**SUDOKU PROJECT**

****

**Submitted By:**

Samar Maithani

B. Tech (C.S.E.)

III – Semester

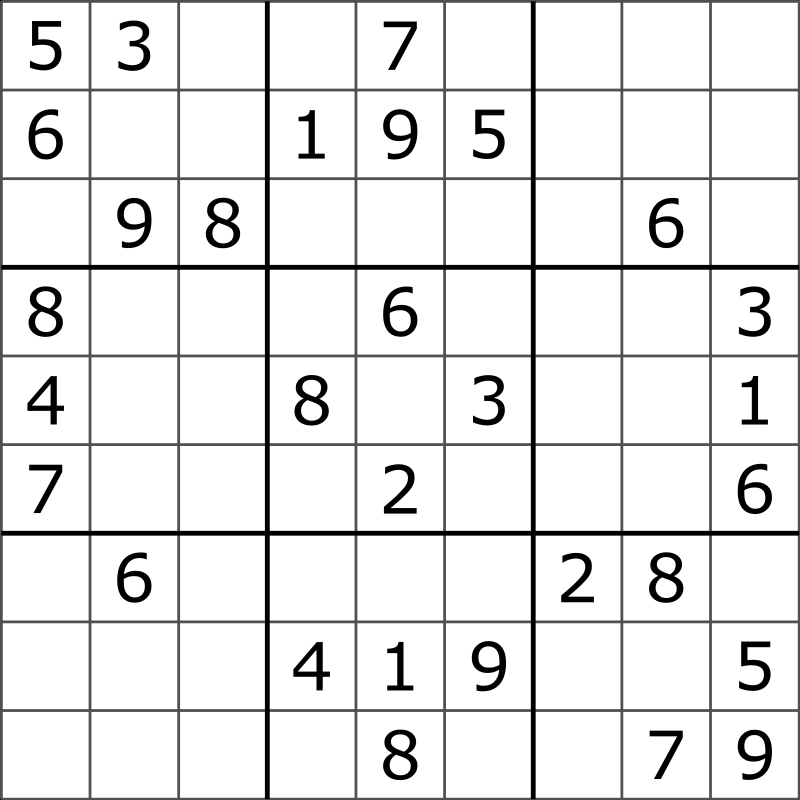
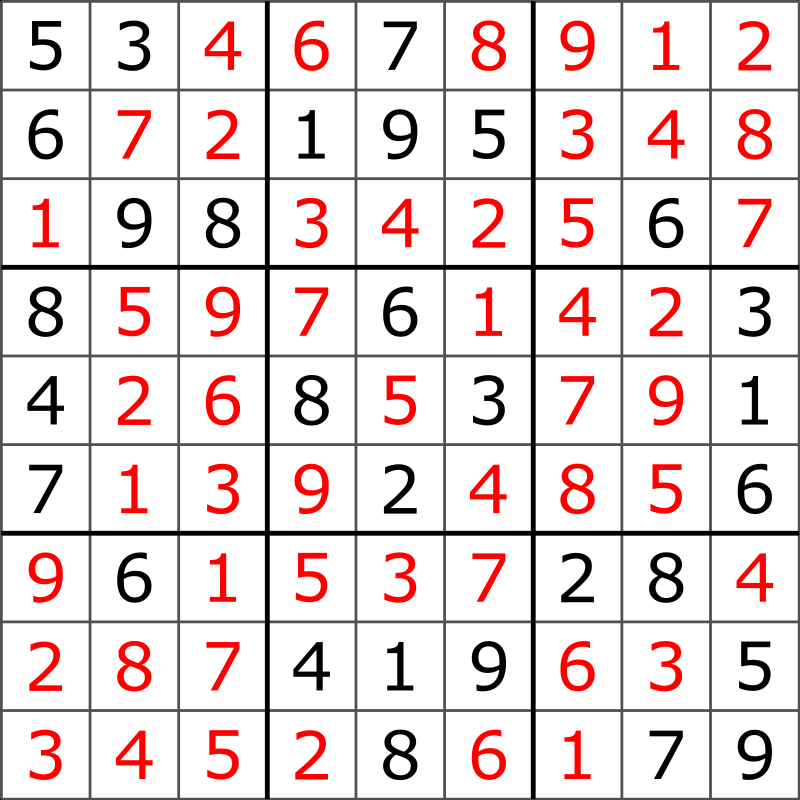
Roll No.: 1918647

Session: 2020-2021

**Introduction**

**About the Project:**

Sudoku is a logic-based, combinatorial number-placement puzzle. In classic sudoku, the objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub grids that compose the grid contain all the digits from 1 to 9. To solve the puzzle, a person must fill in all the empty squares without using the same numeral twice in each column, row, or sub grid, and without changing the numerals that are already in the grid. As shown on the image below of an actual Sudoku grid, each Sudoku grid comes with a few spaces already filled in; the more spaces filled in, the easier the game – the more difficult Sudoku puzzles have very few spaces that are already filled in.

**(**A typical Sudoku puzzle) (Its Solution/Solved Sudoku)

**C++ Programming Language:**

C++ is a general-purpose programming language and widely used now a days for competitive programming. It has imperative, object-oriented, and generic programming features. C++ runs on lots of platform like Windows, Linux, Unix, Mac etc.

**Data Types:**

**Integer**: Keyword used for integer data types is **int**. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.

**Character**: Character data type is used for storing characters. Keyword used for character data type is **char**. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.

**Boolean**: Boolean data type is used for storing Boolean or logical values. A Boolean variable can store either *true* or *false*. Keyword used for Boolean data type is **bool**.

**Floating Point**: Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is **float**. Float variables typically requires 4 byte of memory space.

**Double Floating Point**: Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is **double**. Double variables typically require 8 byte of memory space.

**ARRAYS**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. You access an array element by referring to the index number.

**FUNCTIONS**

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. Functions are used to perform certain actions, and they are important for reusing code: Define the code once and use it many times.

**System Requirements:**

1. **Minimum Hardware Requirement:**

* x86, x64, ARM, SPARC, RISC-V, or any CISC/RISC based microprocessor
* 512 MB main memory, i.e., RAM
* 1Mb secondary memory, i.e., Hard Disk
* Display monitor
* Keyboard
* Mouse

1. **Software Requirement:**

* Basic Operating System, i.e, Windows, Macintosh OS, Linux distributions, BSD, UNIX, Tails OS, DOS, Xerox OS, or any other.
* Installed C/C++ compiler, preferably GNU/G++ or Clang/LLVM or Mingw.
* Installed Shell prompt and dependencies, preferably PowerShell or cmd for Windows/.NET systems, bash/zsh for Linux, BSD, and Mac OS.
* System drivers for Display-out, and keyboard-in, as well as proper input/output channel calibration on a system-wide scale.

**Working of the Project**

**Taking Input**

First, we have to create a **9x9** 2D character array and initialize some cells with any numerical value between **‘1’ to ‘9’** (Initialize any cell with **‘.’** to represent empty cell that is to be filled). After that display it. Check whether the values entered in the cells are between ‘1’ – ‘9’ or ‘.’, If not, we give an error message and end the program, otherwise we’ll continue and check whether the Sudoku formed is valid or not (**analysis()** function is used for that purpose) by checking that each row, each column and each sub grid doesn’t repeat the same number more than once.

Graphical user interface, text, application, email

Description automatically generated

**SUDOKU Layout**

To display the Sudoku, we do it the same way as displaying an ordinary 2D Matrix, each element is separated by double white spaces. And after every 3 elements in a row and every 3 elements in a column we give 3 white spaces and 2 new lines respectively to show the 3x3 sub grid.

We use the **system(“CLS”)** function to clear the Screen every time the **display()** function is called. And at the end of the function we use the **system(“pause”)** function to wait until the user presses a key, this way the user can look at the sudoku he/she entered, otherwise the entered sudoku will just flash for an instant on the screen (we won’t get to see the entered sudoku) and then just move straight to the result.

A picture containing text

Description automatically generated

**Safety Analysis**

We use **analysis()** function (with i**nteger** return type) to check whether an element doesn’t repeat itself in that particular row, column and sub grid. The matrix along with the coordinates of the row and column is passed along to the function. We take a **counter** and set it to **0**, we run a for loop and in the loop, we check if the element given in the coordinate is equal to the elements in that row and column, if yes, we **increment** the value of the counter. After coming out of the loop we check if the counter is equal to **2**, since the element given in the coordinate should occur only once in both the row and column, therefore the counter should only increment twice (once for checking the row and other for checking the column). If the counter is not equal to **2,** we will stop the function by **returning 0**.

Now we have to check for the sub grid. For that we first **reinitialize** the counter to **0**, then we **divide** the row and column coordinates by **3** and store their values in another variables (say **t and t2**). And since they are integer values so if their value is less than 3 it will return 0, but if their value is greater than 2 and less than 6 it will return 1. So, in that case we will check if **t or t2** is equal to **1**, if true, we change the value of **t or t2 to 3**. Same we will do when the value of the coordinates is greater than 5 and less than 9, in that case the division will result **2**, and if that will be the case, we will change the value of **t or t2 to 6**. These new values t and t2 will serve as the **starting point** for checking the sub grid. And now we will check that the element given in the coordinate is equal to the element in the sub grid. We will run a for loop for the row from **t to t+3** and inside that loop we will run another for loop for the column from **t2 to t2+3**. Inside the second loop we check the occurrence of the element passed. If present, we increment the counter and coming out of both the loops we will check whether the counter is equal to **1**, since an element should occur only once inside the sub grid therefore if the value of counter is not equal to 1 then we will **return 0** otherwise we will **return 1**.

Text

Description automatically generatedText, letter

Description automatically generated

**Solution**

For solving the sudoku a different function is created with **Boolean** return type. The Sudoku we entered will be solve by assigning numbers to the **empty cell**. But before assigning numbers to the empty cell, first we must locate an empty cell, in this program the empty cell will be represented by **‘.’**. If an empty cell is found, we will **break** out of the loops so that we will have the exact coordinates of the empty cell. But if no empty cell is found we will **return true**.

Now we will run a character for loop from **‘1’ to ‘9’** and assign the number to the empty cell and after that check whether that number was safe to assign, for that we will call the **analysis()** function. The **analysis()** function will **return 0** if the number was not safe to be assigned, in that case we will unassign the number from that cell and turn it back to an empty cell by assigning **‘.’ character** to it. And if it is safe to assign the **analysis()** function will **return 1** and then we will **recursively** check whether this assignment leads to a solution or not. If the current assignment does not lead to a solution, we will try the next number for the current empty cell. And if none of the number leads to a solution, we will **return false** which will trigger the **Backtracking** and eventually the entire sudoku will be solved. And coming back to the main function we will print the solved Sudoku with the help of display() function.

Text

Description automatically generated Text

Description automatically generated

Text

Description automatically generated with medium confidence(main() function)

**How to Run the Program**

**OUTPUT**

1. The Program for the project is made by using **C++** on **CodeBlocks** software. In this software to run the program you have to press the **run button (a green play button)** located below the menu bar.

**Graphical user interface, text, application

Description automatically generated**

1. Once you press the run button a **Terminal** will open. In that terminal a message will be written asking the user (that is, you) to enter some numbers between 1 to 9 to create the numbered cells for the sudoku and enter ‘.’ to create an empty cell.

**Text

Description automatically generated**

1. Enter numbers and ‘.’ for the Sudoku **Row Wise** and press the “**Enter**” key after inputting each number or period (‘.’).

**A screenshot of a computer

Description automatically generated**

1. After entering the numbers and periods on the terminal your entered Sudoku will be shown on the screen. On the bottom of the sudoku it will ask you to press a button to continue. Press any button and it will show you the Solution of the Sudoku or rather a Solved Sudoku. But if you entered an Invalid Sudoku then instead an Error Message will be displayed on the screen.

**Shape

Description automatically generated**

**Text

Description automatically generated**