**K L UNIVERSITY**

**FRESHMAN ENGINEERING DEPARTMENT**

**A Project Based Lab Report**

**On**

SOLVING SUDOKU PUZZLE

**SUBMITTED BY:**

|  |  |  |
| --- | --- | --- |
| S.NO | I.D NUMBER | NAME |
| 1. | 170031326 | U.Mahesh |
| 2. | 170031333 | V.Lokesh |
| 3. | 170031388 | V.Sri sita devi |
| 4. | 170031395 | CH.Bindu kalpana |

**UNDER THE ESTEEMED GUIDANCE OF**

**<FULL NAME>**

**<DESIGNATION>**



**KL UNIVERSITY**

Green fields, Vaddeswaram – 522 502

Guntur Dt., AP, India.

**DEPARTMENT OF BASIC ENGINEERING SCIENCES**



**CERTIFICATE**

This is to certify that the project based laboratory report entitled “<TITLE>” submitted by Mr./Ms**. <name>** bearing Regd. No. <REGD.NO> to the **Department of Basic Engineering Sciences, KL University** in partial fulfillment of the requirements for the completion of a project based Laboratory in “C PROGRAMMING & DATA STRUCTURES LAB”course in I B Tech I Semester, is a bonafide record of the work carried out by him/her under my supervision during the academic year 2017 – 2 018.

PROJECT SUPERVISOR HEAD OF THE DEPARTMENT

< GUIDE NAME> Dr. D.HARITHA

**ACKNOWLEDGEMENTS**

It is great pleasure for me to express my gratitude to our honorable President **Sri. Koneru Satyanarayana**, for giving the opportunity and platform with facilities in accomplishing the project based laboratory report.

I express the sincere gratitude to our principal **Dr. A. Anand Kumar** for his administration towards our academic growth.

I express sincere gratitude to our Coordinator and HOD-BES **Dr. D.Haritha** for her leadership and constant motivation provided in successful completion of our academic semester. I record it as my privilege to deeply thank for providing us the efficient faculty and facilities to make our ideas into reality.

I express my sincere thanks to our project supervisor <name> for his/her novel association of ideas, encouragement, appreciation and intellectual zeal which motivated us to venture this project successfully.

Finally, it is pleased to acknowledge the indebtedness to all those who devoted themselves directly or indirectly to make this project report success.

Name: <NAME>

Regd . No: <IDNO>

**ABSTRACT**

1.First we have to prepare the program to create a 9\*9 board that creates the Sudoku Puzzle.

2.The board should contain 9 rows and 9 columns each of nine 3\*3 boxes , and each should contain 1 to 9 numbers & those numbers should repeat only once in each row and column if repeated it is to said wrong.

3.Mustly the board should contain n2 rows and N2 columns.

4.The numbers 1 t0 N2 (inclusive)are to be filled in each row and column

that is All numbers in a row are distinct and All numbers in each column are distinct.

5.We have to enter all the numbers in the sub-matrix having rows from (i\*N+1) to (i+1)\*N, and columns from (j\*N+1) to (j+1)\*N both are inclusive, must and should be distinct. Where 0<=I, j<=N-1.

6.Rows and Columns are to be indexed. Such that each sub-matrix is called as a “box (or) region”.

7.We have to give a partially filled Sudoku Board, that you have filled it in a perfect manner as it is possible and the Sudoku Board should be solved is to in the form of general N2\*N2 for the given problem.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO** |
| 1 | Introduction | 6 |
| 2 | Aim of the Project | 7 |
| 2.1 | Advantages & Disadvantages | 7 |
| 2.2 | Future Implementation | 8 |
| 3 | Software & Hardware Details | 9 |
| 4 | Data Flow Diagram | 10 |
| 5 | Algorithm for each modula | 11 |
| 6 | Implementation | 12 |
| 7 | Integration and System Testing | 15 |
| 8 | Conclusion | 17 |

**INTRODUCTION**

**SUDOKU:**

A standard Sudoku contains 81 cells, in a 9×9 grid, and has 9 boxes, each box being the intersection of the first, middle, or last 3 rows, and the first, middle, or last 3 columns. Each cell may contain a number from one to nine, and each number can only occur once in each row, column, and box. A Sudoku starts with some cells containing numbers (*clues*), and the goal is to solve the remaining cells. Proper Sudokus have one solution.

Puzzles must be of the standard 9x9 variety using the (ASCII) characters **1** through **9**for the puzzle symbols. Puzzles should be submitted as 81 character strings which, when read left-to-right will fill a 9x9 Sudoku grid from left-to-right and top-to-bottom. In the puzzle specification, the characters 1 - 9 represent the puzzle "givens" or clues. Any other non-blank character represents an unsolved cell.

This is a console-based Linux program, written in C language, that solves Sudoku puzzles (aka Sudoku, Number Place, etc., ) using deductive logic. It will only resort to trial-and-error and backtracking approaches upon exhausting its deductive moves.

Now onto our Sudoku Solver, it’s just like a maze but with 9 choices at each junction, rather than just two.  Assuming we have our Sudoku data recorded in a 9×9 array and there are no clashes when we call the backtrack function,

**Sudoku is one of the logical based game. We have some mathematical logics for solving sudoku game. In this project to solve sudoku in c-language we use two dimensional arrays for print the rows and columns .** Mustly the board should contain n2 rows and N2 columns.

The numbers 1 t0 N2 (inclusive)are to be filled in each row and column that is All numbers in a row are distinct and All numbers in each column are distinct. Rows and Columns are to be indexed. Such that each sub-matrix is called as a “box (or) region”.

**AIM**

**Advantages:-**

**1.By solving sudoku puzzles using c-programme is very useful and we can solve it very easily.**

**2.Solving sudoku puzzle in c in many ways and we can use many logics.**

**3.We can get output within in seconds if there is no error in our logic and suitable c - commands in solving the required sudoku puzzles.**

**4.The objective of the proposed project is to increase the thinking capability.**

**5.We can make our own sudoku and at any step we can go back to one step as well as we can see the solution of it.**

**Disadvantages:-**

1. **The disadvantage of this method is that the solving time may be comparatively slow compared to algorithms modeled after deductive methods.**
2. **One programmer reported that such an algorithm may typically require as few as 15,000 cycles, or as many as 900,000 cycles to solve a Sudoku, each cycle being the change in position of a "pointer" as it moves through the cells of a Sudoku.**
3. **A Sudoku can be constructed which is designed to work against backtracking.**

**Future enhancements:-**

**These are some of the main suggestions given by the user for improvement.**

**1.Instant help by giving hints and users progress while playing.**

**2.Score system based on time and accuracy, and database to keep track of top ten record.**

**3**. One enhancement would be to store this result and then check for remaining numbers (in backtrack) and see if any other solution exists.

4.If it does, also store that and keep on doing this till no other solution can be found.

5.After the logical solve, try to find the doublets (cells where 2 values are valid) and while backtracking, assume one value on this cell and then solve the puzzle.

6.This would reduce the sample space from 9 values to 2 values. Once this step is done we can find the triplets and so on. This can be considered to be a random constrained backtrack implementation.

7.This improvement would not speed up the solving time for most puzzles

but the puzzles with multiple solutions and exotic grids (say 16x16)

would benefit a lot from it.

**SYSTEM REQUIREMENTS**

* **SOFTWARE REQUIREMENTS:**

The major software requirements of the project are as follows:

Language : Turbo-C

Operating system**:** Windows Xp or later.

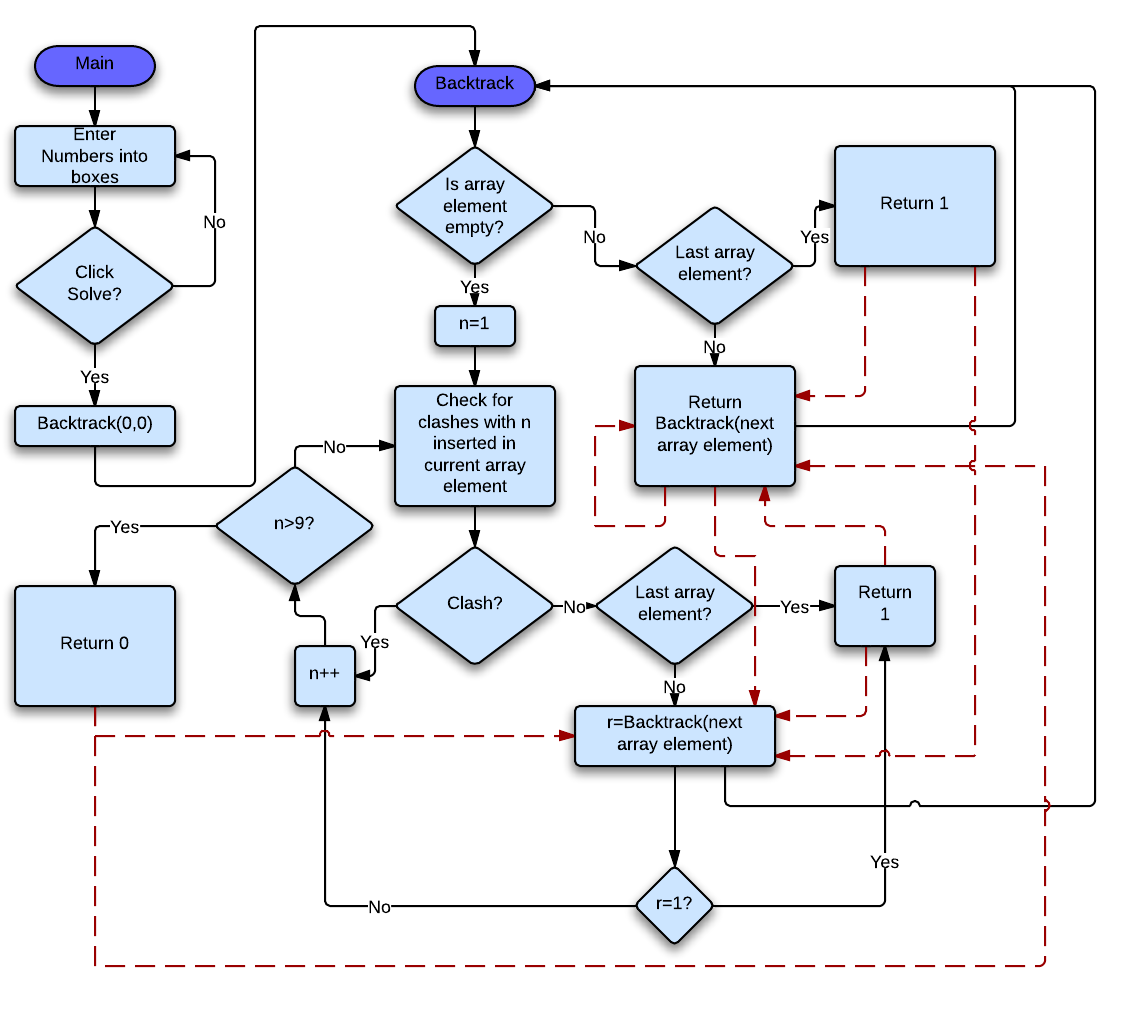
* **HARDWARE REQUIREMENTS:**

The hardware requirements that map towards the software are as follows:

RAM : 2GB

Processor : INTEL CORE

**DATA FLOW DIAGRAM**

[](http://www.icode4u.net/how-to-create-a-sudoku-solver/)Now onto our Sudoku Solver, it’s just like a maze but with 9 choices at each junction, rather than just two.  Assuming we have our Sudoku data recorded in a 9×9 array and there are no clashes when we call the backtrack function, then a flow chart would look like this:

**ALGORITHM**

The algorithm follows this sequence of steps:

STEP1: Create the row and column zones, then create the sub square zones.

STEP2:Construct the adjacency matrix of the graph.

STEP3:The search routine:

STEP3(1): Print the solution if there are no more vertices to be assigned a color, and return.

* + Sort the remaining vertices in descending order of colors present among their neighbors.
  + Pick the/a vertex with the largest number of assigned colors.
  + Try each of the remaining possible colors recursively.

STEP4:Update the hash table of vertex neighboring colors to reflect the assignment.

* + - Update the partial solution to reflect the assignment.
    - Recurse.
    - Remove the color from the partial solution.
    - Undo the color assignments from the neighboring colors hash table.

STEP5:Before the search begins, read the initial color assignment.

STEP6:Compute the set of vertices to be assigned a color, i.e. not present in the initial assignment.

STEP7:Compute the initial state of the hash table of neighbor colors.

STEP8:Start the search.

STEP9:The above algorithm can enter into loops.

STEP10: To detect this, add a hash table that stores seen configurations.

STEP11:When this happens, terminate the computation and indicate FAIL (e.g. by throwing an exception). Repeat with a different seed of the random number generator if desired.

IMPLEMENTATION

**C PROGRAM TO SOLVE SUDOKU PUZZLE**

Here is a C program to solve sudoku. We have manually entered the puzzle in the array, just manually change the values in the array before running it.

**Note:**0 represents the empty cell of sudoku puzzle

Here's the Code:

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define RESET 0

int sudokuSolver();

int findEmptyCell();

int isValid();

void printGrid();

void inputGrid();

int grid[9][9]= { {0, 0, 5, 3, 0, 0, 0, 0, 0},

{8, 0, 0, 0, 0, 0, 0, 2, 0},

{0, 7, 0, 0, 1, 0, 5, 0, 0},

{4, 0, 0, 0, 0, 5, 3, 0, 0},

{0, 1, 0, 0, 7, 0, 0, 0, 6},

{0, 0, 3, 2, 0, 0, 0, 8, 0},

{0, 6, 0, 5, 0, 0, 0, 0, 9},

{0, 0, 4, 0, 0, 0, 0, 3, 0},

{0, 0, 0, 0, 0, 9, 7, 0, 0}

};

int row,col;

//this variable was used just to keep track of number of recursive calls

long int totalNumOfCalls=0;

void main(){

int i,j,solution=0;

char ch;

clrscr();

printf("You can change the puzzle before running the program \nby changing the values in the \"grid\" array\n\n");

printf("The Entered Sudoku puzzle is: \n");

printGrid();

printf("Press 'c' to confirm and solve, or 'e' to exit: ");

ch=getch();

if(ch=='e')

exit(0);

else if(ch=='c'){

clrscr();

solution=sudokuSolver();

if(solution){

printf("\nThe Solved Sudoku is: \n\n");

printGrid();

}

else

printf("\n No Possible Solution!!\n\n");

getch();

}

}

int findEmptyCell(){

int i,j;

for(i=row;i<=8;i++)

for(j=0;j<=8;j++){

if(grid[i][j]==0)

{

row=i;col=j;

return 1;

}

}

return 0;

}

int isValid(int cellRow, int cellCol, int num){

int i,j,trow,tcol,trow1,tcol1;

int rowStart = (cellRow/3) \* 3;

int colStart = (cellCol/3) \* 3;

// to check the presence of number in row and column

for(j=0;j<=8;j++){

if(grid[cellRow][j]==num)

return 0;

if(grid[j][cellCol]==num)

return 0;

}

// to check the presence of number in 3X3 box

for(i=rowStart;i<=rowStart+2;i++)

for(j=colStart;j<=colStart+2;j++)

if(grid[i][j]==num)return 0;

return 1;

}

int sudokuSolver(){

int digit;

int prevRow,prevCol; // for backtracking

totalNumOfCalls++;

if(!findEmptyCell())

return 1;

for(digit=1;digit<=9;digit++){

if(isValid(row,col,digit)){

grid[row][col]=digit;

prevRow=row;prevCol=col;

if(sudokuSolver())

return 1;

//while backtracking assigning previous values to row and col

row=prevRow;col=prevCol;

grid[row][col]=RESET;

}

}

return 0;

}

void printGrid(){

int i,j;

printf("\t-------------------------\n");

for(i=0;i<9;i++){

printf("\t");

for(j=0;j<9;j++){

if(j==0)

printf("| ");

if(grid[i][j]==0)

printf(". ");

else

printf("%d ",grid[i][j]);

if((j+1)%3==0 )

printf("| ");

}

if((i+1)%3==0 )

printf("\n\t-------------------------");

printf("\n");

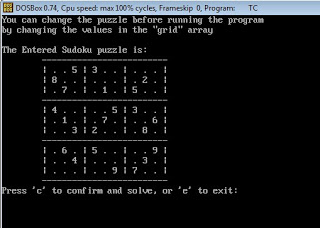
}

}

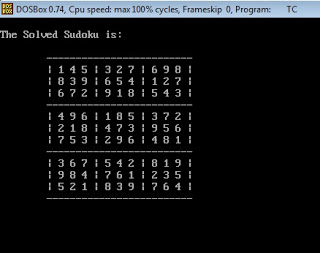
**INTEGRATION AND SYSTEM TESTING**

OUTPUTS

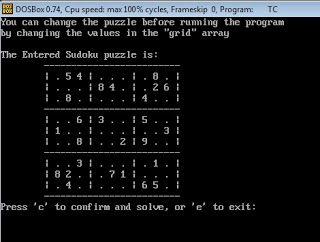
Input #1:

[](http://4.bp.blogspot.com/-Xhf3gSuI1zg/Vi6aqtDHQiI/AAAAAAAAAww/DE6y15zveTQ/s1600/sudoku-input3.jpg)

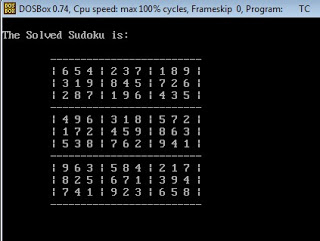
Output #1:

[](http://4.bp.blogspot.com/-IVVwZuFrN2k/Vi6asQkczII/AAAAAAAAAxA/lsBrH1sidXw/s1600/sudoku-output3.jpg)

Input #2:

[](http://1.bp.blogspot.com/-DuY3bCQswfA/Vi6axkmXx3I/AAAAAAAAAxQ/T-SZdLvo-oI/s1600/sudoku-input.jpg)

Output #2:

[](http://3.bp.blogspot.com/-V5raFwqDE28/Vi6at9l1h0I/AAAAAAAAAxI/zSSlEQ4uW4k/s1600/sudoku-output.jpg)

50
CONCLUSION
Achievements
Through generating this Sudoku solver and generator I feel I have impro
ved my programming abil...