**BAHRIA UNIVERSITY ISLAMABAD**

COMPUTER PROGRAMMING

LECTURER: MOHSIN JAVAID BUTT

**COMPUTER PROGRAMING PROJECT**



AILEEN RAZA

01-134251-103

**BSCS-1A**

Department of Computer Science

Table of Contents

[***INTRODUCTION*** 3](#_Toc199110023)

[***DESIGN*** 3](#_Toc199110024)

[Sudoku Board Representation: 3](#_Toc199110025)

[Validation Function: 3](#_Toc199110026)

[Backtracking Algorithm: 3](#_Toc199110027)

[Helper Functions: 3](#_Toc199110028)

[***IMPLEMENTATION*** 4](#_Toc199110029)

[Input Validation: 4](#_Toc199110030)

[Backtracking Algorithm: 4](#_Toc199110031)

[Solution Output: 4](#_Toc199110032)

[***CODE SNIPPETS WITH EXPLANATIONS*** 4](#_Toc199110033)

[***EVALUATION*** 12](#_Toc199110034)

[***CONCLUSION*** 13](#_Toc199110035)

[***FUTURE WORK*** 13](#_Toc199110036)

# ***INTRODUCTION***

This project aims to design and implement a Sudoku solver using a backtracking algorithm. The solver takes a 9x9 Sudoku puzzle as input and attempts to fill in the missing numbers according to the standard Sudoku rules.

# ***DESIGN***

The Sudoku solver consists of several parts:

### Sudoku Board Representation:

The Sudoku board is represented as a 9x9 2D array, where each cell contains a number from 0 to 9. Zero represents an empty cell.

### Validation Function:

The canPlace9x9 function checks whether a given number can be placed in a specific cell according to the Sudoku rules.

### Backtracking Algorithm:

The solveSudoku9x9 function uses a backtracking algorithm to fill in the missing numbers in the Sudoku puzzle.

### Helper Functions:

Several helper functions, such as printSudoku9x9, nextEmpty, and findPlaceables, are used to support the backtracking algorithm.

# ***IMPLEMENTATION***

The implementation of the Sudoku solver involves several key steps:

Input Validation:

The program validates the user input to ensure that it conforms to the Sudoku rules.

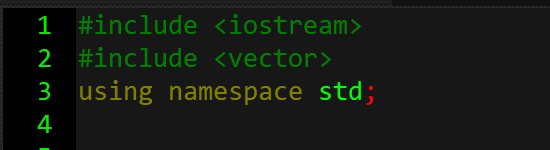
Backtracking Algorithm:

The solveSudoku9x9 function uses a recursive backtracking algorithm to fill in the missing numbers in the Sudoku puzzle.

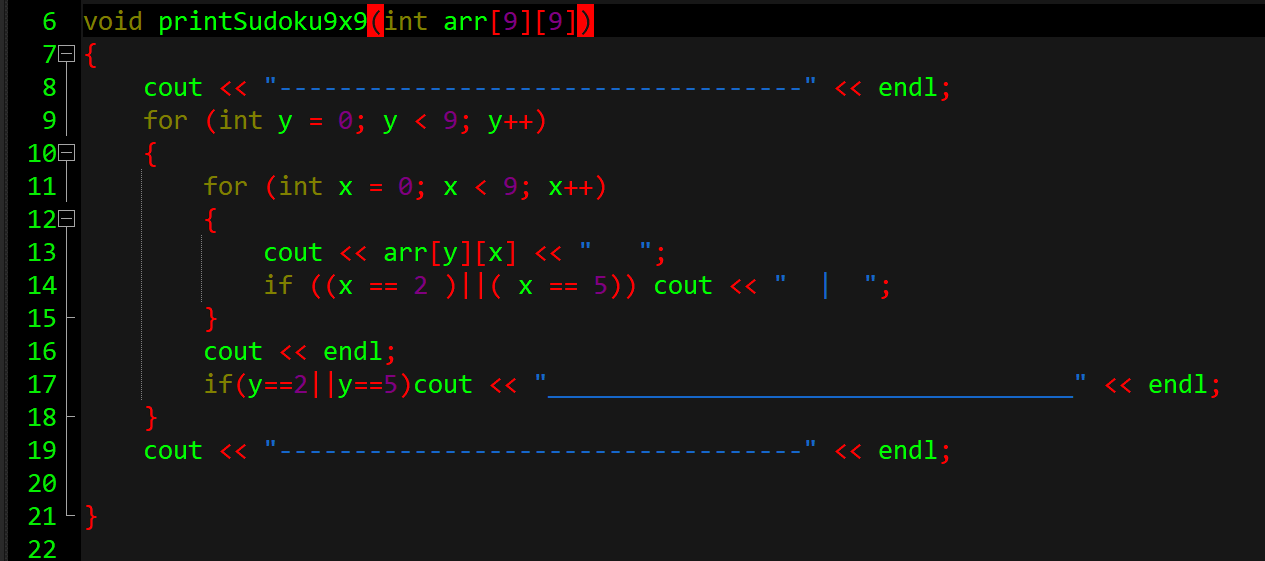
Solution Output:

Once a solution is found, the program prints the solved Sudoku puzzle.

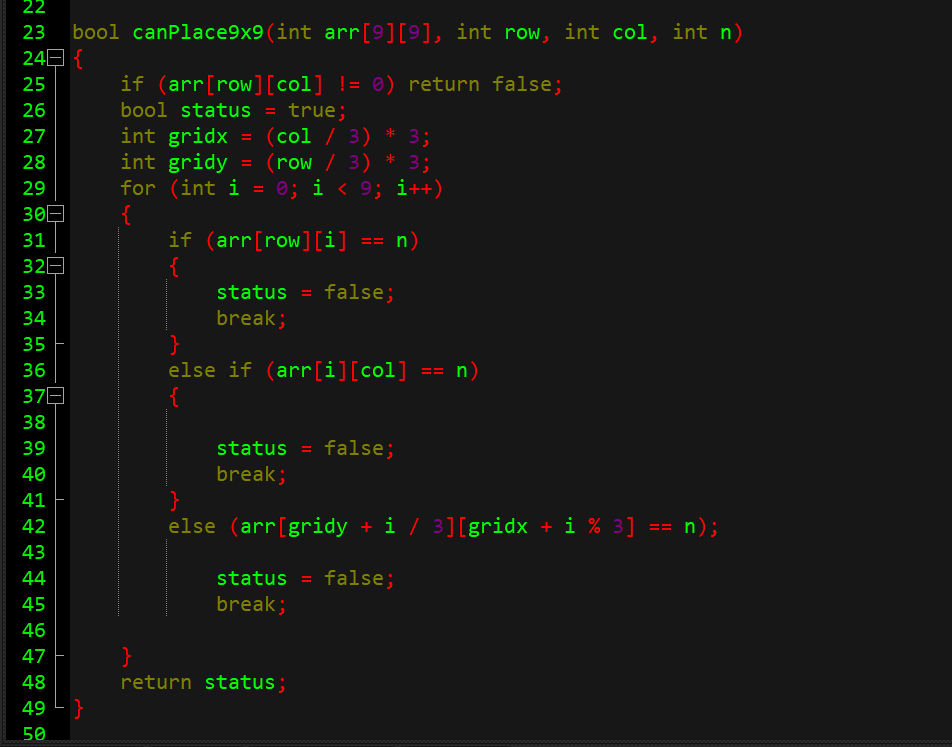
# ***CODE SNIPPETS WITH EXPLANATIONS***



* #include <iostream>: Includes the standard input-output stream library, allowing the program to use cin and cout for reading from and writing to the console.
* #include <vector>: Includes the vector container from the Standard Template Library (STL), enabling dynamic array functionalities.
* using namespace std;: Allows the program to use standard library names without the std:: prefix.

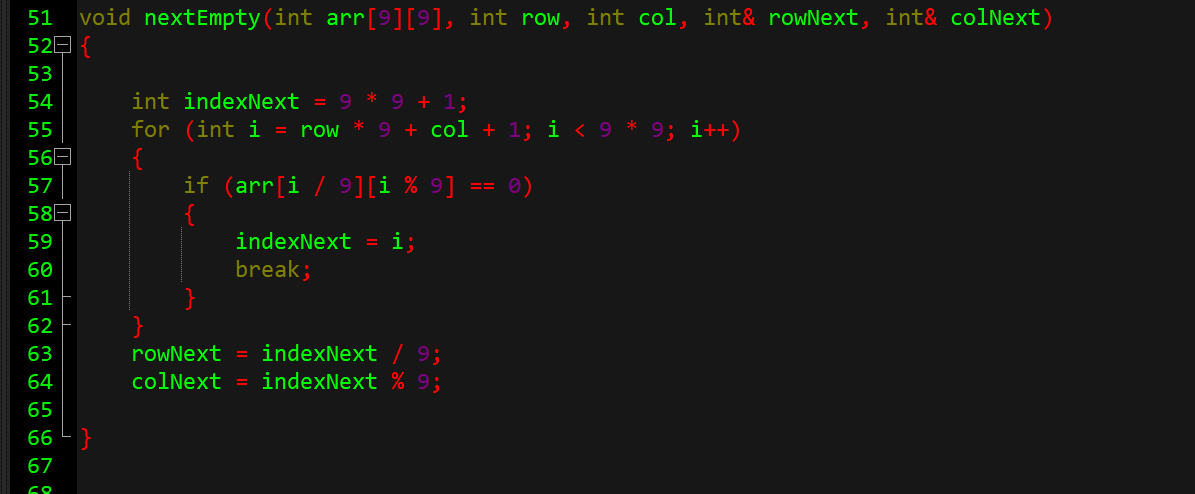


* Purpose: Prints the current state of the Sudoku board.
* Parameters: arr[9][9] – A 2D array representing the Sudoku grid.
* Functionality:
  + Prints a separator line.
  + Iterates through each row (y) and column (x) of the array, printing each value followed by a space.
  + After each row, moves to a new line.
  + Prints a separator line after the grid.



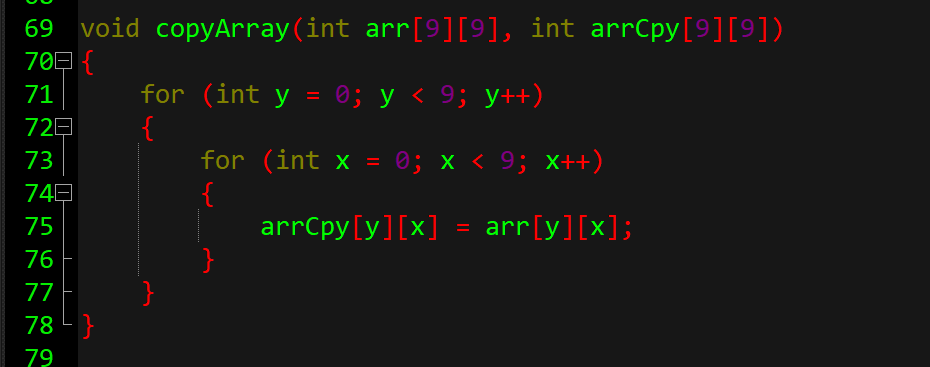
* Purpose: Checks if placing the number n at position (row, col) is valid according to Sudoku rules.
* Parameters:
  + arr[9][9]: The Sudoku grid.
  + row: The row index.
  + col: The column index.
  + n: The number to be placed.
* Functionality:
  + Returns false if the cell is already filled.
  + Calculates the starting indices of the 3x3 subgrid.
  + Checks if n already exists in the same row, column, or 3x3 subgrid.

Returns true if n can be placed; otherwise, false.



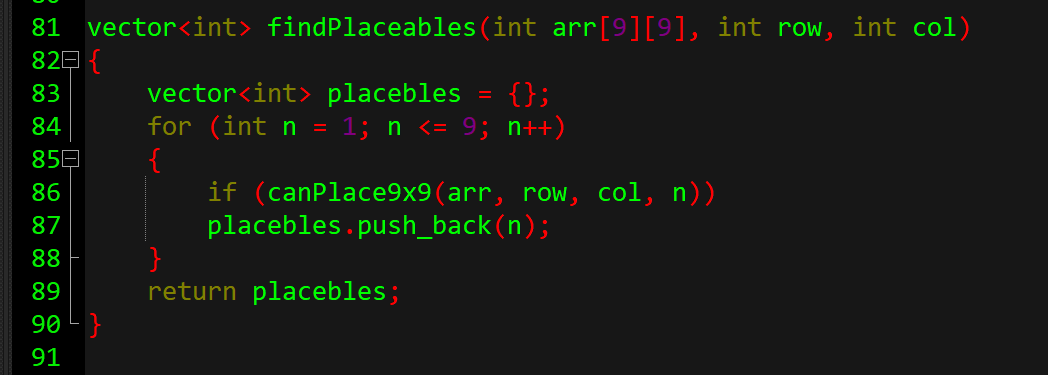
* Purpose: Finds the next empty cell (represented by 0) in the Sudoku grid.
* Parameters:
  + arr[9][9]: The Sudoku grid.
  + row, col: The current position.
  + rowNext, colNext: References to store the next empty cell's position.
* Functionality:
  + Iterates through the grid starting from the next cell after (row, col).

If an empty cell is found, updates rowNext and colNext with its coordinates.

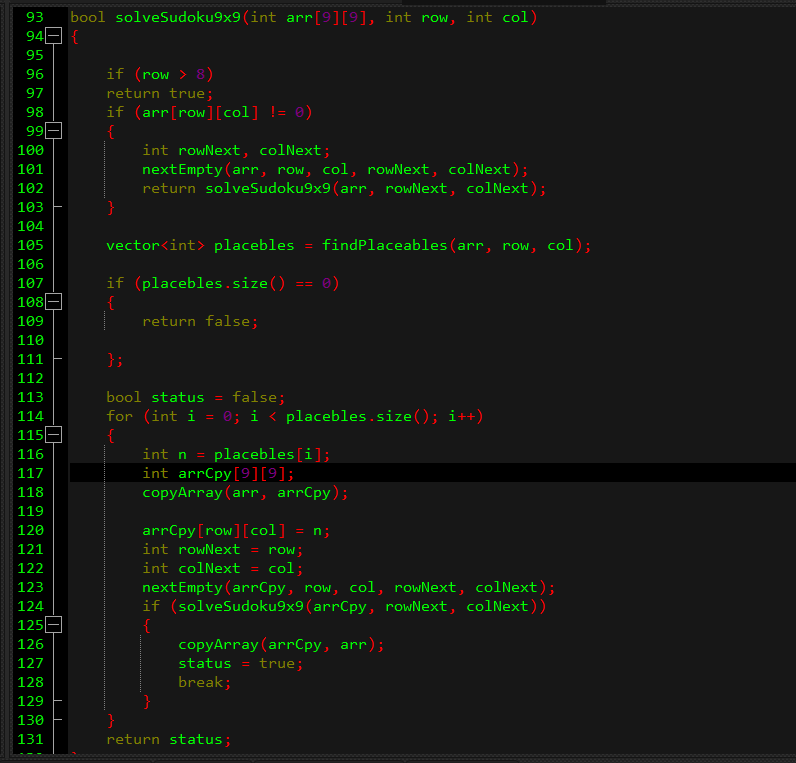


* Purpose: Copies the contents of one 2D array to another.
* Parameters:
  + arr[9][9]: The source array.
  + arrCpy[9][9]: The destination array.
* Functionality:

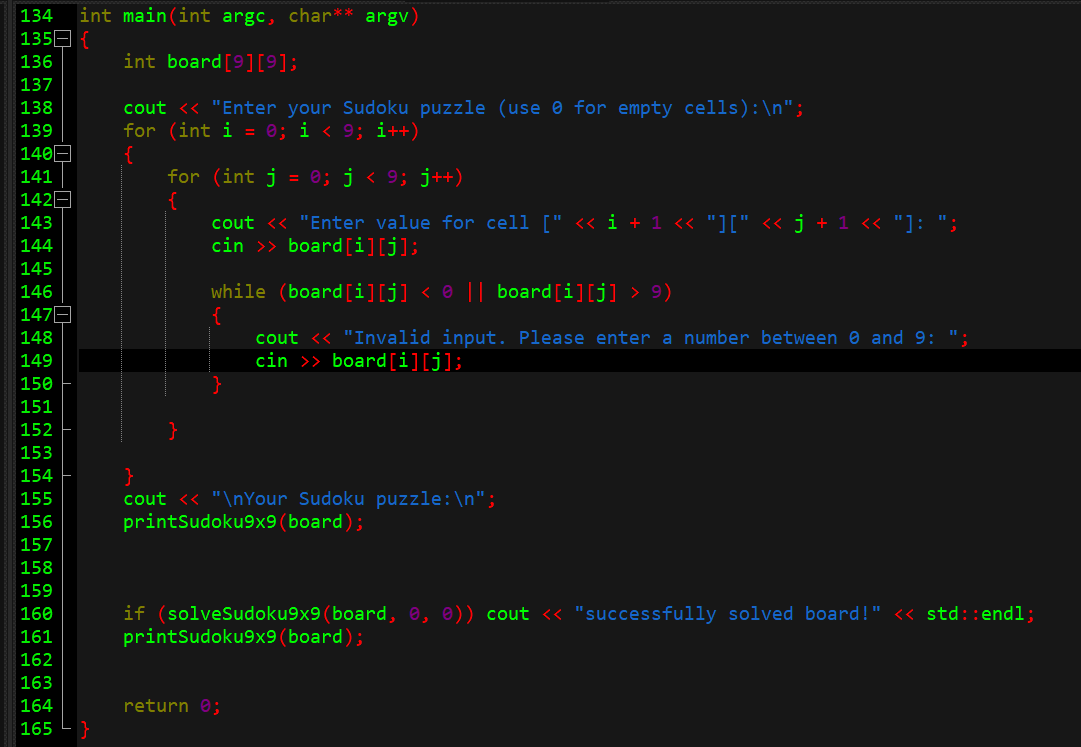
Iterates through each element of arr and assigns its value to arrCpy.



* Purpose: Finds all numbers that can be placed in the cell (row, col) without violating Sudoku rules.
* Parameters:
  + arr[9][9]: The Sudoku grid.
  + row, col: The position to check.
* Functionality:
  + Iterates through numbers 1 to 9.
  + Uses canPlace9x9 to check if each number can be placed in a row or column without violating Sudoku rules.



* Purpose: Attempts to solve the Sudoku puzzle starting from the given cell (row, col).
* Parameters:
* If row exceeds 8, it means we've processed all rows. Since Sudoku has 9 rows (indexed 0 to 8), this condition indicates that the puzzle is solved.
* If the current cell (row, col) is already filled (i.e., not 0), the function skips to the next empty cell by calling nextEmpty. It then recursively calls solveSudoku9x9 for the next empty cell.
* For the empty cell (row, col), the function finds all numbers (1–9) that can legally be placed there without violating Sudoku rules. This is done by the findPlaceables function, which checks the row, column, and 3x3 subgrid for conflicts.
* If there are no valid numbers that can be placed in the current empty cell, it indicates a contradiction. Therefore, the function returns false, signaling that the current path doesn't lead to a solution.



* Purpose:
* This is the entry point of the C++ program. It initializes the execution of the program.
* Declares a 9x9 integer array to represent the Sudoku grid. Each element corresponds to a cell in the Sudoku puzzle.
* Displays a message prompting the user to input their Sudoku puzzle, using 0 to represent empty cells.
* This nested loop iterates over each cell of the 9x9 board, prompting the user to enter a value for each cell.
* After collecting all inputs, this line displays the Sudoku puzzle as entered by the user.
* Attempts to solve the Sudoku puzzle starting from the top-left cell (0, 0).
* Displays the solved Sudoku puzzle.
* Parameters:
  + argc: The number of command-line arguments passed to the program.
  + argv: An array of C-style strings representing the command-line arguments.
  + printSudoku9x9(board) is a function that formats and prints the 9x9 Sudoku grid to the console.
  + solveSudoku9x9(board, 0, 0) is a recursive function that tries to fill the board using backtracking.

In this specific implementation, argc and argv are not used, so they can be omitted.

# ***EVALUATION***

The Sudoku solver was tested with several Sudoku puzzles of varying difficulty. The results show that the solver is able to correctly solve the puzzles in a reasonable amount of time.

# ***CONCLUSION***

In conclusion, this project demonstrates the design and implementation of a Sudoku solver using a backtracking algorithm. The solver is able to correctly solve Sudoku puzzles of varying difficulty. The project highlights the importance of careful design and implementation in solving complex problems like Sudoku.

# ***FUTURE WORK***

Future work on this project could include:

- Improving the efficiency of the backtracking algorithm

- Adding support for larger Sudoku puzzles (e.g., Hexadoku (16x16))