William Kelley

Coins and Sudoku – Assignment 07

```
Start at index 0 of array with coins.
    If ary[0] mod 2 is not 0
        check next index
            if ary[index] mod 2 is not 0 && not 2
                check next index
                    if ary[index] mod 2 is not 0 && not 2
                        move on
                    else
                        move coin to this index
            else
                move coin to this index
    else
        move coin to this index
Recursively call the above function until all odds are moved, skipping the odd numbered
indexes
```

```cpp
// A Backtracking program in C to solve Sudoku problem
// Reference: https://www.geeksforgeeks.org/sudoku-backtracking-7/
#include <stdio.h>
#include <string>
#include <iostream>
#include <fstream>

using namespace std;

ifstream infile;
ofstream outfile;

#define UNASSIGNED 0

#define N 9

bool FindUnassignedLocation(int grid[N][N], int &row, int &col);

bool isSafe(int grid[N][N], int row, int col, int num);

bool SolveSudoku(int grid[N][N])
{
	int row, col;

	if (!FindUnassignedLocation(grid, row, col))
		return true;

	for (int num = 1; num <= 9; num++)
	{
		if (isSafe(grid, row, col, num))
		{
			grid[row][col] = num;

			if (SolveSudoku(grid))
				return true;

			grid[row][col] = UNASSIGNED;
		}
	}
	return false;
}

bool FindUnassignedLocation(int grid[N][N], int &row, int &col)
{
	for (row = 0; row < N; row++)
		for (col = 0; col < N; col++)
			if (grid[row][col] == UNASSIGNED)
				return true;
	return false;
}

bool UsedInRow(int grid[N][N], int row, int num)
{
	for (int col = 0; col < N; col++)
		if (grid[row][col] == num)
			return true;
	return false;
}
```

```
bool UsedInCol(int grid[N][N], int col, int num)
{
	for (int row = 0; row < N; row++)
		if (grid[row][col] == num)
			return true;
	return false;
}

bool UsedInBox(int grid[N][N], int boxStartRow, int boxStartCol, int num)
{
	for (int row = 0; row < 3; row++)
		for (int col = 0; col < 3; col++)
			if (grid[row + boxStartRow][col + boxStartCol] == num)
				return true;
	return false;
}

bool isSafe(int grid[N][N], int row, int col, int num)
{
	return !UsedInRow(grid, row, num) &&
		!UsedInCol(grid, col, num) &&
		!UsedInBox(grid, row - row % 3, col - col % 3, num) &&
		grid[row][col] == UNASSIGNED;
}

void printGrid(int grid[N][N])
{
	fstream file("output.txt", ios::in | ios::out | ios::app);

	file << "Output for Grid" << endl;
	cout << "Output for Grid" << endl;

	for (int row = 0; row < N; row++)
	{
		file << "| ";
		cout << "| ";
		for (int col = 0; col < N; col++) {
			file << grid[row][col] << " | ";
			cout << grid[row][col] << " | ";
		}
		file << endl;
		cout << endl;
	}
}

int main()
{
	int grid[N][N];
	string row;
	string delimiter = ",";

	infile.open("puzzle-1.txt");

	for (int i = 0; i < N; ++i) {
		getline(infile, row);
		size_t last = 0; size_t next = 0;
		int j = 0;
		while (((next = row.find(delimiter, last)) != string::npos) || j ==
N) {
			grid[i][j] = stoi(row.substr(last, next - last));
			last = next + 1;
			++j;
		}
		grid[i][j] = stoi(row.substr(last));
	}

	infile.close();

	fstream file("output.txt", ios::in | ios::out | ios::app);

	if (SolveSudoku(grid) == true) {
		printGrid(grid);
	}
	else {
		cout << "No solution exists" << endl;
		file << "No solution exists" << endl;
	}


	system("PAUSE");
	return 0;
}

/*
Output for Grid
| 3 | 1 | 6 | 5 | 7 | 8 | 4 | 9 | 2 |
| 5 | 2 | 9 | 1 | 3 | 4 | 7 | 6 | 8 |
| 4 | 8 | 7 | 6 | 2 | 9 | 5 | 3 | 1 |
| 2 | 6 | 3 | 4 | 1 | 5 | 9 | 8 | 7 |
| 9 | 7 | 4 | 8 | 6 | 3 | 1 | 2 | 5 |
| 8 | 5 | 1 | 7 | 9 | 2 | 6 | 4 | 3 |
| 1 | 3 | 8 | 9 | 4 | 7 | 2 | 5 | 6 |
| 6 | 9 | 2 | 3 | 5 | 1 | 8 | 7 | 4 |
| 7 | 4 | 5 | 2 | 8 | 6 | 3 | 1 | 9 |
Output for Grid
| 4 | 7 | 9 | 6 | 5 | 1 | 8 | 2 | 3 |
| 6 | 5 | 2 | 9 | 8 | 3 | 4 | 7 | 1 |
| 8 | 3 | 1 | 7 | 2 | 4 | 5 | 9 | 6 |
| 1 | 4 | 7 | 3 | 6 | 2 | 9 | 5 | 8 |
| 3 | 6 | 5 | 4 | 9 | 8 | 7 | 1 | 2 |
| 2 | 9 | 8 | 1 | 7 | 5 | 3 | 6 | 4 |
| 9 | 1 | 4 | 5 | 3 | 6 | 2 | 8 | 7 |
| 5 | 2 | 3 | 8 | 1 | 7 | 6 | 4 | 9 |
| 7 | 8 | 6 | 2 | 4 | 9 | 1 | 3 | 5 |
No solution exists
*/
```