NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

COMPLEX COMPUTING PROBLEM (CCP)

PROJECT TITLE: SUDOKU VALIDATOR

COURSE NAME:

CT-175 (Programming Fundamentals)

GROUP MEMBERS:

KHADIJA FASIH (CT-25160)

SYEDA AMNA HASNAIN (CT-25164)

UMME HANI ADEEL (CT-25151)

GROUP NAME:

WeedCode

SUBMITTED TO:

Mr. MUHAMMAD ABDULLAH

TABLE OF CONTENTS

Abstract:

This project focuses on creating a program that checks whether a completed Sudoku board is valid. Sudoku is a popular 9×9 number puzzle where each row, column, and 3×3 sub-grid must contain all the digits from 1 to 9 without repeating any number. With this program, users can enter a completed board, and it will automatically verify if all the rules are followed, giving instant feedback on whether the board is valid or not. This removes the potential errors and the difficulty faced while checking manually. This project uses fundamental programming concepts like arrays, loops and conditional statements to automate the validation process of a solved sudoku board.

1. Introduction:

The popular Japanese puzzle game Sudoku is based on the logical placement of numbers. It is a game of logic; Sudoku doesn’t require any calculation or special math skills; all that is needed are logical thinking and concentration.

In this game, the player is given a 9x9 grid, divided into nine 3x3 sub-grids. The user is provided with a few boxes of some sub-grids of the board already filled with numbers from 1 to 9, and the user has to complete it while following sudoku rules, which are:

- Only digits from 1 to 9 can be used.
- There should be no repetition of any digit in any row.
- There should be no repetition of any digit in any column.
- There should be no repetition of any digit in any 3x3 sub-grid.

This game seems simple but as the difficulty level of the puzzle increases, the task of validating it manually becomes more difficult, time consuming and prone to errors, especially for the beginners. This difficulty can be solved by using a validator that checks if the given board follow the rules, and tells if the board is “VALID” or “INVALID”.

2. Technical Terms:

- **2D Array/Matrix**: The 9×9 grid representation.
- **Brute Force Algorithm:** Checking all rows, columns, and sub-grids systematically.
- **Nested Loops:** The multiple levels of for-loops for traversal.
- **Validation Array:** The check_r, check_c, check_g arrays.
- **Input Validation:** Checking if numbers are between 1-9
- **Modular Validation:** Separate checks for rows, columns, and sub-grids
- **User Interface (UI):** The input instructions and output messages.

3. WORKING:

Methodology:

It involved the following steps:

- **Step 1:** Input the Sudoku grid.
- **Step 2:** Check if the input given by user is within the valid range (i.e., 1 to 9).
- **Step 3:** Check all rows for duplicates.
- **Step 4:** Check all columns for duplicates.
- **Step 5:** Check each 3x3 sub-grid.
- **Step 6:** If any rule is violated, return “Invalid”, else “Valid”.

It can be further understood by the pseudo-code of our program:

Pseudocode:

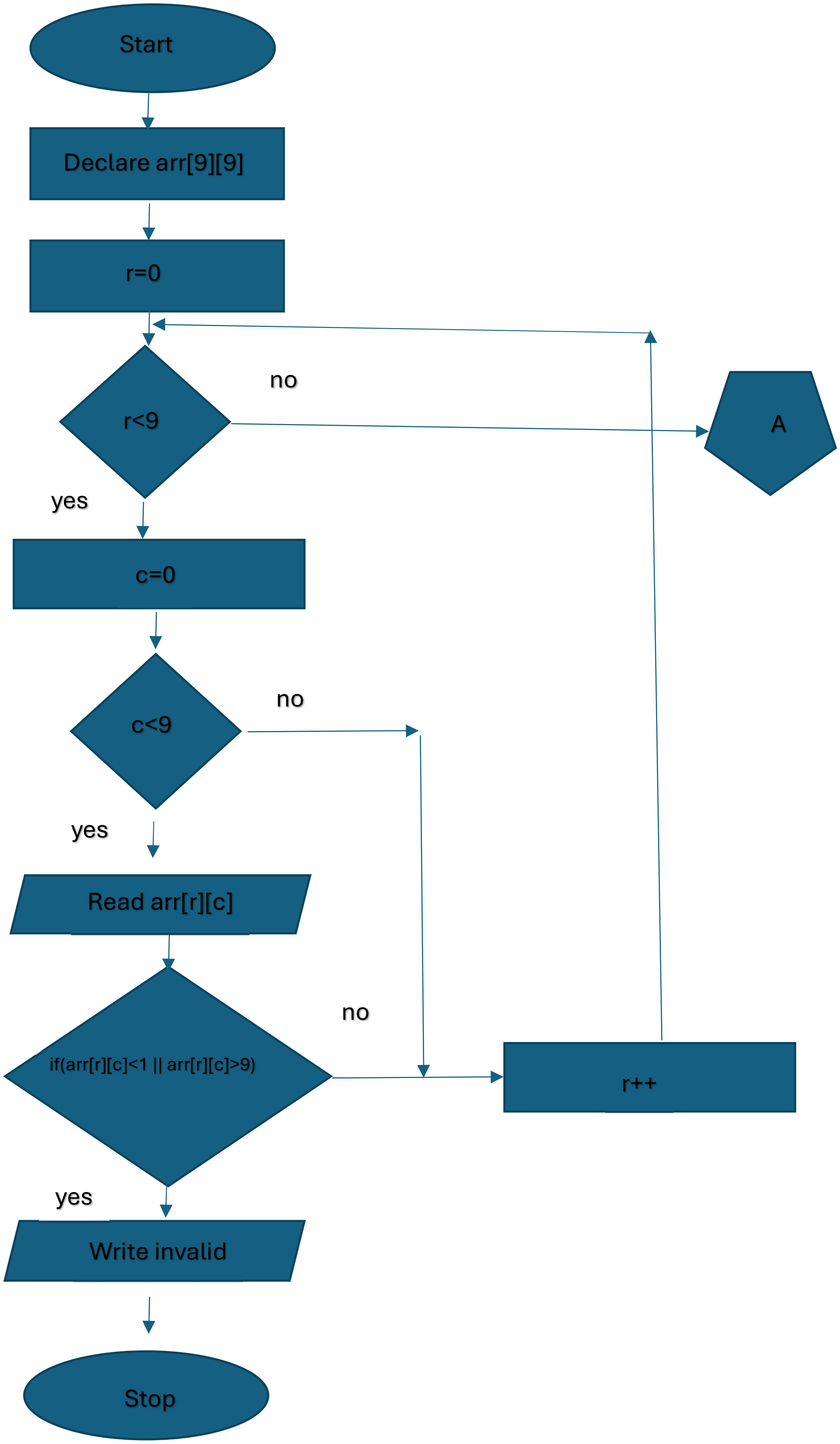
```
1) Start

2) Declare int arr[9][9] (i.e., a 2-D array named arr with integer data type and 9 by 9 size)

3) Declare integer variables r and c

    4) FOR r = 0 to r < 9, increment r by 1 after each iteration

        FOR c = 0 to c < 9, increment c by 1 after each iteration

            Read arr[r][c]

            IF arr[r][c] < 1 OR arr[r][c] > 9 THEN

              Display "INVALID INPUT!! SUDOKU RULE      VIOLATED!!
    Number must be between 1-9"

                Exit program

            End IF

        End FOR

End FOR

    5) FOR r = 0 to r < 9, increment r by 1 after each iteration

        Declare int check_r[9] = {0}

        FOR c = 0 to c < 9, increment c by 1 after each iteration

            Declare int d_r = arr[r][c]

            IF check_r[d_r - 1] == 0 THEN

                Set/Assign 1 to check_r[d_r - 1]

            ELSE

                Display "THE BOARD IS INVALID!! HINT: There's a repetition in
row %d", r + 1
```

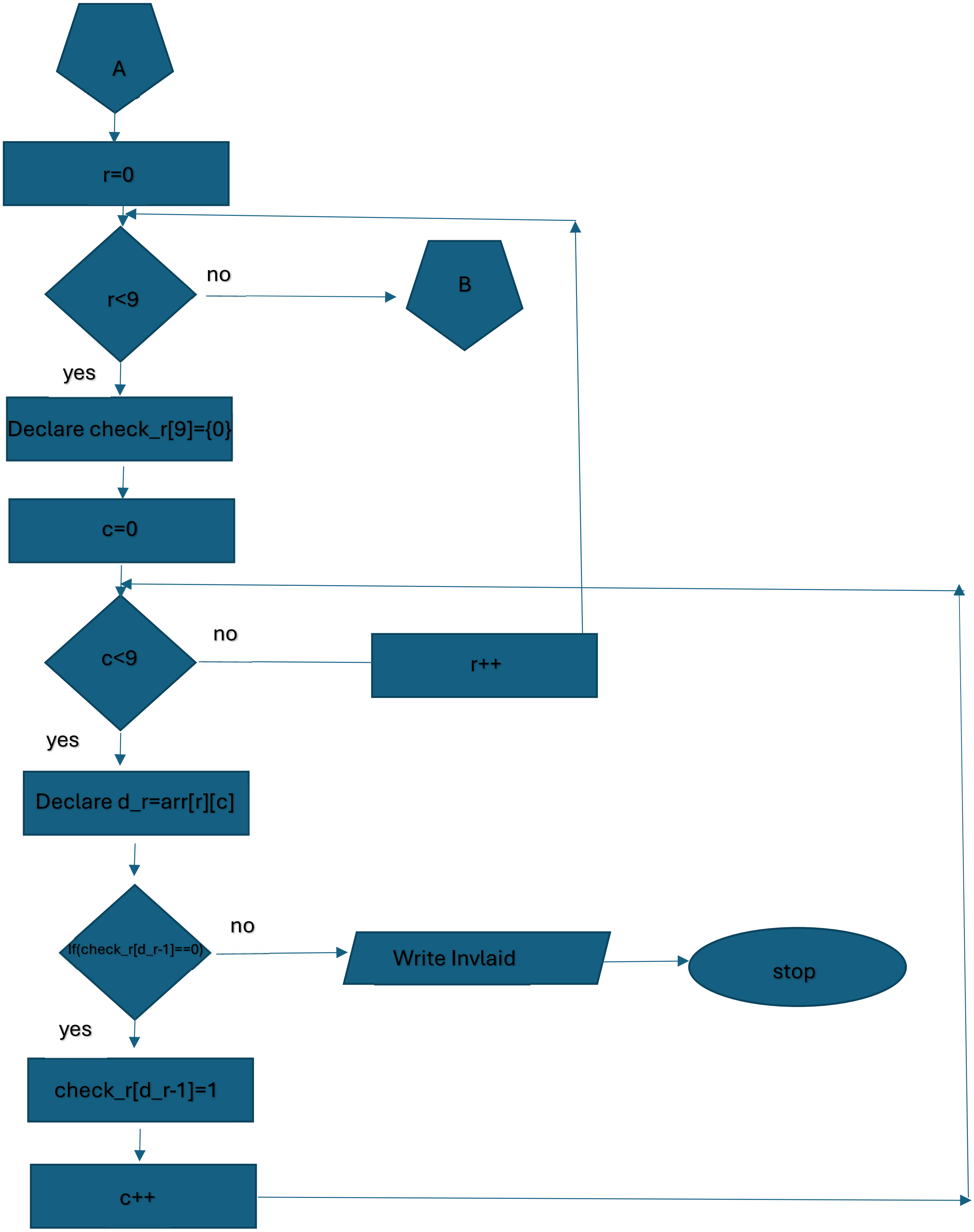
```
                Exit program
            End ELSE
        End FOR
End FOR
   6) FOR c = 0 to c < 9, increment c by 1 after each iteration
        Declare int check_c[9] = {0}
        FOR r = 0 to r < 9, increment r by 1 after each iteration
            Declare int d_c = arr[r][c]
            IF check_c[d_c - 1] == 0 THEN
                Set/Assign 1 to check_c[d_c - 1]
            ELSE
                Display "THE BOARD IS INVALID!! HINT: There's a repetition in column %d", c + 1
                Exit program
            End ELSE
        End FOR
End FOR
   7) FOR i = 0 to i < 9, increment i by 3 after each iteration
        FOR j = 0 to j < 9, increment j by 3 after each iteration
            Declare int check_g[9] = {0}
            FOR r = i to r < i + 3, increment r by 1 after each iteration
                FOR c = j to c < j + 3, increment c by 1 after each iteration
                    Declare int d_g = arr[r][c]
                    IF check_g[d_g - 1] == 0 THEN
                        Set/Assign 1 to check_g[d_g - 1]
                    ELSE
                        Display "THE BOARD IS INVALID!!"
                        IF i == 0 THEN
                            IF j == 0 THEN Display "There's a repetition in sub-grid 1"
                            ELSE IF j ==3 THEN Display "There's a repetition in sub-grid 2"
                            ELSE Display "There's a repetition in sub-grid 3"
                        ELSE IF i == 3 THEN
                            IF j == 0 THEN Display "There's a repetition in sub-grid 4"
                            ELSE IF j ==3 THEN Display "There's a repetition in sub-grid 5"
                            ELSE Display "There's a repetition in sub-grid 6"
                        ELSE
                            IF j == 0 THEN Display "There's a repetition in sub-grid 7"
                            ELSE IF j ==3 THEN Display "There's a repetition in sub-grid 8"
                            ELSE Display "There's a repetition in sub-grid 9"
                        Exit program
                    End ELSE
                End FOR
            End FOR
End FOR
   8) Display "GOOD JOB! Your board is VALID."
   9) End Program
```

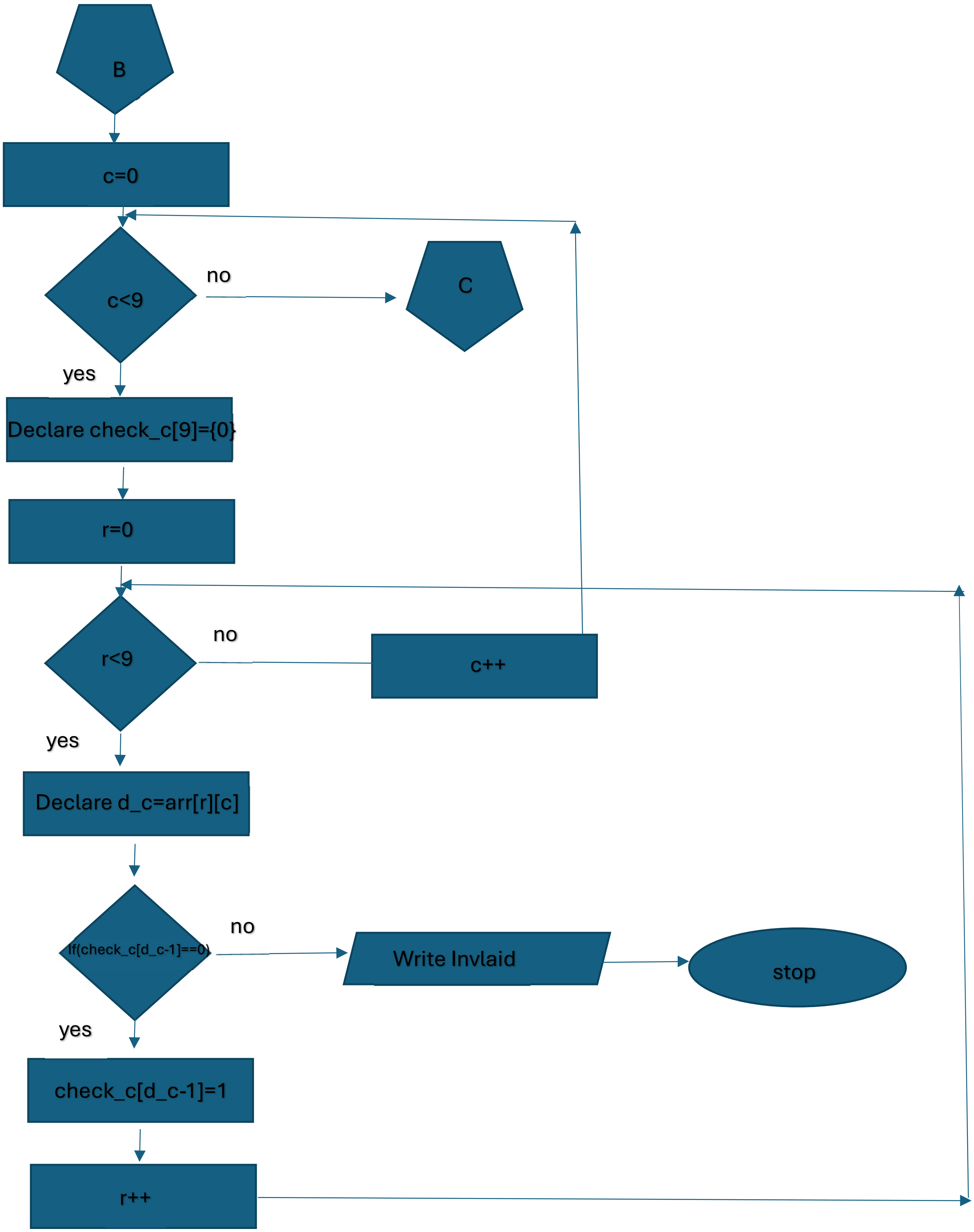
VARIABLE AND ARRAY NAMES AND WHAT THEY REPRESENT (To help understanding the pseudocode and code):

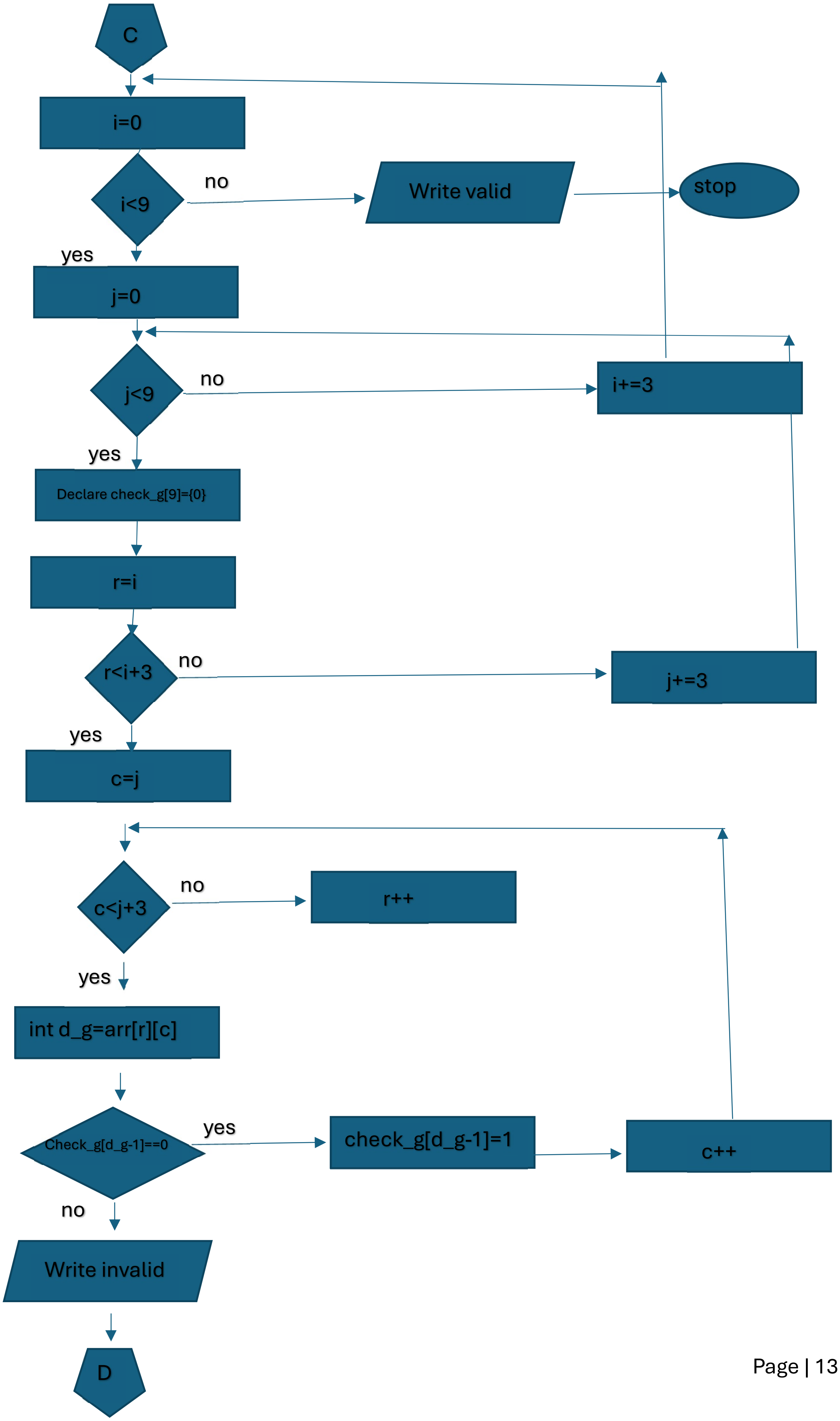
- arr : array used to take input and store the sudoku board
- check_r : an array we're using to help in checking rows.
- check_c : an array we're using to help in checking columns.
- check_g : an array we're using to help in checking sub-grids.
- d_r : digit in row
- d_c: digit in column
- d_g: digits in sub-grids.

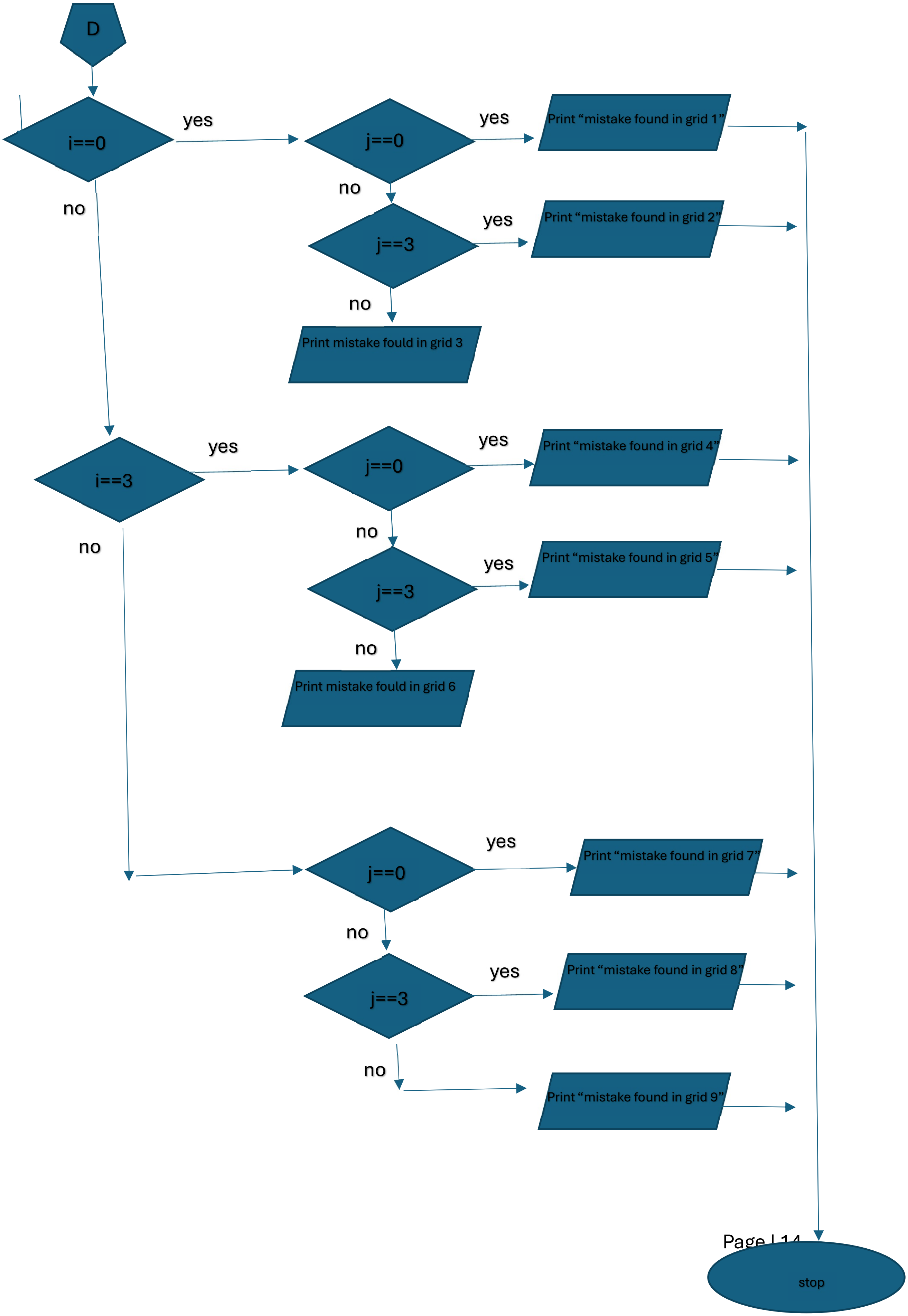
4.FLOWCHART


Start
Declare arr[9][9]
r=0
r<9
no
A
yes
c=0
c<9
no
yes
Read arr[r][c]
if(arr[r][c]<1 || arr[r][c]>9)
no
r++
yes
Write invalid
Stop

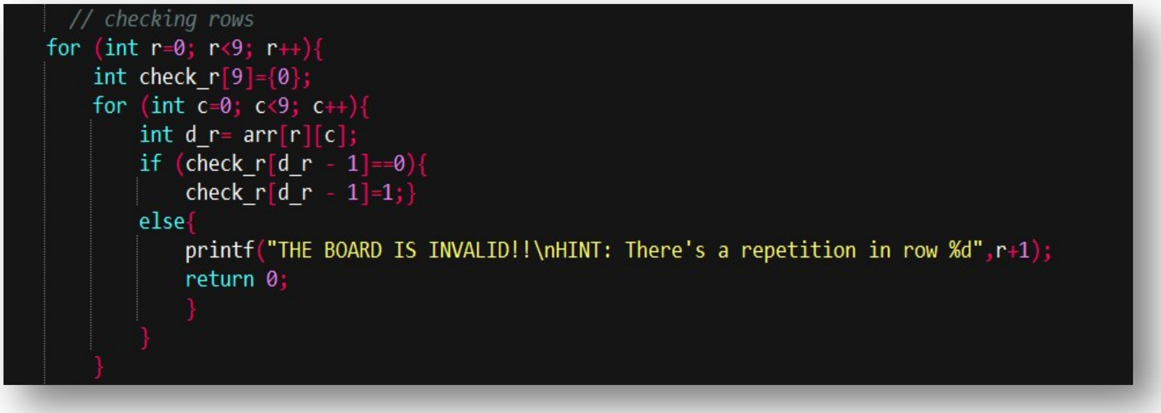

A
r=0
r<9
no
B
yes
Declare check_r[9]={0}
c=0
c<9
no
r++
yes
Declare d_r=arr[r][c]
If(check_r[d_r-1]==0)
no
Write Invlaid
stop
yes
check_r[d_r-1]=1
c++


B
c=0
c<9
no
C
yes
Declare check_c[9]={0}
r=0
r<9
no
c++
yes
Declare d_c=arr[r][c]
If(check_c[d_c-1]==0)
no
Write Invlaid
stop
yes
check_c[d_c-1]=1
r++


C
i=0
i<9
no
Write valid
stop
yes
j=0
j<9
no
i+=3
yes
Declare check_g[9]={0}
r=i
r<i+3
no
j+=3
yes
c=j
c<j+3
no
r++
yes
int d_g=arr[r][c]
Check_g[d_g-1]==0
yes
check_g[d_g-1]=1
c++
no
Write invalid
D


D
i==0
yes
j==0
yes
Print “mistake found in grid 1”
no
no
yes
j==3
Print “mistake found in grid 2”
no
Print mistake fould in grid 3
yes
i==3
yes
j==0
Print “mistake found in grid 4”
no
no
yes
j==3
Print “mistake found in grid 5”
no
Print mistake fould in grid 6
yes
j==0
Print “mistake found in grid 7”
no
yes
j==3
Print “mistake found in grid 8”
no
Print “mistake found in grid 9”
stop

5.KEY CODE SNIPPETS:

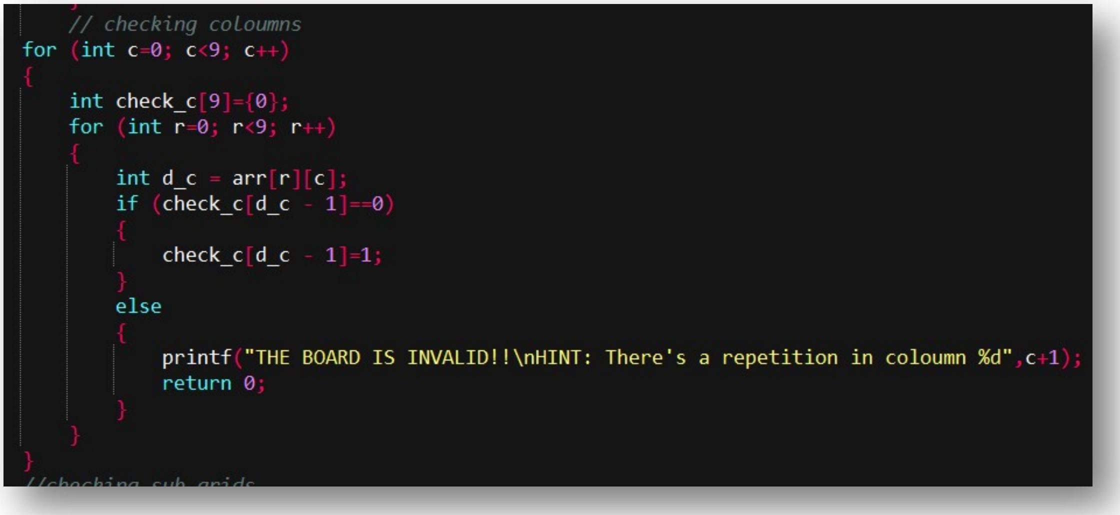


```
 // checking rows
for (int r=0; r<9; r++){
    int check_r[9]={0};
    for (int c=0; c<9; c++){
        int d_r= arr[r][c];
        if (check_r[d_r - 1]==0){
            check_r[d_r - 1]=1;}
        else{
            printf("THE BOARD IS INVALID!!\nHINT: There's a repetition in row %d",r+1);
            return 0;
            }
        }
    }
```

Explanation:

This code segment is responsible for **checking each row of the Sudoku board** to ensure it contains no repeated numbers.

- **Outer loop (for r=0; r<9; r++)** iterates through each row of the board.
- **check_r[9] array** is initialized to all zeros at the start of each row; it keeps track of which numbers (1–9) have already appeared in the current row.
- **Inner loop (for c=0; c<9; c++)** iterates through each column within the current row.
- **d_r = arr[r][c]** stores the number at the current row and column for easier reference.
- **Duplicate check:**
    - If check_r[d_r - 1] is 0, it means the number has not been seen yet, so it is marked as seen by setting it to 1.
    - If check_r[d_r - 1] is already 1, a duplicate is detected in the row. The program prints an error message specifying the row number (r+1) and terminates.

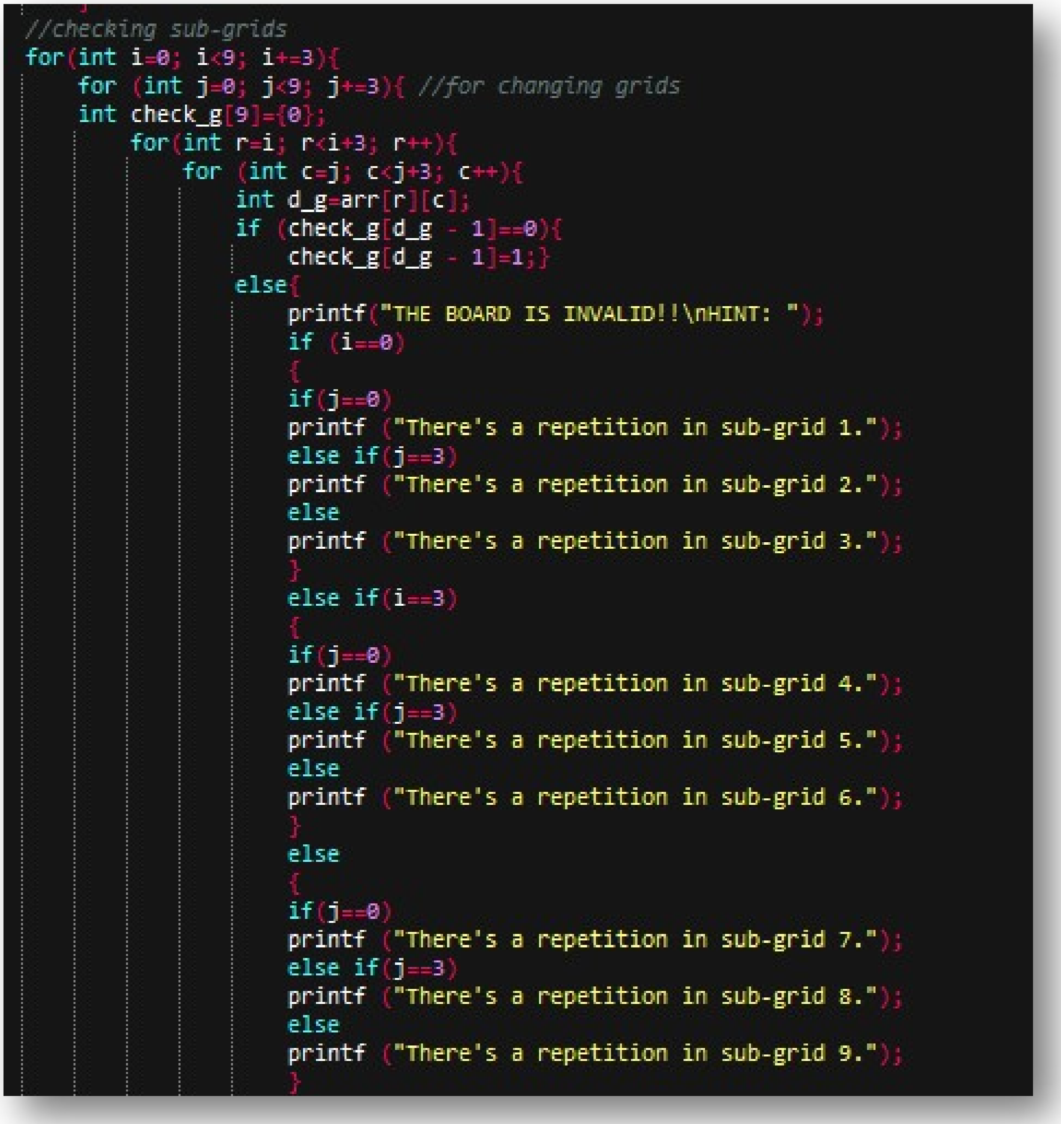


```
    // checking coloumns
for (int c=0; c<9; c++)
{
    int check_c[9]={0};
    for (int r=0; r<9; r++)
    {
        int d_c = arr[r][c];
        if (check_c[d_c - 1]==0)
        {
            check_c[d_c - 1]=1;
        }
        else
        {
            printf("THE BOARD IS INVALID!!\nHINT: There's a repetition in coloumn %d",c+1);
            return 0;
        }
    }
}
//checking sub grids
```

Explanation:

This code segment is responsible for **checking each coloumn of the Sudoku board** to ensure it contains no repeated numbers.

- **Outer loop (for c=0; c<9; c++)** iterates through each column of the board.
- **check_c[9] array** is initialized to all zeros at the start of each column; it keeps track of which numbers (1–9) have already appeared in the current column.
- **Inner loop (for r=0; r<9; r++)** iterates through each row within the current row.
- **d_c = arr[r][c]** stores the number at the current row and column for easier reference.
- **Duplicate check:**
  - If check_c[d_c - 1] is 0, it means the number has not been seen yet, so it is marked as seen by setting it to 1.
  - If check_c[d_c - 1] is already 1, a duplicate is detected in the column. The program prints an error message specifying the row number (c+1) and terminates.



```c
//checking sub-grids
for(int i=0; i<9; i+=3){
    for (int j=0; j<9; j+=3){ //for changing grids
    int check_g[9]={0};
        for(int r=i; r<i+3; r++){
            for (int c=j; c<j+3; c++){
                int d_g=arr[r][c];
                if (check_g[d_g - 1]==0){
                    check_g[d_g - 1]=1;}
                else{
                    printf("THE BOARD IS INVALID!!\nHINT: ");
                    if (i==0)
                    {
                    if(j==0)
                    printf ("There's a repetition in sub-grid 1.");
                    else if(j==3)
                    printf ("There's a repetition in sub-grid 2.");
                    else
                    printf ("There's a repetition in sub-grid 3.");
                    }
                    else if(i==3)
                    {
                    if(j==0)
                    printf ("There's a repetition in sub-grid 4.");
                    else if(j==3)
                    printf ("There's a repetition in sub-grid 5.");
                    else
                    printf ("There's a repetition in sub-grid 6.");
                    }
                    else
                    {
                    if(j==0)
                    printf ("There's a repetition in sub-grid 7.");
                    else if(j==3)
                    printf ("There's a repetition in sub-grid 8.");
                    else
                    printf ("There's a repetition in sub-grid 9.");
                    }
```

This code segment verifies that each 3×3 sub-grid of the Sudoku board contains unique numbers without repetition.

- **Outer loops** (for i and for j) move through the board in steps of 3 to cover all nine sub-grids.
- **check_g[9]** array is initialized to zeros for every sub-grid to track which numbers (1–9) have appeared.
- **Nested loops** (for r and for c) iterate through each cell inside the current 3×3 sub-grid.
- **d_g = arr[r][c]** stores the current number being checked.

- **Duplicate check:**
  - If check_g[d_g - 1] is 0, it marks the number as seen by setting it to 1.
  - If it's already 1, a repetition is detected.

- The program then identifies the sub-grid (1–9) where the duplication occurred and displays an error message before exiting.

6.CONCLUSION:

This program successfully validates a Sudoku board by ensuring that all Sudoku rules are followed i.e., each row, column, and 3×3 sub-grid must contain unique numbers between 1 and 9 without any repetition. The logic systematically checks every part of the grid, immediately identifying and reporting any violations with clear hints for easier correction making it user- friendly. Hence, the program serves as a reliable and well-structured solution for verifying the correctness of a Sudoku board.