Fortify Source Code Analyzer Software Engineer

Candidate Homework

* *The code must be complete and must be placed in some public github repository for us to review*

# Project 1

You will define, implement and test a Sudoku solver.

**Sudoku rules:**

The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contains all of the digits from 1 to 9.

**Sudoku examples:**

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

**Requirements**

* **Write an architecture document explaining the choices you have made to implement this solver.**

Answer: 2D matrix of size 9x9 is given , which is partially filled and the goal is to fill its empty cells satisfying the given condition.

Two Utility has been implemented to achieve the purpose. Main utility is sudokusolver which is used to fill the cells and isValid utility is used to check whether the number is valid for that particular cell or not.

* **Implement the solver in C or any other language.** 
  + **Document the issues you have encountered and how you resolved them**

Answer : The issue was in handling the edge case like if we encounter last cell the exit or if last column then move to next row. Also while checking validity we have to make sure the subgrid validity holds good.

* + **Include the source code and other necessary files, setup instructions in your response.**

Answer: Below is the source code , to compile it either copy and paste the code in any online C compiler and run it. OR Create a file with dot(.) c extension and copy and paste this source code and then save the file. Compile it with GCC and then run it.

#include <stdio.h>

#include <stdlib.h>

// N is the size of the 2D matrix N\*N

#define N 9

/\* isValid utility will check whether assignment of num at row & col will be valid or invalid. \*/

int isValid(int grid[N][N], int row,

int col, int num)

{

/\* Check if row has same num then it is a invalid entry \*/

for (int x = 0; x <= 8; x++)

if (grid[row][x] == num)

return 0;

/\* Check if col has same num then it is a invalid entry \*/

for (int x = 0; x <= 8; x++)

if (grid[x][col] == num)

return 0;

/\* Check if subgrid has same num then it is a invalid entry \*/

int subRow = row - row % 3;

int subCol = col - col % 3;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (grid[i + subRow][j + subCol] == num)

return 0;

/\* All checks are successful , hence returning 1 as it is a valid entry \*/

return 1;

}

/\* sudokuSolver is the utility which will fill the grid if the

grid is valid else it will return 0 for invalid grid \*/

int sudokuSolver(int grid[N][N], int row, int col)

{

/\* Check if we have reached the last row and last column then

return 1 as no more entry to shell to check and fill the value \*/

if (row == N - 1 && col == N)

return 1;

/\* Check if we are in last column then go to next row \*/

if (col == N)

{

row++;

col = 0;

}

/\* Check if the current grid position value >0, then we iterate for next column \*/

if (grid[row][col] > 0)

return sudokuSolver(grid, row, col + 1);

for (int num = 1; num <= N; num++)

{

/\* Check if it is valid to place num at row & col position \*/

if (isValid(grid, row, col, num)==1)

{

grid[row][col] = num;

/\* Checking for next possibility for next column \*/

if (sudokuSolver(grid, row, col + 1)==1)

return 1;

}

/\* As our assumption was wrong hence making the grid value to 0. \*/

grid[row][col] = 0;

}

return 0;

}

int main()

{

// 0 means unassigned cells

int grid[N][N] = { { 0, 0, 2, 0, 0, 0, 0, 4, 1 },

{ 0, 0, 0, 0, 8, 2, 0, 7, 0 },

{ 0, 0, 0, 0, 4, 0, 0, 0, 9 },

{ 2, 0, 0, 0, 7, 9, 3, 0, 0 },

{ 0, 1, 0, 0, 0, 0, 0, 8, 0 },

{ 0, 0, 6, 8, 1, 0, 0, 0, 4 },

{ 1, 0, 0, 0, 9, 0, 0, 0, 0 },

{ 0, 6, 0, 4, 3, 0, 0, 0, 0 },

{ 8, 5, 0, 0, 0, 0, 4, 0, 0 } };

if (sudokuSolver(grid, 0, 0)==1)

{

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

printf("%d ",grid[i][j]);

printf("\n");

}

}

else

printf("No solution exists");

return 0;

}

* **Check your solver with at least the 2 examples provided above**

Output for Easy

==============

6 1 3 8 7 9 4 2 5

9 2 4 6 3 5 1 7 8

5 8 7 1 2 4 9 3 6

4 9 8 3 5 6 2 1 7

7 3 1 9 8 2 5 6 4

2 5 6 7 4 1 8 9 3

8 6 9 5 1 3 7 4 2

3 4 5 2 9 7 6 8 1

1 7 2 4 6 8 3 5 9

=============

Output for Hard

==============

6 3 2 9 5 7 8 4 1

4 9 1 6 8 2 5 7 3

7 8 5 3 4 1 2 6 9

2 4 8 5 7 9 3 1 6

3 1 9 2 6 4 7 8 5

5 7 6 8 1 3 9 2 4

1 2 4 7 9 5 6 3 8

9 6 7 4 3 8 1 5 2

8 5 3 1 2 6 4 9 7

=========================End Of First Problem===================================

# Project 2

The following code in C language contains some vulnerabilities. Analyze the code, report in the code as comments the vulnerabilities you have found and explain why these are vulnerabilities.

#include <stdio.h>

#include <stdlib.h>

#include <wchar.h>

#define PASSWORD "ABCD1234!" /\* Storing password like this is not safe and also not good practice \*/

/\*You need not worry about other include statements if at all any are missing \*/

void func1()

{

char \* data; /\* Good practice to initialize NULL to a pointer \*/

char \* dataBuffer = (char \*)ALLOCA(100\*sizeof(char)); /\*There is no ALLOCA function in C , either it malloc\,callocor realloc\*/

memset(dataBuffer, 'A', 100-1);

dataBuffer[100-1] = '\0';

data = dataBuffer - 8;/\*Not correct as we are pushing back the pointer , it will cause segmentation fault \*/

{

char source[100];

memset(source, 'C', 100-1);

source[100-1] = '\0';

strcpy(data, source); /\* Cause segmetation fault as data location is invalid \*/

if(data != NULL)

{

printf("%s\n", data); /\* might print corrupted data \*/

}

}

}

void func2()

{

char \* data; /\* Good practice to initialize NULL to a pointer \*/

data = NULL;

data = (char \*)calloc(100, sizeof(char));

strcpy(data, "A String"); /\* may result in buffer overflow , better to use strncpy \*/

if(data != NULL)

{

printf("%s\n", data);

}

}

void func3()

{

char \* password; /\* Good practice to initialize NULL to a pointer \*/

char passwordBuffer[100] = "";

password = passwordBuffer;

strcpy(password, PASSWORD); /\* may result in buffer overflow , better to use strncpy \*/

{

HANDLE pHandle;

char \* username = "User";

char \* domain = "Domain";

/\* Let's say LogonUserA is a custon authentication function\*/

if (LogonUserA(

username,

domain,

password,

&pHandle) != 0)

{

printf("User logged in successfully.\n");

CloseHandle(pHandle);

}

else

{

printf("Unable to login.\n");

}

}

}

static void func4()

{

char \* data; /\* Good practice to initialize NULL to a pointer \*/

data = NULL;

data = (char \*)calloc(20, sizeof(char));

if (data != NULL)

{

strcpy(data, "Initialize"); /\* may result in buffer overflow , better to use strncpy \*/

if(data != NULL)

{

printf("%s\n", data);

}

free(data); /\* Before free first it should be checked whether it contain memory address \*/

}

}

void func5()

{

int i = 0;

do

{

printf("%d\n", i);

i = (i + 1) % 256;

} while(i >= 0); /\* Infinite while loop \*/

}

void func6()

{

char dataBuffer[100] = "";

char \* data = dataBuffer;

printf("Please enter a string: ");

if (fgets(data, 100, stdin) < 0)

{

printf("fgets failed!\n");

exit(1);

}

if(data != NULL)

{

printf("%s\n", data);

}

}

void func7()

{

char \* data; /\* Good practice to initialize NULL to a pointer \*/

data = "Fortify";

data = NULL;

printf("%s\n", data); /\* segmetation fault \*/

}

int main(int argc, char \* argv[])

{

printf("Calling func1\n");

func1();

printf("Calling func2\n");

func2();

printf("Calling func3\n");

func3();

printf("Calling func4\n");

func4();

printf("Calling func5\n");

func5();

printf("Calling func6\n");

func6();

printf("Calling func7\n");

func7();

return 0;

}