**Submitted by: Anjali Bhandari**

**Roll number: 12**

**Section: SE**

**Q – Program for solving 9\*9 sudoku using C++.**

#include <iostream>

using *namespace* std;

// N is the size of the 2D matrix   N\*N

#define N 9

*void* print(*int* *arr*[N][N])

{

    for (*int* i = 0; i < N; i++)

    {

        for (*int* j = 0; j < N; j++)

            cout << *arr*[i][j] << " ";

        cout << endl;

    }

}

*bool* isSafe(*int* *grid*[N][N], *int* *row*,

*int* *col*, *int* *num*)

{

    for (*int* x = 0; x <= 8; x++)

        if (*grid*[*row*][x] == *num*)

            return false;

    for (*int* x = 0; x <= 8; x++)

        if (*grid*[x][*col*] == *num*)

            return false;

*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;

}

/\* Takes a partially filled-in grid and attempts

to assign values to all unassigned locations in

such a way to meet the requirements for

Sudoku solution (non-duplication across rows,

columns, and boxes) \*/

*bool* solveSuduko(*int* *grid*[N][N], *int* *row*, *int* *col*)

{

    // Check if we have reached the 8th

    // row and 9th column (0

    // indexed matrix) , we are

    // returning true to avoid

    // further backtracking

    if (*row* == N - 1 && *col* == N)

        return true;

    // Check if column value  becomes 9 ,

    // we move to next row and

    //  column start from 0

    if (*col* == N) {

*row*++;

*col* = 0;

    }

    // Check if the current position of

    // the grid already contains

    // value >0, we iterate for next column

    if (*grid*[*row*][*col*] > 0)

        return solveSuduko(*grid*, *row*, *col* + 1);

    for (*int* num = 1; num <= N; num++)

    {

        // Check if it is safe to place

        // the num (1-9)  in the

        // given row ,col  ->we

        // move to next column

        if (isSafe(*grid*, *row*, *col*, num))

        {

           /\* Assigning the num in

              the current (row,col)

              position of the grid

              and assuming our assined

              num in the position

              is correct     \*/

*grid*[*row*][*col*] = num;

            //  Checking for next possibility with next

            //  column

            if (solveSuduko(*grid*, *row*, *col* + 1))

                return true;

        }

*grid*[*row*][*col*] = 0;

    }

    return false;

}

*int* main()

{

    // 0 means unassigned cells

*int* grid[N][N] = { { 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 } };

    if (solveSuduko(grid, 0, 0))

        print(grid);

    else

        cout << "no solution  exists " << endl;

    return 0;

}