**Sudoku Solver - C++ Code**

SUDOKU SOLVER

#include <iostream> using namespace std; #define N 9

// Function to print the grid void printGrid(int grid[N][N]) { for (int row = 0; row < N; row++) { for (int col = 0; col < N; col++) cout << grid[row][col] << " "; cout << endl; }

}

// Check if a number can be placed at grid[row][col] bool isSafe(int grid[N][N], int row, int col, int num) { // Check row and column for (int x = 0; x < N; x++) {

if (grid[row][x] == num || grid[x][col] == num) return false; }

// Check 3x3 subgrid

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

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

if (grid[i + startRow][j + startCol] == num) return false;

return true; }

// Backtracking function to solve the puzzle bool solveSudoku(int grid[N][N]) { int row, col; bool emptyFound = false;

// Find an empty cell for (row = 0; row < N; row++) { for (col = 0; col < N; col++) { if (grid[row][col] == 0) { emptyFound = true; break; } }

if (emptyFound) break; }

// If no empty cell, puzzle solved

if (!emptyFound) return true;

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

if (solveSudoku(grid)) return true;

// Backtrack grid[row][col] = 0; } }

return false; // Trigger backtracking }

int main() { // 0 means empty cell int grid[N][N] = { {5,3,0, 0,7,0, 0,0,0},

{6,0,0, 1,9,5, 0,0,0},

{0,9,8, 0,0,0, 0,6,0},

{8,0,0, 0,6,0, 0,0,3},

{4,0,0, 8,0,3, 0,0,1},

{7,0,0, 0,2,0, 0,0,6},

{0,6,0, 0,0,0, 2,8,0},

{0,0,0, 4,1,9, 0,0,5}, {0,0,0, 0,8,0, 0,7,9} };

if (solveSudoku(grid)) printGrid(grid); else

cout << "No solution exists" << endl;

return 0; }

**Output:**

Output:

1. 3 4 6 7 8 9 1 2
2. 7 2 1 9 5 3 4 8

1 9 8 3 4 2 5 6 7

8 5 9 7 6 1 4 2 3

4 2 6 8 5 3 7 9 1

7 1 3 9 2 4 8 5 6

9 6 1 5 3 7 2 8 4

1. 8 7 4 1 9 6 3 5
2. 4 5 2 8 6 1 7 9