, it prompts

```
Select a row and column: 0 0
Occupied.
Select a row and column: 10 10
Out of bound. Input Again.
Select a row and column:
```

The program will *keep asking* for the player's selection if the selected cell is occupied or the input format is invalid (e.g., out of range):

- In case of occupied cell, "Occupied." is printed.
- In case of invalid output, "Out of bound. Input Again." is printed.

You need to complete the program segment (in **main()**) to accomplish the cell selection task.

4.5 Checking the Number (16%)

Having the cell selected (row and column index), the program then calls a function **inputBoard()** to read an input from the player and updates the grid:

1. The line “Input a number [or H: hint]: ” will be displayed and the program will read a character input by the player. (You may assume that the player only inputs one character followed by ‘\n’. **This character is either alphabet or digit.**)

2. The program should check if the input is valid: either ‘1’ to ‘9’ or ‘H’ for hint. It goes back to **Cell Selection** (Section 4.4) for any invalid input:

   - if the input is not ‘1’ to ‘9’ or ‘H’, the program outputs "Invalid Input."
   - if ’H’ is inputted and the game is in Hard mode, the program outputs "No hint in Hard mode.".

3. The function updates the game board if ‘1’ to ‘9’ or ‘H’ is inputted.

4. The function returns 1 if a hint is used; -1 if the input is invalid; and 0 if the character is in ‘1’ to ‘9’. You should make use of the return value for the control flow.

In Easy mode, the inputted number will be checked against the solution (with if-else statement in **main()**). The program will tell the player and goes back to cell selection if it has inputted a wrong number. The program will fill in the solution for the player after a correct number or ‘H’ is inputted. It will also count the number of hint used.

Sample Outputs

Assuming the player selected (1,1), e.g., continues from Code 2, and 7 is inputted to the cell:

```
Input a number [or H: hint]: 7
Sorry, 7 should not be placed at (1, 1).
  012 345 678
 +---+---+---+
0|5  | 63|4  |
1|   |7  |   |
2|1  | 5 |83 |
 +---+---+---+
3|   | 18|  7|
4|  6|9  |   |
5| 43|   |9  |
 +---+---+---+
6|   |   |   |
7|   | 7 | 2 |
8|32 |64 |5  |
 +---+---+---+
Select a row and column:
```

and when the correct answer 3 (or ‘H’) is inputted to the cell (1,1):

```
Input a number [or H: hint]: 3
  012 345 678
 +---+---+---+
0|5  | 63|4  |
1| 3 |7  |   |
2|1  | 5 |83 |
 +---+---+---+
3|   | 18|  7|
4|  6|9  |   |
5| 43|   |9  |
 +---+---+---+
6|   |   |   |
7|   | 7 | 2 |
8|32 |64 |5  |
 +---+---+---+
Select a row and column:
```

For an invalid input entered by the player after it has chosen cell (1, 1):

```
Input a number [or H: hint]: a
Invalid Input.
  012 345 678
 +---+---+---+
0|5  | 63|4  |
1|   |7  |   |
2|1  | 5 |83 |
 +---+---+---+
3|   | 18|  7|
4|  6|9  |   |
5| 43|   |9  |
 +---+---+---+
6|   |   |   |
7|   | 7 | 2 |
8|32 |64 |5  |
 +---+---+---+
Select a row and column:
```

(Programmer's) Hard Mode

In Hard mode, the program **will not** check the input number against the solution. Instead, it only prompts whether there is a collision with the numbers in the same row, column, or that subgrid (via **checkSolutionDetail()**). The player is given 3 chances for inputting such number. It is possible that the player's input does not cause a collision, but is in fact an incorrect solution[2].

Depending on the three cases of collision, the program prints the following **in order**:

- "Check again the numbers in row x." if there is a repeated number at the same row,
- "Check again the numbers in column y." if there is a repeated number at the same column,
- "Check again the numbers in the subgrid where (x, y) is at." if there is a repeated number at the same subgrid,

where the underlined x and y are the row and column index for the selected cell. The program deducts the chances if and only if the inputted number causes a collision.

[2]Since we do not allow the player to edit filled cells, the player will eventually lose the game for a single wrong guess.

When **3 wrong guesses** are inputted, the player loses the game. The program prints "You lose." Here is a sample output:

```
Select a row and column: 1 1
  012 345 678
 +---+---+---+
0|5  | 63|4  |
1| X |7  |   |
2|1  | 5 |83 |
 +---+---+---+
3|   | 18|  7|
4|  6|9  |   |
5| 43|   |9  |
 +---+---+---+
6|   |   |   |
7|   | 7 | 2 |
8|32 |64 |5  |
 +---+---+---+
Input a number [or H: hint]: 1
Check again the numbers in the subgrid where (0, 1) is at.
You have 2 chances left. //print "You lose." if depleted
```

4.6 Check Finished Puzzle (4%)

After the player fills a cell, the program will call **checkFinish()** to check if the puzzle is finished. If the puzzle is unfinished, the program will start again from displaying the board, i.e., goes back to “Display the Board (4.3)” in the flowchart (Figure 2).

If the player has finished the puzzle, the program prints:

- "Congratulations! You have finished a puzzle in easy mode and used 5 hints." for a player winning the easy mode with the number of hints used.
- "Congratulations! You have finished a puzzle in hard mode with 3 chances left." for a player winning the hard mode with the number of chances left.

5 Additional Features (40%)

In Project Part 2, your task is to enrich your Sudoku Game with the following features. **Make a copy of your course code for Part 1.** You need to submit another source code for Part 2.

1. Read puzzle from file (15%).
    - The program reads from one text file, puzzle.txt, storing a puzzle and its solution. (The file is assumed to have valid format. Moreover, you should implement error handling for file IO.)
        - The format of the text file is as follows:

```
5 0 0 0 6 3 4 0 0
0 0 0 7 0 0 0 0 0
...
0 0 0 0 7 0 0 2 0
3 2 0 6 4 0 5 0 0

5 9 8 1 6 3 4 7 2
6 3 2 7 8 4 1 5 9
...
9 8 1 3 7 5 6 2 4
3 2 7 6 4 1 5 9 8
```

        - Each line (ends with '\n') contains 9 integers separated by space corresponding to a row in the 9 $\times$ 9 grid.
        - The first 9 lines are the clues for the puzzle.
        - Line 10 is an "empty" line.
        - The next 9 lines are the finished puzzle (solution).
    - The program reads puzzle.txt after the player has chosen the game mode. (This modification does not change the program outputs.)
    - You can remove the hardcoded variable myPuzzle and mySolution.
2. Save and load **one** unfinished puzzle since hard puzzles often take time to finish. (15%)
    - During cell input after cell selection (see Section 4.5), the player can input 'S' to save the current puzzle and solution to saveGame.txt. The program will exit after saving the game:
        - The selected cell needs not to be saved.
        - You need to determine what to save to saveGame.txt (check also the FAQ in Project folder on BlackBoard).
        - You should modify **inputBoard()** to return another value representing game saving.
    - Complete **savePuzzle()** that saves the game board and solution to saveGame.txt
        - It prints "Game Saved." and exits the program.
    - The program asks the player whether it shall load the saved game saveGame.txt before the game mode selection.
        - Your program will print "Load the saved game (y/n)?" and reads a character from the player.

  - The player inputs[3] 'y' to load a saved game (or 'n' to not load the saved game).
  - The program proceeds to game mode selection, and reads from saveGame.txt if the input is 'y' (read from puzzle.txt if the input is 'n').
- Complete **loadPuzzle()** that takes as input the game board to load the puzzle in the file to the game board. You can (and shall) terminate the program if it fails to read the puzzle (e.g., fail to open the file).

3. A locked board checker added to Hard mode. (10%)
   - In Hard mode, the program checks if the board is locked, i.e., there exists a cell in which no valid number can be inputted after a cell is filled. You need to complete the function **isLockBoard()** which outputs 1 for a locked board:
     - The function checks for each *non-empty* cell that the set of possible answers is not empty.
     - Inside **isLockBoard()**, it utilizes a function **checkFillable()** that finds the set of possible answers for the cell (given the coordinate (x, y)) in the game board.
     - **checkFillable()** returns 0 if the set of possible answer is empty, i.e., cell (x, y) is a (locked) cell that cannot be filled, and returns 1 otherwise.
     - **isLockBoard()** returns 1 if there exists a locked cell, and 0 if the board is alive.
   - After filling each cell, the program should call **isLockBoard()** to determine if the player loses the game. It outputs "Cell (x, y) is locked.", "Board is locked.", and the losing message after finding the first locked cell.

6 Sample Run

5 sample runs are provided (3 for Project Part 1 and 2 for the locked block checker). 5 sample input files are also provided. You are recommended to design your own test cases to debug the program.

7 Describe Your Approach

Please also **write as comments in your main program** near the related code segment(s) or after the declaration comment block to describe your approach/code for the following:

- Board checking for checkSolutionDetail().
  (a) How do you check collision in row and column?
  (b) How do you check collision for the 3×3 subgrid for the selected cell?
- Implementation of the save/load function.
  (a) How do you save the puzzle as a file, e.g., the format?
  (b) Where (and how) do you add the load option?
  (c) Can you use **loadPuzzle()** to read puzzle from other text files? Describe how or why not.

[3] You may assume the player inputs one character and need to apply input check.

- Locked Block Checker.
  (a) How do you find the set of possible numbers for an empty cell in checkFillable()?
  (b) How do you make use of the function checkFillable()?
- Any extra features/functions you have implemented.

Marks will be deducted for each item missed.

8 Schedule

Suggested schedule after the project release:

Week	Part
Week 1,2	Finish the easy mode (Section 4.2 to 4.6) • Parsing puzzle and solution • Print the game board • Cell Selection and cell input checks • Checking the input number is correct • Check if the board is finished and prints the number of hints used
Week 3	Extend the easy mode to hard mode: • Complete checkSolutionDetail(). • Add condition for game losing.
Week 3+	Copy your source file for Part 1 and modify it for project Part 2 (Section 5): • Implement the locked board checker (e.g., isLockBoard(), checkFillable()). • Load the puzzle and solution from the given file. • Implement the save/load function

9 Academic Honesty and Declaration Statement

You are prohibited from using any AI tools to complete your assignments, assessments and any other works that count towards their final grade of the course or attainment of the desired learning outcomes.

The Chinese University of Hong Kong places very high importance on honesty in academic work submitted by students, and adopts a policy of zero tolerance on academic dishonesty. Any related offence will lead to disciplinary action including termination of studies at the University. Details may be found at https://www.cuhk.edu.hk/policy/academichonesty/.

Tools such as software similarity measurement, AI-tool fingerprint detection, etc., might be used to inspect all submissions. Every suspected offending case will be reported to the Faculty.

You are required to fill in the declaration in the comment block at the beginning of each of the source files you submit.

```
/**
* ENGG1110 Problem Solving by Programming
*
* Course Project
*
* I declare that the project here submitted is original
* except for source material explicitly acknowledged,
* and that the same or closely related material has not been
* previously submitted for another course.
* I also acknowledge that I am aware of University policy and
* regulations on honesty in academic work, and of the disciplinary
* guidelines and procedures applicable to breaches of such
* policy and regulations, as contained in the website.
*
* University Guideline on Academic Honesty:
*   https://www.cuhk.edu.hk/policy/academichonesty/
*
* Student Name  :
* Student ID    :
* Class/Section :
* Date          :
*/
```

10 Grading Platform and Submission

You should use Code::Blocks (mingw32) and we will test on Windows 10 (or 11).

- All the materials must be submitted via BlackBoard.
- You need to submit **two source code files**:
  - projectPart1_<Student ID>.c for project Part 1 AND
  - projectPart2_<Student ID>.c for project Part 2.
- Submit all files to the submission place on BlackBoard.

Late submission within 72 hours will result in a mark penalty of -10% per 24 hours, and a 50% mark penalty for late submission between 72-144 hours. No mark will be given for late submission more than 144 hours.