Report.pdf

Takunda Takaindisa — Assignment 1

March 2024

1 Appendix

```
/*
Sudoku rules:
1) Each 3*3 box must have the #'s 1-9
2) Each Row must have the #'s 1-9
3) Each column must have the #'s 1-9
4) A number can only appear once in each row,column, 3*3
    box
5) Order doesn't matter
*/

#include <stdio.h>
#include <stdlib.h>
#define N 9
int checkRow(int grid[N][N], int row, int num);
int checkColumn(int grid[N][N], int column, int num);
int checkBox(int grid[N][N], int boxRow, int boxColumn,
    int num);
int solveSudoku(int grid[N][N], int row, int column);
int main();
int count = 0;
//check if num is already in row, if yes, then num can't
    be placed there
int checkRow(int grid[N][N], int row, int num) {
    for (int column = 0; column < N; column++)
        if (grid[row][column] == num)
            return 0;
    return 1;
}
//check if num is  already in column, if yes, then num
    can't be placed there
```

```
int checkColumn(int grid[N][N], int column, int num) {
    for (int row = 0; row < N; row++)
        if (grid[row][column] == num)
            return 0;
    return 1;
}
//check if num is present in the 3*3 box, if yes, then
   num can't be placed there
int checkBox(int grid[N][N], int boxRow, int boxColumn,
   int num) {
    for (int row = 0; row < 3; row++)
        for (int column = 0; column < 3; column++)
            if (grid[row + boxRow][column + boxColumn] ==
                num)
                return 0;
    return 1;
}
/*
Calculation of boxRow/boxColumn:
row%3/column%3, the remainder gives the relative position
    of the current row/column in 3*3 box
subtracting the remainder from row/column gives the
   starting row/column index of the row/column in the 3*3
    box
from there, itertate 3 times, mening you've gone through
   the 3 rows/columns of the 3*3 box
*/
}

int solveSudoku(int grid[N][N], int row, int column) {
    count++;
    //count appended after every function call
    if (row == N - 1 && column == N)
    /*
    row = N-1 means the last row
    column = N (after the last row) means all the rows in
        the current column are processed/move to next row
    column needs to be compared to N because if you
       stopped at N-1, the progran wouldn't iterate for
       the bottom right cell
    */
        return 1;

    if (column == N) { //when you're at the last column,
       move on to the next row, start from th first
       column
```

```
        row++;
        column = 0;
    }

    if (grid[row][column] > 0) //if the element in the
        current place isn't 0/ is a number, move on and
        start from next column
        return solveSudoku(grid, row, column + 1);

    for (int num = 1; num <= N; num++) {
        if (checkRow(grid, row, num) && checkColumn(grid,
            column, num) && checkBox(grid, row - row % 3,
            column - column % 3, num)) {
            //if the number isn't in the current row,
                column or 3*3 grif
            grid[row][column] = num; //the number can
                then be put in the placeholder
            if (solveSudoku(grid, row, column + 1)) //if
                all previous checks/iterations return True
                , run the function again for the next
                column
                return 1;
            grid[row][column] = 0; //else that number won
                't work, set it to 0 again
        }
    }
    return 0;
}

int main() {
    int grid[N][N] = {
    {0, 2, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 6, 0, 0, 0, 0, 3},
    {0, 7, 4, 0, 8, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 3, 0, 0, 2},
    {0, 8, 0, 0, 4, 0, 0, 1, 0},
    {6, 0, 0, 5, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 0, 7, 8, 0},
    {5, 0, 0, 0, 0, 9, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 4, 0}
};


    if (solveSudoku(grid, 0, 0)) {
        printf("Solution found after %d iterations:\n",
            count);
```

```
        for (int i = 0; i < N; i++) {
            if (i % 3 == 0 && i != 0) {
                printf("---------------------\n");
            }
            for (int j = 0; j < N; j++) {
                if (j % 3 == 0 && j != 0) {
                    printf("| ");
                }
                printf("%d ", grid[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("No solution exists\n");
    }

    return 0;
}
```

2 Functions

2.1 checkRow

```
//check if num is already in row, if yes, then num can't
    be placed there
int checkRow(int grid[N][N], int row, int num) {
    for (int column = 0; column < N; column++)
        if (grid[row][column] == num)
            return 0;
    return 1;
}
```

2.2 checkColumn

```
//check if num is  already in cilumn, if yes, then num
can't be placed there
int checkColumn(int grid[N][N], int column, int num) {
    for (int row = 0; row < N; row++)
        if (grid[row][column] == num)
            return 0;
    return 1;
}
```

2.3 checkBox

```
//check if num is present in the 3*3 box, if yes, then nu
    can't be placed there
int checkBox(int grid[N][N], int boxRow, int boxColumn,
    int num) {
    for (int row = 0; row < 3; row++)
        for (int column = 0; column < 3; column++)
            if (grid[row + boxRow][column + boxColumn] ==
                num)
                return 0;
    return 1;
```

2.4 solveSudoku

```
int solveSudoku(int grid[N][N], int row, int column) {
    count++;
    //count appended after every function call
    if (row == N - 1 && column == N)
    /*
    row = N-1 means the last row
    column = N (after the last row) means all the rows in
         the current column are processed/move to next row
    column needs to be compared to N because if you
        stopped at N-1, the progran wouldn't iterate for
        the bottom right cell
    */
        return 1;

    if (column == N) { //when you're at the last column,
        move on to the next row, start from th first
        column
        row++;
        column = 0;
    }

    if (grid[row][column] > 0) //if the element in the
        current place isn't 0/ is a number, move on and
        start from next column
        return solveSudoku(grid, row, column + 1);

    for (int num = 1; num <= N; num++) {
        if (checkRow(grid, row, num) && checkColumn(grid,
            column, num) && checkBox(grid, row - row % 3,
            column - column % 3, num)) {
            //if the number isn't in the current row,
                column or 3*3 grif
            grid[row][column] = num; //the number can
                then be put in the placeholder
            if (solveSudoku(grid, row, column + 1)) //if
                all previous checks/iterations return True
                , run the function again for the next
                column
                return 1;
            grid[row][column] = 0; //else that number won
                't work, set it to 0 again
        }
    }
```

```
    return 0;
}
```

2.5 main

```
int main() {
    int grid[N][N] = {
    {0, 2, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 6, 0, 0, 0, 0, 3},
    {0, 7, 4, 0, 8, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 3, 0, 0, 2},
    {0, 8, 0, 0, 4, 0, 0, 1, 0},
    {6, 0, 0, 5, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 0, 7, 8, 0},
    {5, 0, 0, 0, 0, 9, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 4, 0}
};


    if (solveSudoku(grid, 0, 0)) {
        printf("Solution found after %d iterations:\n",
            count);
        for (int i = 0; i < N; i++) {
            if (i % 3 == 0 && i != 0) {
                printf("------------------------\n");
            }
            for (int j = 0; j < N; j++) {
                if (j % 3 == 0 && j != 0) {
                    printf("| ");
                }
                printf("%d ", grid[i][j]);
            }
            printf("\n");
        }
    } else {
        printf("No solution exists\n");
    }

    return 0;
}
```

3 Summary

In order to find a solution for a given grid, the program has functions called checkRow, checkBox,checkColumn, sudokuSolver and main. The sudokuSolver iterates the value of num from 1 to 9 using a for loop, and for each value of num, the checkRow function is used to check if num is already present in a specific row, checkColumn will be used to check if num is already present in a specific column, and checkBox will be used to check if num is already present in a given 3*3 grid. If any of these conditions are met, then num cannot be placed there as per the rules of sudoku. The solveSudoku function needs to do this in order to check if valid numbers can be placed in the empty cells of the given sudoku grid without violating the rules of Sudoku. The recursive nature of the solveSudoku function allows it to backtrack to ensure that all possibilities for the individual placeholders are looked over until a valid solution is found. The main function is where the sudoku grid is initialized via the user hard coding it in the program and then the program will solve the grid by calling the solveSudoku function inside the main function. The program will return "no solution exists" in the event solveSudoku function reaches a point where no valid number can be placed in an empty cell without violaing the rules of Sudoku.