C++ OVERVIEW

C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features.

C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.

C++ fully supports object-oriented programming, including the four pillars of object-oriented development:

- ✔ Encapsulation
- ✔ Data hiding
- ✔ Inheritance
- ✔ Polymorphism

Standard Libraries of C++ consists of three important parts:

- ✔ The core language giving all the building blocks including variables, data types and literals, etc.
- ✔ The C++ Standard Library giving a rich set of functions manipulating files, strings, etc.
- ✔ The Standard Template Library (STL) giving a rich set of methods manipulating data structures, etc.

Uses:

- ✔ C^{++} is used by hundreds of thousands of programmers in essentially every application domain.
- ✔ C^{++} is being highly used to write device drivers and other software that rely on direct manipulation of hardware under real-time constraints.
- ✔ C^{++} is widely used for teaching and research because it is clean enough for successful teaching of basic concepts.

When we consider a C^{++} program, it can be defined as a collection of objects that communicate via invoking each other's methods. The basic terminology used in C^{++} is:

- ✔ **Object** - Objects have states and behaviour.
  Example: A dog has states - colour, name, breed as well as behaviour - wagging, barking, and eating. An object is an instance of a class.
- ✔ **Class** - A class can be defined as a template/blueprint that describes the behaviour/states that object of its type support.
- ✔ **Methods** - A method is basically a behaviour. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- ✔ **Instant Variables** - Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.

INTRODUCTION

The modern version of Sudoku was invented in 1979 by Howard Garns in USA (where it was called `Number Place'). It became really popular in Japan in the 1980s and in the UK since late 2004. It is now quickly spreading worldwide. The word **Sudoku** is an abbreviation of a phrase which means "**the digits must occur only once**".

The aim of a Sudoku puzzle is to fill in the grid so that each row, each column and each box contains all the numbers from 1 to 9. Usually the grid is 9 by 9, using the numbers from 1 to 9, but the easier grids are smaller, using numbers from 1 to 4 or 1 to 6.

There are several levels of difficulty:

- ✔ **Easy:** A 4 by 4 grid to start off, leading to a 6 by 6 grid.
- ✔ **Medium:** 9 by 9 grids, but with plenty of numbers already in place and some helpful clues.
- ✔ **Hard:** The hard puzzles have less numbers already in place and require more strategies to be used.

Sudoku is a logic game and involves absolutely no math. Sometimes Sudoku puzzles even come with pictures, letters or symbols instead of numbers

Playing Sudoku regularly can have benefits, like boosting your concentration and focus, preventing or easing depression and possibly even preventing dementia and Alzheimer's disease, according to some studies.

Sudoku is good for anyone and any age and helps develop mental abilities as well as keeps them in good condition.

ABOUT MY PROGRAM

Solving Sudoku has been a challenging problem in the last decade. The purpose of this program is to develop more effective algorithm in order to reduce the computing time and utilize lower memory space. This program uses an algorithm for solving Sudoku puzzle by using a method, called back tracking algorithm. There are currently different variants of Sudoku such as 4X4 grids, 9X9 grids and 16X16 grids. This work is focused on classic and regular Sudoku of 9X9 board, and then a comparison is performed between the paper-and-pencil method and this algorithm.

This program can be very handy while solving Sudoku puzzles. It uses the concept of data file handling for taking input. It fetches the unsolved Sudoku problem from an already stored data file and computes a possible solution to the given problem.

ALGORITHM

The algorithm of the program is as follows:

- ✔ First input Sudoku puzzle (9X9) from the user is taken and the grid is stored in the data file 'Sudoku.txt'.
- ✔ Before solving the Sudoku input grid is displayed on the terminal window.
- ✔ Firstly the grid is checked whether it is correct or not. If correct following steps are executed to solve the puzzle else it will display the message '**The Sudoku puzzle you entered is incorrect**".
  - Firstly empty locations of the entire grid are checked for all possible digits and entries at those locations are made where there is only one possibility.
  - Consequently, empty locations in the grid are filled on the basis of hints we get from the previous entries and the process moves on till no further entries can be made in this way.
  - If still the grid is unsolved then program tries to solve it by hit and trial methodology. Using hit and trial method, program makes some assumptions about a possible number at the empty location and make further entries on the basis of these assumptions, till either contradiction is encountered or Sudoku is solved. If a contradiction is encountered then all the entries based on faulty assumptions are removed and the next possibilities are tried. This process is followed till the Sudoku is solved.
- ✔ Once the Sudoku is solved, solved puzzle is displayed on the terminal window.

FUNCTIONS USED

- ✔ **nextEmptyLoc()**: To set the values of 'r' and 'c' such that they locate the next empty location then.
- ✔ **validRow():** To check whether the number is repeating in that row or not. It returns true if the number is not repeating.
- ✔ **validCol():** To check whether the number is repeating in that column or not. It returns true if the number is not repeating.
- ✔ **validBox():** To check whether the number is repeating in 3X3 grid or not. It returns true if the number is not repeating.
- ✔ **IsAllowed():** To check whether a number is safe to enter a number at a particular location. It returns true if the number entered is valid
- ✔ **checkSudoku():** To check the input is correct or not
- ✔ **IsConfirmed():** To checks a location with all possibilities from digit 1 to 9. It also enters the value at that location where only one number is possible.
- ✔ **fillConfirmed():** It is used to stack all the values for a particular depth of recursion and checks for any new entries by IsConfirmed() function.
- ✔ **removeConfirmed():** To remove all the entries from that depth onwards whose assumptions proved to be wrong.
- ✔ **hitAndTriall():** To assume a possible number at the location where no confirmed entry is done.
- ✔ **printGrid():** To print the grid.
- ✔ **solveSudoku():** To solve the Sudoku.

FILES USED

HEADER FILES:

- ✔ iostream
- ✔ fstream

INPUT FILES:

- ✔ input.txt

OUTPUT FILES:

- ✔ output.txt

SOURCE CODE

```
#include <iostream>
using namespace std;
#include <fstream>
using std::ifstream;
//#include <cstdlib>


int crow[100][100], ccol[100][100], cEntries[100];
int maxdepth = 0;

int nextEmptyLoc(int sudoku[9][9], int &row, int &col) {
	while (row < 9) {
		row = row + (col == 8);
		col = (col + 1) % 9;
		if (sudoku[row][col] == 0)
			return 1;
	}

	return 0;
}

int validRow(int sudoku[9][9], int row, int num) {
	for (int i=0 ; i<9 ; ++i)
		if (sudoku[row][i] == num)
			return 0;
	return 1;
```

```
}

int validCol(int sudoku[9][9], int col, int num) {
	for (int i=0 ; i<9 ; ++i)
		if (sudoku[i][col] == num)
			return 0;
	return 1;
}

int validBox(int sudoku[9][9], int row, int col, int num) {
	for (int i=0 ; i<3 ; ++i)
		for (int j=0 ; j<3 ; ++j)
			if (sudoku[row+i][col+j] == num)
				return 0;
	return 1;
}

int isAllowed(int sudoku[9][9], int row, int col, int num) {
	if (validRow(sudoku, row, num) && validCol(sudoku, col, num) &&
validBox(sudoku, (row/3)*3, (col/3)*3, num))
		return 1;
	return 0;
}

int isConfirmed(int sudoku[9][9], int row, int col) {
	int safeCount = 0, safeNum;
	for (int num=1 ; num<=9 ; ++num) {
		if (isAllowed(sudoku, row, col, num)) {
```

```
                        safeCount++;
                        safeNum = num;
                }
        }

        if (safeCount == 1) {
                sudoku[row][col] = safeNum;
                return 1;
        }
        return 0;
}

int checkSudoku(int sudoku[9][9]) {
        for (int i=0 ; i<9 ; i++) {
                for(int j=0 ; j<9 ; j++) {
                        if (sudoku[i][j]) {
                                int n = sudoku[i][j];
                                sudoku[i][j] = 0;

                                if(isAllowed(sudoku,i,j,n))
                                        sudoku[i][j] = n;
                                else
                                        return 0;
                        }
                }
        }

        return 1;
```

```
}

void fillConfirmed(int sudoku[9][9], int depth) {
	int checkagain = 1;
	int ent = 0;
	while (checkagain) {
		checkagain = 0;
		int row = 0, col = -1;
		while (nextEmptyLoc(sudoku, row, col)) {
			if (isConfirmed(sudoku, row, col)) {
				checkagain = 1;
				crow[depth][ent] = row;
				ccol[depth][ent] = col;
				ent++;
			}
		}
	}
	cEntries[depth] = ent;
}

void removeConfirmed(int sudoku[9][9], int depth) {
	for (int i=0 ; i<cEntries[depth] ; ++i)
		sudoku[crow[depth][i]][ccol[depth][i]] = 0;

	cEntries[depth] = 0;
}

int hitAndTrial(int sudoku[9][9], int depth) {
```

```
	int row = 0, col = -1;

	maxdepth = max(maxdepth, depth);
	if (!nextEmptyLoc(sudoku, row, col))
		return 1;

	for (int num=1 ; num<=9 ; ++num) {
		if (isAllowed(sudoku, row, col, num)) {
			sudoku[row][col] = num;
			fillConfirmed(sudoku, depth);
			if (hitAndTrial(sudoku, depth+1))
				return 1;
			removeConfirmed(sudoku, depth);
			sudoku[row][col] = 0;
		}
	}

	return 0;
}

void printSudoku(int sudoku[9][9]) {
	for (int i=0 ; i<9 ; ++i) {
		for (int j=0 ; j<9 ; ++j)
			cout << sudoku[i][j] << ' ';
		cout << endl;
	}
}
```

```
int solveSudoku(int sudoku[9][9]) {
	fillConfirmed(sudoku, 0);
	if (hitAndTrial(sudoku, 1))
	{
		//printSudoku(sudoku);
		return 1;
	}
	else
	{
		cout << "No solution exists.\n";
		return 0;
	}
}

int main() {
	int sudoku[9][9];

	ifstream file; // indata is like cin
	file.open("C:/Users/Tanya Gupta/Downloads/input.txt"); // opens the
file
	if(!file)
	{
   cerr << "Error: file could not be opened" << endl;
   exit(1);
	}
	// Input sudoku
	for (int i=0 ; i<9 ; ++i)
		for (int j=0 ; j<9 ; ++j)
```

```
                        file>>sudoku[i][j];
    file.close();

    ofstream file1;
    file1.open("C:/Users/Tanya Gupta/Downloads/output.txt");

    if (checkSudoku(sudoku))
        {
            if(solveSudoku(sudoku))
            {
                for (int i=0 ; i<9 ; ++i)
                {
                    for (int j=0 ; j<9 ; ++j)
                        file1<<sudoku[i][j]<<" ";
                    file1<<endl;
                }
            }
            file1.close();
        }

        else
            cout << "Invalid Input.\n";

        return 0;
}
```

BIBLIOGRAPHY

1. https://en.wikipedia.org/wiki/sudoku_solver
2. http://www.cplusplus.com/forum/beginer/3662
3. computer science with C++, Sunita Arrora, Volume 1, Edition 10