```cpp
#include <bits/stdc++.h>
#include <time.h>
using namespace std;

class Sudoko Generator {
    public:
    // 9 X 9 size of sudoko board
    // size of 1 box 3 X 3

    int sudoko[9][9];
    int m; // no. of missing values


    // to fill small box


    bool is_valid_box(int row, int col, int value)
    {
        for(int i = 0; i < 3; i++) {
            for(int j = 0; j < 3; j++) {
                if(sudoko[row+i][col+j] == value) return false;
            }
        }
        return true;
    }

    void generate_box(int row, int col)
    {
        int val;
        srand(time(0));
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                do
                {
                    val = (int)(rand()%9);
                    val++;
                }
                while (!is_valid_box(row, col, val));
                sudoko[row+i][col+j] = val;
            }
        }
    }
```

```
// check for valid row and column

bool is_valid_row(int row, int value)
{
    for (int i = 0; i < 9; i++)
        if (sudoko[row][i] == value)
            return false;
    return true;
}

bool is_valid_column(int col, int value)
{
    for (int i = 0; i < 9; i++)
        if (sudoko[i][col] == value)
            return false;
    return true;
}

// Check if entry made is valid to put in cell
bool is_safe(int row,int col,int value)
{
    return (is_valid_row(row, value) && is_valid_column(col, value) &&
            is_valid_box(row - row%3, col - col%3, value));
}

// Filling starts

// first of all diagonal boxes are filled
void fill_diagonal()
{
    for(int i = 0; i < 9; i = i + 3)
    {
        generate_box(i,i);
    }
}
// Using recursion to fill the remaining entries

bool fill_the_rest(int row, int col)
{
    // Base case to stop recursion
    if(row >= 9 && col >= 9)
        return true;

    if(col >= 9 && row < 8) // if current row is filled, move to next row
    {
```

```
            row += 1;
            col = 0;
        }

        if (row < 3) // to check if its fall in diagonal box
        {
            if (col < 3)
                col = 3;
        }
        else if (row < 6)
        {
            if (col == (int)(row/3)*3)
                col =  col + 3;
        }
        else
        {
            if (col == 6)
            {
                row += 1;
                col = 0;
                if (row >= 9)
                    return true;
            }
        }

        for (int value = 1; value <= 9; value++)
        {
            if (is_safe(row, col, value))
            {
                sudoko[row][col] = value;
                if (fill_the_rest(row, col+1)) // fill the puzzle row wise
                    return true;

                sudoko[row][col] = 0;
            }
        }
        return false;
    }

    //Sudoko is completed. Now remove m - missing values from sudoko;

    pair<int,int> get_index()
    {
        int x = (int)(rand()%81);
        x++;
```

```
        int row = (x/9);
        int col = x%9;

        return make_pair(row,col);
    }

    void form_puzzle()
    {
        while(m != 0)
        {
            pair<int,int> cell = get_index();
            int row = cell.first;
            int col = cell.second;

            if (col != 0)
                col = col - 1;

            if (sudoko[row][col] != 0)
            {
                m--;
                sudoko[row][col] = 0;
            }
        }
    }

    void gen_sudoko()
    {
        fill_diagonal();
        fill_the_rest(0,3);
        form_puzzle();
    }

    void print()
    {
        for(int i = 0; i < 9; i++) {
            if(i % 3 == 0 and i != 0) {
                cout<<"- - - - - - - - - - - - -"<<endl;
            }
            for(int j = 0; j < 9; j++)  {
                if(j % 3 == 0) {
                    cout<<" | ";
                }
                cout<<sudoko[i][j]<<" ";
            }
            cout<<endl;
```

```
        }
    }

} ;

class Sudoko_Solver: public Sudoko_Generator
{
    public:

    pair<int,int> get_empty_cell() {
        for(int i = 0; i < 9; i++) {
            for(int j = 0; j < 9; j++) {
                if(sudoko[i][j] == 0) return make_pair(i,j);
            }
        }

        return make_pair(-1,-1);
    }

    bool is_valid_entry(int value, int row, int col)
    {
        // check for valid row
        for(int i = 0; i < 9; i++)
        {
            if(sudoko[row][i] == value and i != col) return false;
        }

        // check for valid col
        for(int i = 0; i < 9; i++)
        {
            if(sudoko[i][col] == value and i != row) return false;
        }

        // check for valid box
        int cell_row = row/3;
        int cell_col = col/3;

        for(int i = cell_row*3; i < cell_row*3 + 3; i++) {
            for(int j = cell_col*3; j < cell_col*3 + 3; j++) {
                if(sudoko[i][j] == value and (row != i and col != j)) return fals
e;
            }
        }

        return true;
```

```
}

bool solve()
{
    pair<int,int> cell = get_empty_cell();
    int row = cell.first;
    int col = cell.second;

    if(row == -1 or col == -1) // base case to stop recursion
    {
        return true;
    }

    for(int value = 1; value < 10; value++)
    {
        if(is_valid_entry(value, row, col))
        {
            sudoko[row][col] = value;

            if(solve()) return true;

            // if not possible to solve, then backtrack
            sudoko[row][col] = 0;
        }

    }

    return false;
}


void print_solved_sudoko()
{
    solve();
    for(int i = 0; i < 9; i++) {
        if(i % 3 == 0 and i != 0) {
            cout<<"- - - - - - - - - - - - -"<<endl;
        }
        for(int j = 0; j < 9; j++)  {
            if(j % 3 == 0) {
                cout<<" | ";
            }
            cout<<sudoko[i][j]<<" ";
        }
```

```
            cout<<endl;
        }
    }


} sudoko;

int main()
{
   int m;
   cin>>m;
   sudoko.m = m;
   cout<<" Please Solve ME\n\n";

   sudoko.gen_sudoko();
   sudoko.print();

   cout<<endl;
   cout<<"**********\n";
   sudoko.print_solved_sudoko();

   return 0;
}
```