// C++ Program to solve Sudoku problem

#include <iostream>

#include <vector>

using namespace std;

// Function to heck if it is safe to place num at mat[row][col]

bool isSafe(vector<vector<int>> &mat, int i, int j, int num,

vector<int> &row, vector<int> &col, vector<int> &box) {

if( (row[i] & (1 << num)) || (col[j] & (1 << num)) ||

(box[i / 3 \* 3 + j / 3] & (1 << num)) )

return false;

return true;

}

bool sudokuSolverRec(vector<vector<int>> &mat, int i, int j,

vector<int> &row, vector<int> &col, vector<int> &box) {

int n = mat.size();

// base case: Reached nth column of last row

if (i == n - 1 && j == n)

return true;

// If reached last column of the row go to next row

if (j == n) {

i++;

j = 0;

}

// If cell is already occupied then move forward

if (mat[i][j] != 0)

return sudokuSolverRec(mat, i, j + 1, row, col, box);

for (int num = 1; num <= n; num++) {

// If it is safe to place num at current position

if (isSafe(mat, i, j, num, row, col, box)) {

mat[i][j] = num;

// Update masks for the corresponding row, column and box

row[i] |= (1 << num);

col[j] |= (1 << num);

box[i / 3 \* 3 + j / 3] |= (1 << num);

if (sudokuSolverRec(mat, i, j + 1, row, col, box))

return true;

// Unmask the number num in the corresponding row, column and box masks

mat[i][j] = 0;

row[i] &= ~(1 << num);

col[j] &= ~(1 << num);

box[i / 3 \* 3 + j / 3] &= ~(1 << num);

}

}

return false;

}

void solveSudoku(vector<vector<int>> &mat) {

int n = mat.size();

vector<int> row(n, 0), col(n, 0), box(n, 0);

// Set the bits in bitmasks for values that are initital present

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (mat[i][j] != 0) {

row[i] |= (1 << mat[i][j]);

col[j] |= (1 << mat[i][j]);

box[ (i / 3) \* 3 + j / 3] |= (1 << mat[i][j]);

}

}

}

sudokuSolverRec(mat, 0, 0, row, col, box);

}

int main() {

vector<vector<int>> mat = {

{3, 0, 6, 5, 0, 8, 4, 0, 0},

{5, 2, 0, 0, 0, 0, 0, 0, 0},

{0, 8, 7, 0, 0, 0, 0, 3, 1},

{0, 0, 3, 0, 1, 0, 0, 8, 0},

{9, 0, 0, 8, 6, 3, 0, 0, 5},

{0, 5, 0, 0, 9, 0, 6, 0, 0},

{1, 3, 0, 0, 0, 0, 2, 5, 0},

{0, 0, 0, 0, 0, 0, 0, 7, 4},

{0, 0, 5, 2, 0, 6, 3, 0, 0}};

solveSudoku(mat);

for (int i = 0; i < mat.size(); i++) {

for (int j = 0; j < mat.size(); j++)

cout << mat[i][j] << " ";

cout << endl;

}

return 0;

}

Output

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

// C++ Program to solve Sudoku problem

#include <iostream>

#include <vector>

using namespace std;

// Function to check if it is safe to place num at mat[row][col]

bool isSafe(vector<vector<int>> &mat, int row, int col, int num) {

// Check if num exist in the row

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

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

return false;

// Check if num exist in the col

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

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

return false;

// Check if num exist in the 3x3 sub-matrix

int startRow = row - (row % 3), startCol = col - (col % 3);

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

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

if (mat[i + startRow][j + startCol] == num)

return false;

return true;

}

// Function to solve the Sudoku problem

bool solveSudokuRec(vector<vector<int>> &mat, int row, int col) {

int n = mat.size();

// base case: Reached nth column of last row

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

return true;

// If last column of the row go to next row

if (col == n) {

row++;

col = 0;

}

// If cell is already occupied then move forward

if (mat[row][col] != 0)

return solveSudokuRec(mat, row, col + 1);

for (int num = 1; num <= n; num++) {

// If it is safe to place num at current position

if (isSafe(mat, row, col, num)) {

mat[row][col] = num;

if (solveSudokuRec(mat, row, col + 1))

return true;

mat[row][col] = 0;

}

}

return false;

}

void solveSudoku(vector<vector<int>> &mat) {

solveSudokuRec(mat, 0, 0);

}

int main() {

vector<vector<int>> mat = {

{3, 0, 6, 5, 0, 8, 4, 0, 0},

{5, 2, 0, 0, 0, 0, 0, 0, 0},

{0, 8, 7, 0, 0, 0, 0, 3, 1},

{0, 0, 3, 0, 1, 0, 0, 8, 0},

{9, 0, 0, 8, 6, 3, 0, 0, 5},

{0, 5, 0, 0, 9, 0, 6, 0, 0},

{1, 3, 0, 0, 0, 0, 2, 5, 0},

{0, 0, 0, 0, 0, 0, 0, 7, 4},

{0, 0, 5, 2, 0, 6, 3, 0, 0}};

solveSudoku(mat);

for (int i = 0; i < mat.size(); i++) {

for (int j = 0; j < mat.size(); j++)

cout << mat[i][j] << " ";

cout << endl;

}

return 0;

}

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