**AGYA GUPTA**

**SUDOKU GAME USING C++**

**ABOUT GAME**

We all know about the popular Sudoku game, wherein we need to arrange numbers from 1-9 such that they appear only once in a row and column of a 9x9 grid. The program uses the concept of backtracking. In this program, we have hard-coded the initial values, but you can also get the same input from the user (though that will be cumbersome for this program). The main thing to understand is the backtracking to find rows and columns that are not assigned any values (are zero).

**SOURCE CODE**

#include <iostream>

#include <cstdio>

#include <cstring>

#include <cstdlib>

using namespace std;

#define empty 0

#define N 9

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

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

/\* assign values to all the zero (not assigned) values for Sudoku solution

\*/

bool SolveSudoku(int grid[N][N])

{

int row, col;

if (!isEmptyLocation(grid, row, col))

return true;

for (int num = 1; num <= 9; num++)

{

if (isGridSafe(grid, row, col, num))

{

grid[row][col] = num;

if (SolveSudoku(grid))

return true;

grid[row][col] = empty;

}

}

return false;

}

/\* Check for entries that don't have a value. \*/

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

{

for (row = 0; row < N; row++)

for (col = 0; col < N; col++)

if (grid[row][col] == empty)

return true;

return false;

}

/\* Returns whether the assigned entry n in the particular row matches

the given number num. \*/

bool UsedInRow(int grid[N][N], int prow, int number)

{

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

if (grid[prow][col] == number)

return true;

return false;

}

/\* Returns true if the number num matches any number in the column \*/

bool UsedInCol(int grid[N][N], int pcol, int number)

{

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

if (grid[row][pcol] == number)

return true;

else

return false;

}

//Check if the entry used already in the grid box

bool UsedInBox(int grid[N][N], int boxBeginRow, int boxBeginCol, int number)

{

bool tf = false;

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

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

if (grid[row+boxBeginRow][col+boxBeginCol] == number)

tf = true;

return tf;

}

/\* Checks if num can be assigned to a given prow,pcol location. \*/

bool isGridSafe(int grid[N][N], int prow, int pcol, int number)

{

return !UsedInRow(grid, prow, number) && !UsedInCol(grid, pcol, number) &&

!UsedInBox(grid, prow - prow % 3 , pcol - pcol % 3, number);

}

/\* print result \*/

void printResult(int finalgrid[N][N])

{

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

{

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

cout<< finalgrid[row][col]<<" ";

cout<<endl;

}

}

/\* Main \*/

int main()

{

int grid[N][N] = {{0, 0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 0, 0, 0, 3, 0, 8, 5},

{0, 0, 1, 0, 2, 0, 0, 0, 0},

{0, 0, 0, 5, 0, 7, 0, 0, 0},

{0, 0, 4, 0, 0, 0, 1, 0, 0},

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

{5, 0, 0, 0, 0, 0, 0, 7, 3},

{0, 0, 2, 0, 1, 0, 0, 0, 0},

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

if (SolveSudoku(grid) == true)

printResult(grid);

else

cout<<"No solution found"<<endl;

return 0;

}