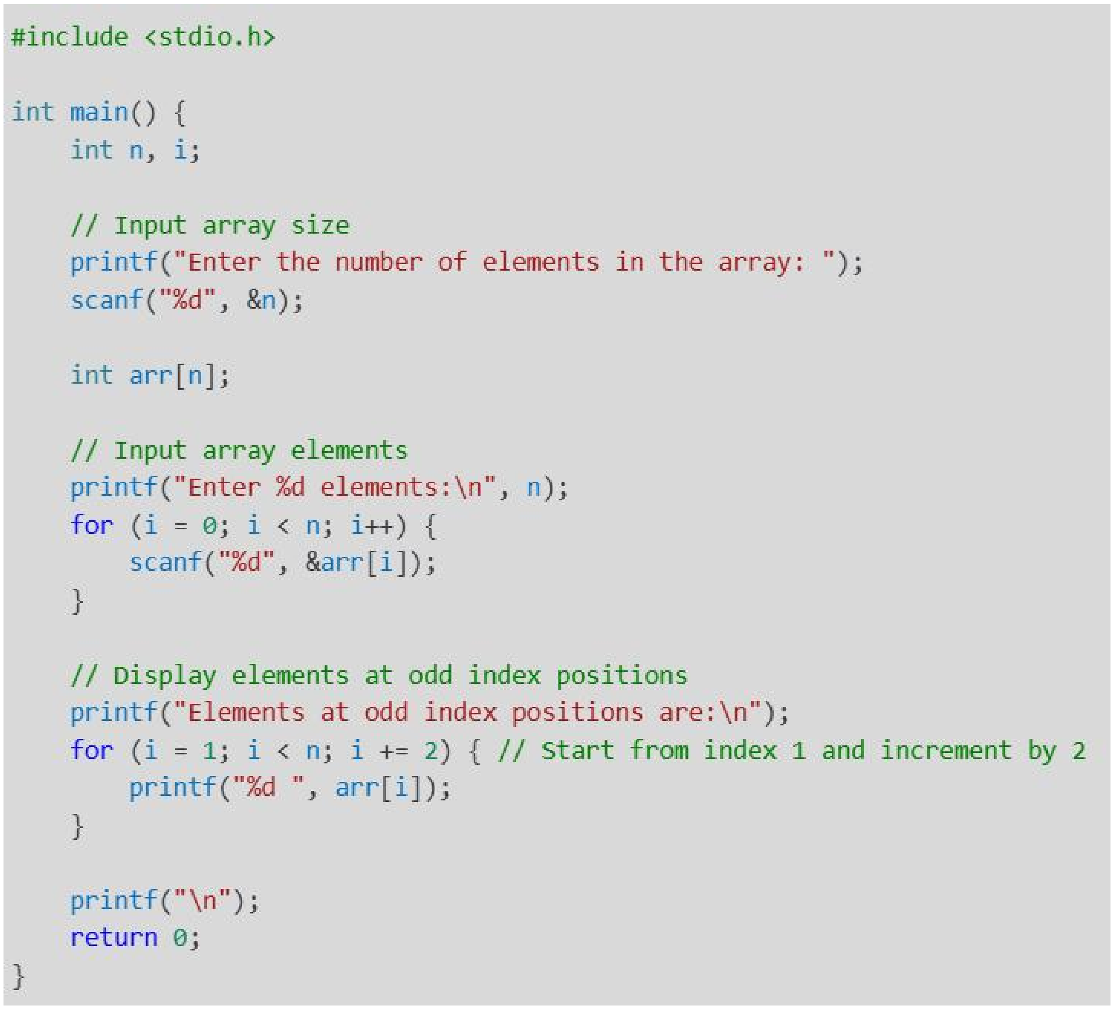
1. a)Write a C program to count the number of occurrences of a given number in an array of n numbers.(5 Marks)

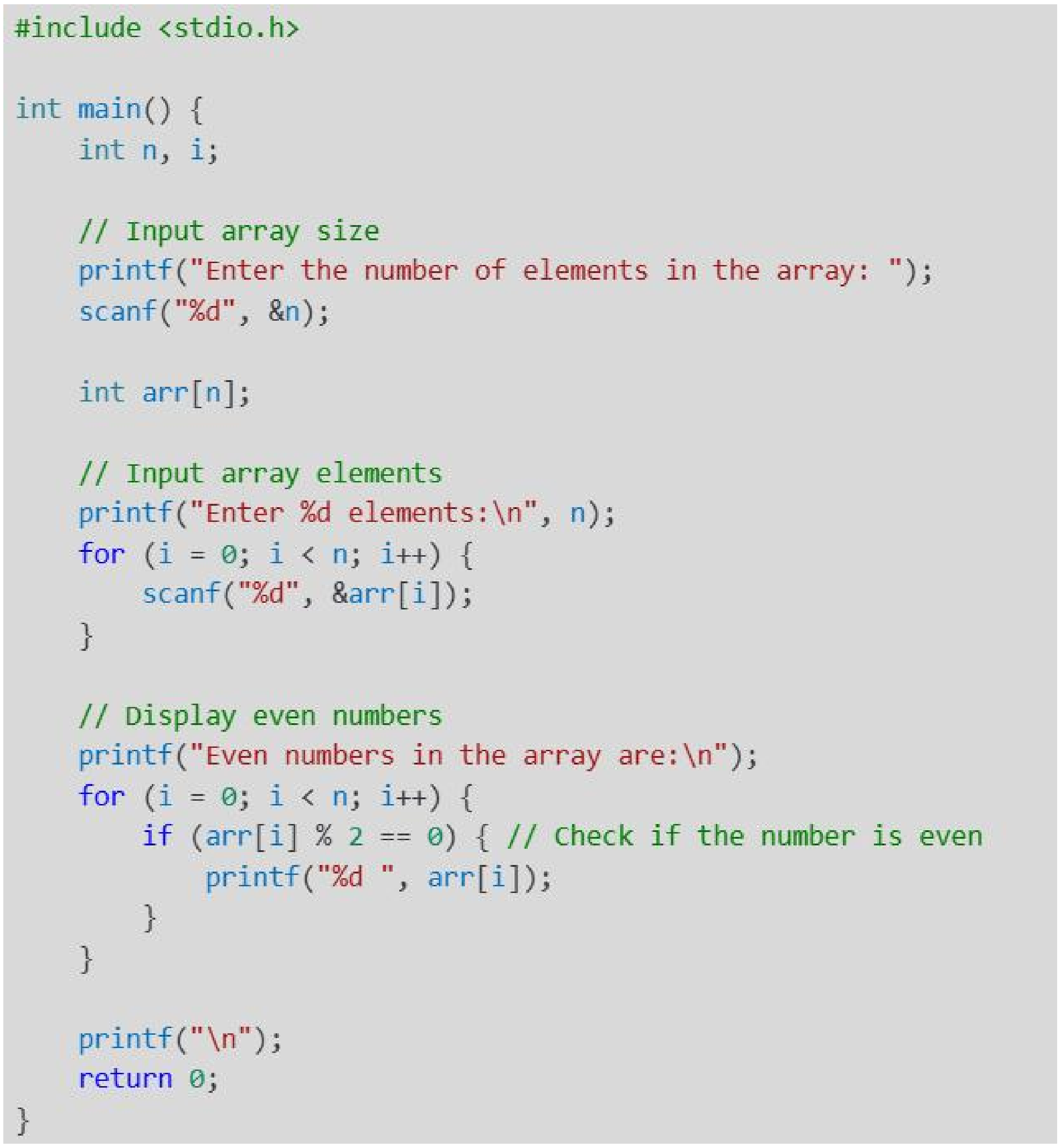
```c
#include <stdio.h>

int main() {
    int n, i, search, count = 0;

    // Input array size
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];

    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Input the number to search
    printf("Enter the number to search: ");
    scanf("%d", &search);

    // Count occurrences
    for (i = 0; i < n; i++) {
        if (arr[i] == search) {
            count++;
        }
    }

    // Output result
    printf("The number %d occurs %d times in the array.\n", search, count);

    return 0;
}
```

b)Write a C program to display all the elements from an odd index position.(4 Marks)



```c
#include <stdio.h>

int main() {
    int n, i;

    // Input array size
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];

    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Display elements at odd index positions
    printf("Elements at odd index positions are:\n");
    for (i = 1; i < n; i += 2) { // Start from index 1 and increment by 2
        printf("%d ", arr[i]);
    }

    printf("\n");
    return 0;
}
```

2. a)Write a C program to display all the even numbers from an array.(5 Marks)

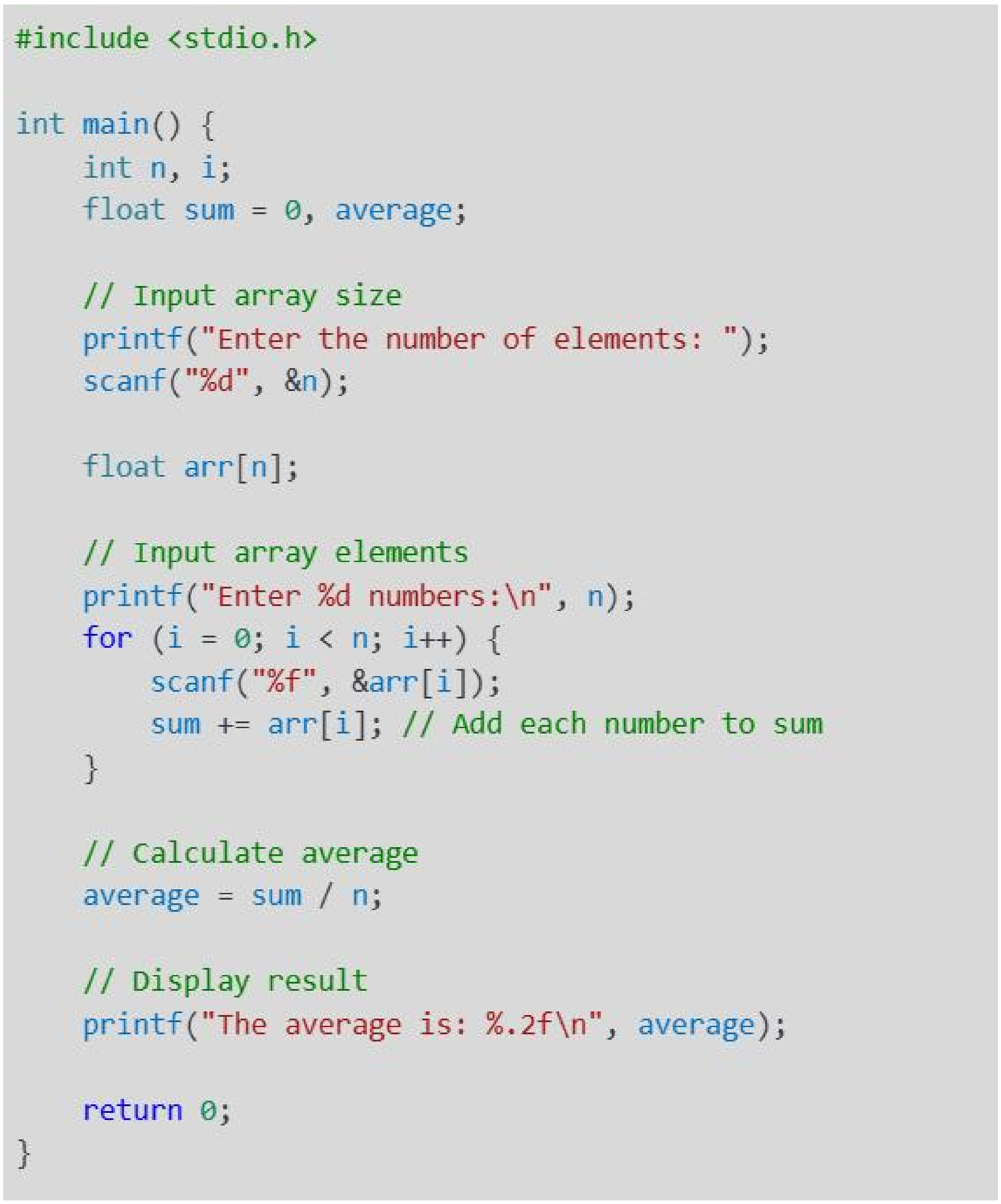


```c
#include <stdio.h>

int main() {
    int n, i;

    // Input array size
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];

    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Display even numbers
    printf("Even numbers in the array are:\n");
    for (i = 0; i < n; i++) {
        if (arr[i] % 2 == 0) { // Check if the number is even
            printf("%d ", arr[i]);
        }
    }

    printf("\n");
    return 0;
}
```

b)Write a program to count and display all the odd numbers in an array.(4 Marks)

```c
#include <stdio.h>

int main() {
    int n, i, count = 0;

    // Input array size
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];

    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Display odd numbers and count them
    printf("Odd numbers in the array are:\n");
    for (i = 0; i < n; i++) {
        if (arr[i] % 2 != 0) { // Check if the number is odd
            printf("%d ", arr[i]);
            count++;
        }
    }

    // Display count of odd numbers
    printf("\nTotal number of odd numbers: %d\n", count);

    return 0;
}
```

3. Write a C program to calculate the average of n numbers stored in an array.



```c
#include <stdio.h>

int main() {
    int n, i;
    float sum = 0, average;

    // Input array size
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    float arr[n];

    // Input array elements
    printf("Enter %d numbers:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%f", &arr[i]);
        sum += arr[i]; // Add each number to sum
    }

    // Calculate average
    average = sum / n;

    // Display result
    printf("The average is: %.2f\n", average);

    return 0;
}
```

4.What is an array? Illustrate using an example, how a single dimensional array is initialised.

An **array** is a collection of elements of the same data type stored in **contiguous memory locations**. It allows multiple values to be stored under a single variable name and accessed using an index.

```c
#include <stdio.h>

int main() {
    // Declaring and initializing an array
    int numbers[5] = {10, 20, 30, 40, 50};

    // Accessing and printing array elements
    printf("First element: %d\n", numbers[0]);
    printf("Second element: %d\n", numbers[1]);

    return 0;
}
```

5.Write a C program to find average marks obtained by a class of 50 students in a test using array.

```c
#include <stdio.h>

int main() {
    int marks[50], i;
    float sum = 0, average;

    // Input marks of 50 students
    printf("Enter the marks of 50 students:\n");
    for (i = 0; i < 50; i++) {
        scanf("%d", &marks[i]);
        sum += marks[i]; // Add each mark to sum
    }

    // Calculate average
    average = sum / 50;

    // Display result
    printf("The average marks of the class is: %.2f\n", average);

    return 0;
}
```

6.What are the different ways of declaring and initialising a single dimensional array?

## 1. Declaration without Initialization

Here, the array is only declared, and the elements are assigned later.

```
#include <stdio.h>

int main() {
    int numbers[5]; // Declaration

    // Assigning values later
    numbers[0] = 10;
    numbers[1] = 20;
    numbers[2] = 30;
    numbers[3] = 40;
    numbers[4] = 50;

    printf("Element at index 0: %d\n", numbers[0]);
    return 0;
}
```

## 2. Declaration with Initialization at the Time of Definition

Here, the array is declared and initialized at the same time.

```
#include <stdio.h>



int main() {

    int numbers[5] = {10, 20, 30, 40, 50}; // Declaration and Initialization



    printf("Element at index 0: %d\n", numbers[0]);

    return 0;

}
```

## 3. Without Size Specification

The compiler automatically determines the size based on the number of elements provided.

```
#include <stdio.h>


int main() {

    int numbers[] = {10, 20, 30, 40, 50}; // Size will be 5 automatically


    printf("Element at index 3: %d\n", numbers[3]);

    return 0;

}
```

## 4. Using Loop for Initialization

Elements can be assigned values using a loop.

```
#include <stdio.h>

int main() {

    int numbers[5];

    for(int i = 0; i < 5; i++) {

        numbers[i] = i * 10; // Assigning values

    }

    printf("Element at index 4: %d\n", numbers[4]); // Will print 40

    return 0;

}
```

7. Explain linear search with an example.

## Linear Search in C

**Definition:**

Linear search (or sequential search) is a simple searching algorithm that checks each element of an array sequentially until the desired value is found or the array ends.

## Working Principle:

1. Start from the first element of the array.
2. Compare each element with the target value.
3. If a match is found, return the index.
4. If the element is not found, return -1.

## Example: Linear Search in C

```c
#include <stdio.h>

// Function to perform Linear Search
int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) { // Check if element matches the target
            return i; // Return the index if found
        }
    }
    return -1; // Return -1 if element not found
}

int main() {
    int numbers[] = {10, 20, 30, 40, 50}; // Array declaration
    int size = sizeof(numbers) / sizeof(numbers[0]); // Calculate array size
    int target = 30; // Element to search

    int result = linearSearch(numbers, size, target);

    if (result != -1) {
        printf("Element %d found at index %d\n", target, result);
    } else {
        printf("Element %d not found in the array\n", target);
    }

    return 0;
}
```

## Explanation:

1. The function linearSearch() loops through the array.
2. If an element matches the target value, it **returns the index**.
3. If no match is found, it **returns -1**.
4. The main() function initializes an array and calls linearSearch() with a target value.
5. The result is printed, indicating whether the element is found or not.

## Advantages of Linear Search:

✔ Works on both sorted and unsorted arrays.
✔ Simple and easy to implement.

## Disadvantages of Linear Search:

❌ Slow for large datasets.
❌ Inefficient compared to other searching algorithms like **Binary Search**

8. Write a C Program to sort an array and display its largest, second largest and smallest element.

```
#include <stdio.h>



// Function to sort the array using Bubble Sort

void sortArray(int arr[], int n) {

    int i, j, temp;

    for (i = 0; i < n - 1; i++) {

        for (j = 0; j < n - i - 1; j++) {

            if (arr[j] > arr[j + 1]) {

                // Swap elements

                temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

}

int main() {

    int n, i;
    // Input array size
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    // Sort the array
    sortArray(arr, n);
    // Display sorted array
    printf("Sorted array: ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    // Display results
    printf("\nSmallest element: %d", arr[0]);
    printf("\nSecond largest element: %d", arr[n - 2]);
    printf("\nLargest element: %d\n", arr[n - 1]);
    return 0;
}
```

9. Write a C program to sort an array using bubble sort.

```c
#include <stdio.h>
// Function to perform Bubble Sort
void bubbleSort(int arr[], int n) {
    int i, j, temp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap elements
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
int main() {
    int n, i;
    // Input array size
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    // Sort the array using Bubble Sort
    bubbleSort(arr, n);

    // Display the sorted array
    printf("Sorted array: ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

10. Write a C program to perform linear search on an array of numbers.

```c
#include <stdio.h>

// Function to perform Linear Search
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i; // Return the index if the element is found
        }
    }
    return -1; // Return -1 if the element is not found
}
int main() {
    int n, key, i, result;

    // Input array size
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    // Input the number to search
    printf("Enter the number to search: ");
    scanf("%d", &key);
    // Perform Linear Search
    result = linearSearch(arr, n, key);
    // Display result
    if (result != -1) {
        printf("Element %d found at index %d\n", key, result);
    } else {
        printf("Element %d not found in the array\n", key);
    }
    return 0;
}
```

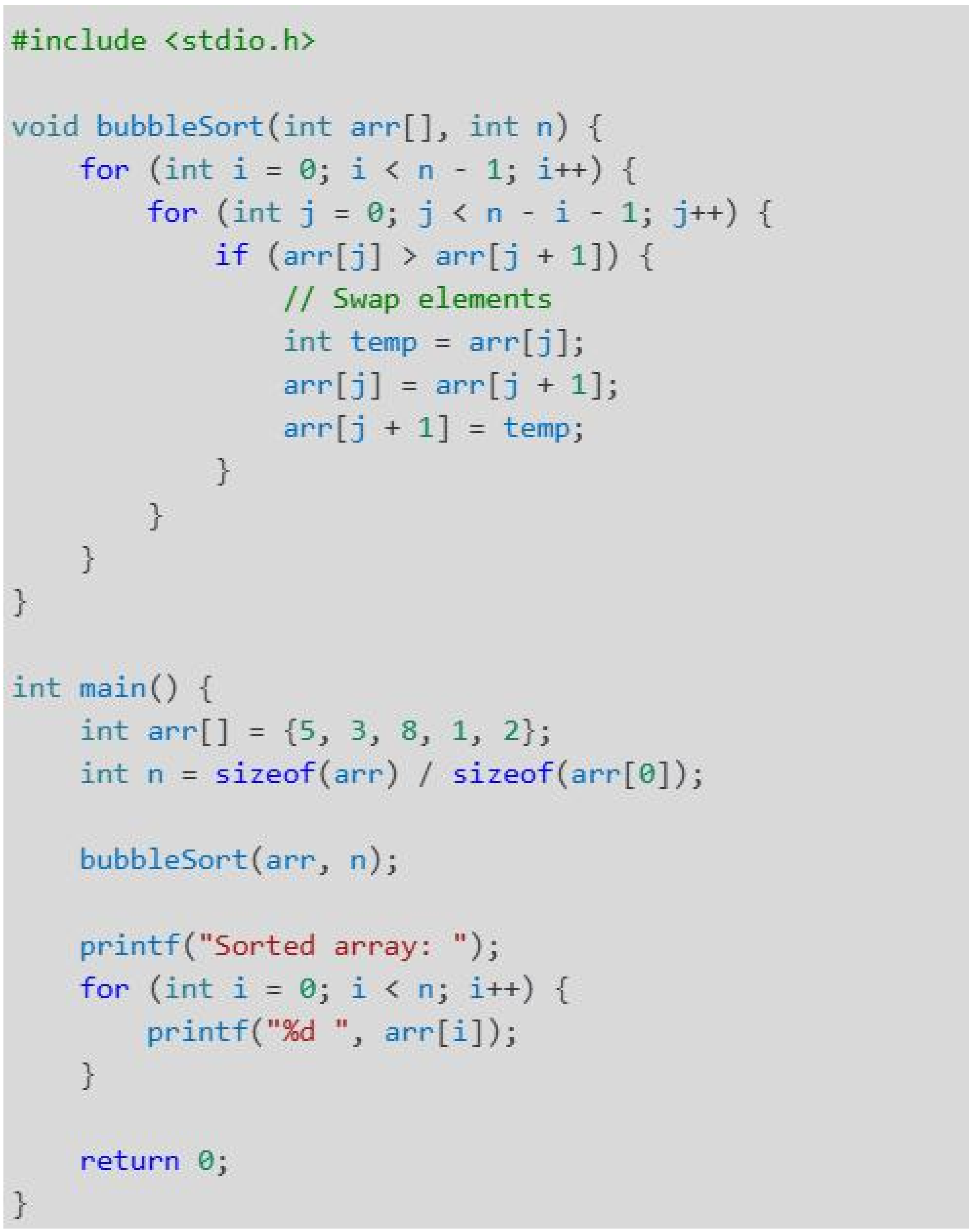
11. Explain the concept of Bubble Sort with an example.

## Bubble Sort in C

Bubble Sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted.

## Algorithm Steps

1. Compare adjacent elements.
2. Swap them if they are in the wrong order.
3. Repeat this process for all elements.
4. Continue the process until no swaps are needed (i.e., the list is sorted).

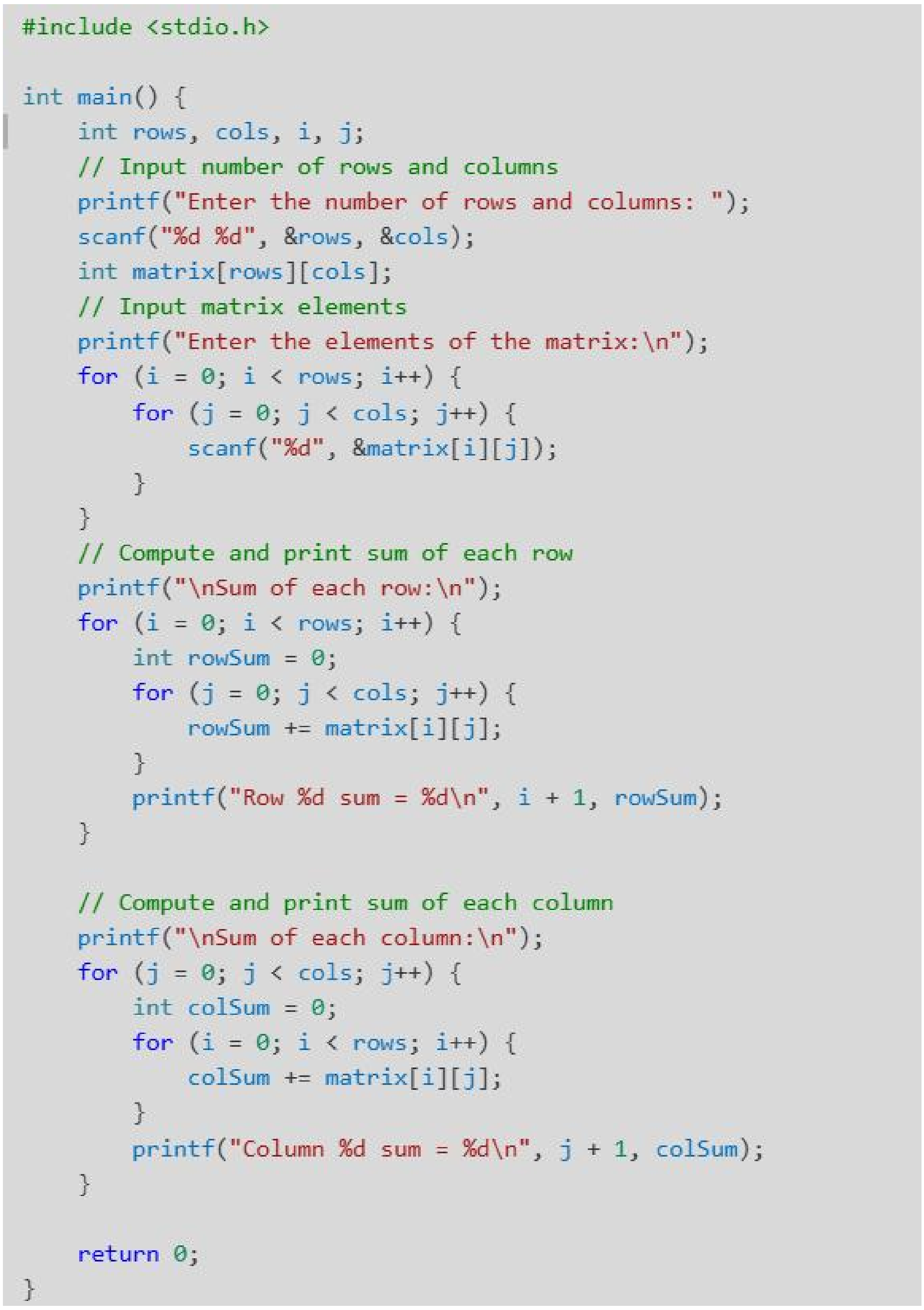


```c
#include <stdio.h>

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap elements
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int arr[] = {5, 3, 8, 1, 2};
    int n = sizeof(arr) / sizeof(arr[0]);

    bubbleSort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

## 12. Write a C program to find the largest element in an array

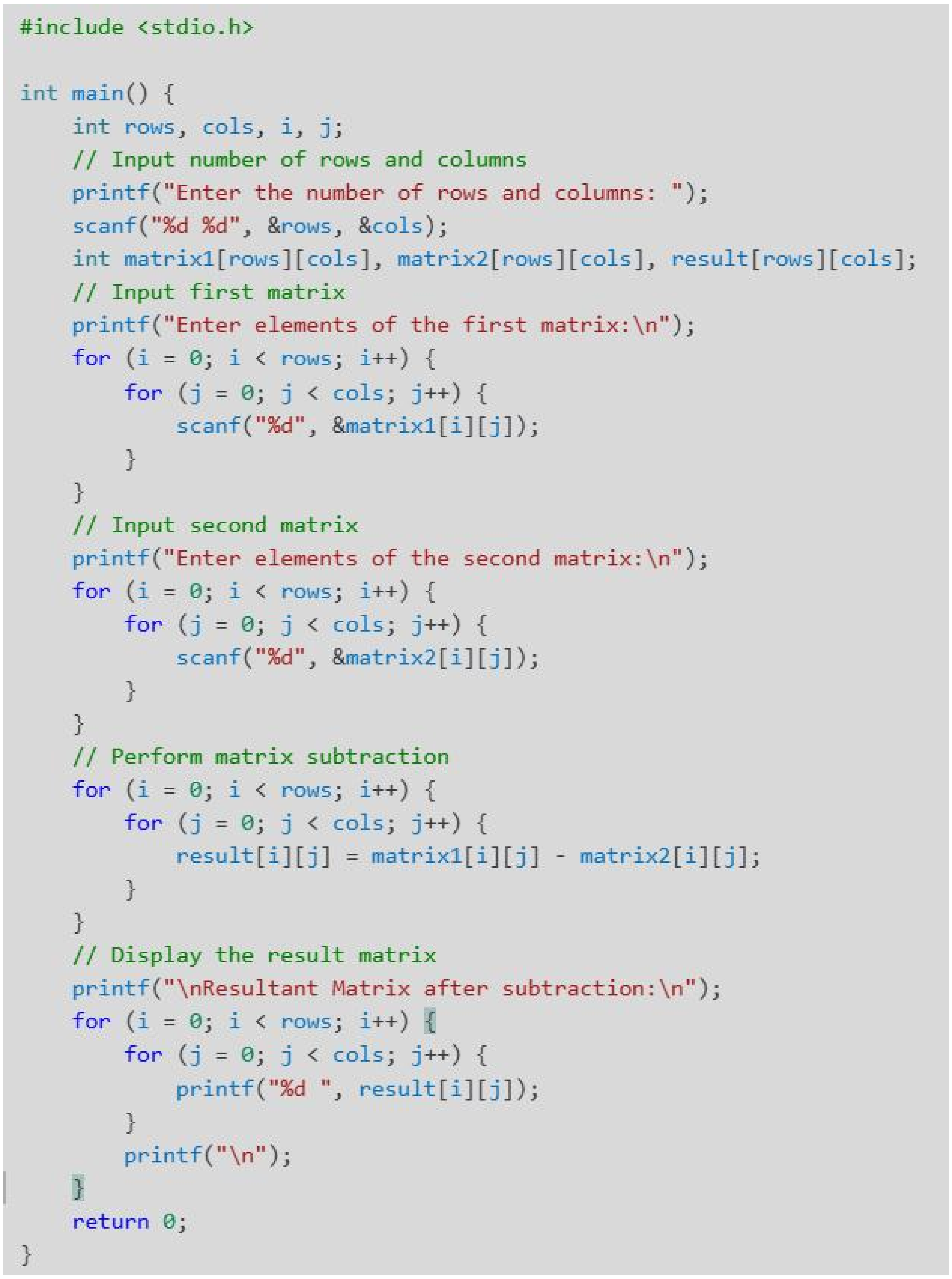
```c
#include <stdio.h>

int main() {
    int n, i, largest;

    // Input array size
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];

    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Assume first element is the largest
    largest = arr[0];

    // Iterate through the array to find the largest element
    for (i = 1; i < n; i++) {
        if (arr[i] > largest) {
            largest = arr[i];
        }
    }

    // Display the largest element
    printf("The largest element in the array is: %d\n", largest);

    return 0;
}
```

## 13. Write a C program that takes a matrix as input and prints the sum of each row and each column.



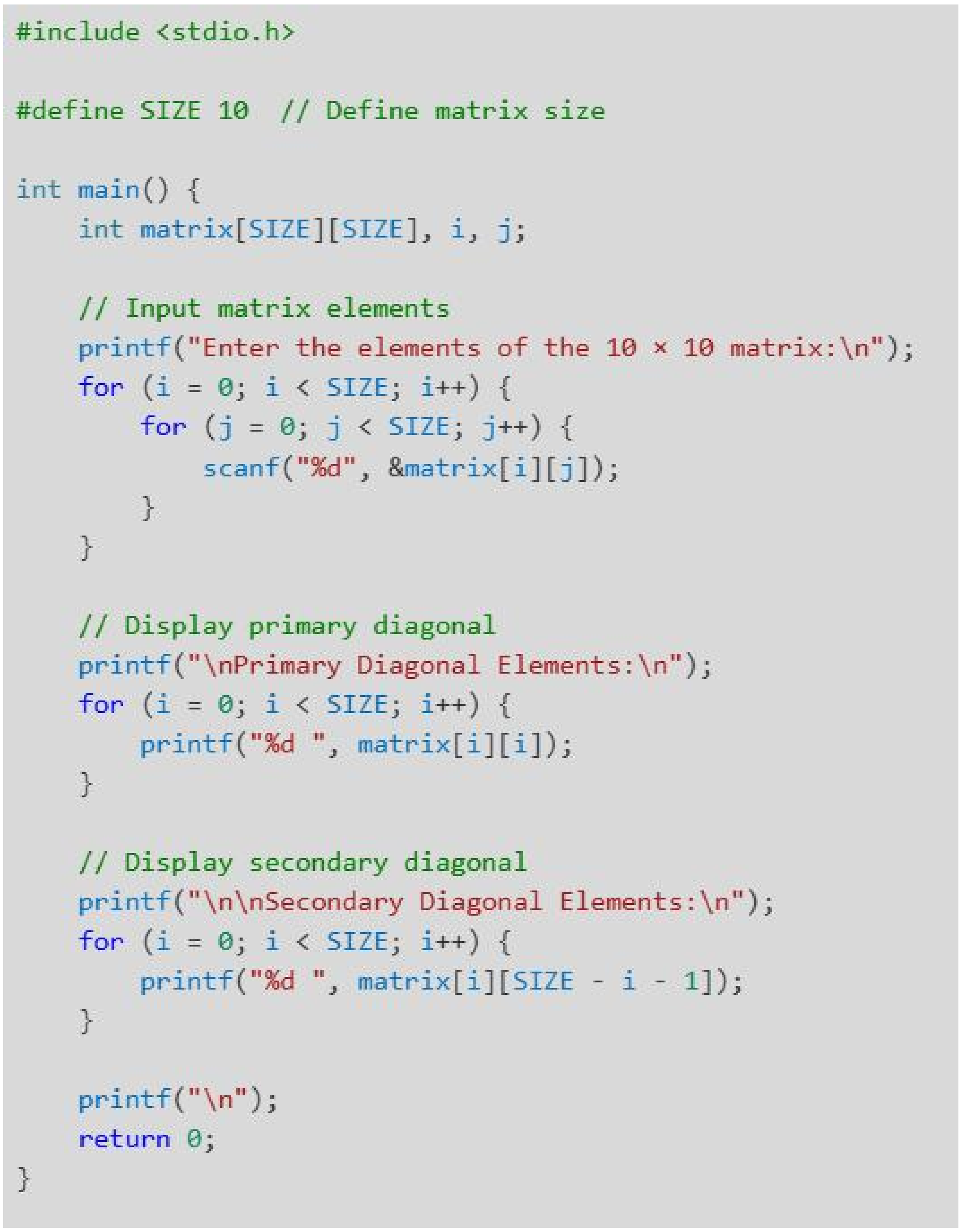
```c
#include <stdio.h>

int main() {
    int rows, cols, i, j;
    // Input number of rows and columns
    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &rows, &cols);
    int matrix[rows][cols];
    // Input matrix elements
    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    // Compute and print sum of each row
    printf("\nSum of each row:\n");
    for (i = 0; i < rows; i++) {
        int rowSum = 0;
        for (j = 0; j < cols; j++) {
            rowSum += matrix[i][j];
        }
        printf("Row %d sum = %d\n", i + 1, rowSum);
    }

    // Compute and print sum of each column
    printf("\nSum of each column:\n");
    for (j = 0; j < cols; j++) {
        int colSum = 0;
        for (i = 0; i < rows; i++) {
            colSum += matrix[i][j];
        }
        printf("Column %d sum = %d\n", j + 1, colSum);
    }

    return 0;
}
```

14. Write a C program that takes two matrices as input and performs matrix subtraction.



```c
#include <stdio.h>

int main() {
    int rows, cols, i, j;
    // Input number of rows and columns
    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &rows, &cols);
    int matrix1[rows][cols], matrix2[rows][cols], result[rows][cols];
    // Input first matrix
    printf("Enter elements of the first matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }
    // Input second matrix
    printf("Enter elements of the second matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }
    // Perform matrix subtraction
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            result[i][j] = matrix1[i][j] - matrix2[i][j];
        }
    }
    // Display the result matrix
    printf("\nResultant Matrix after subtraction:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

15. Write a C program to display the diagonal values of a 10 × 10 matrix.



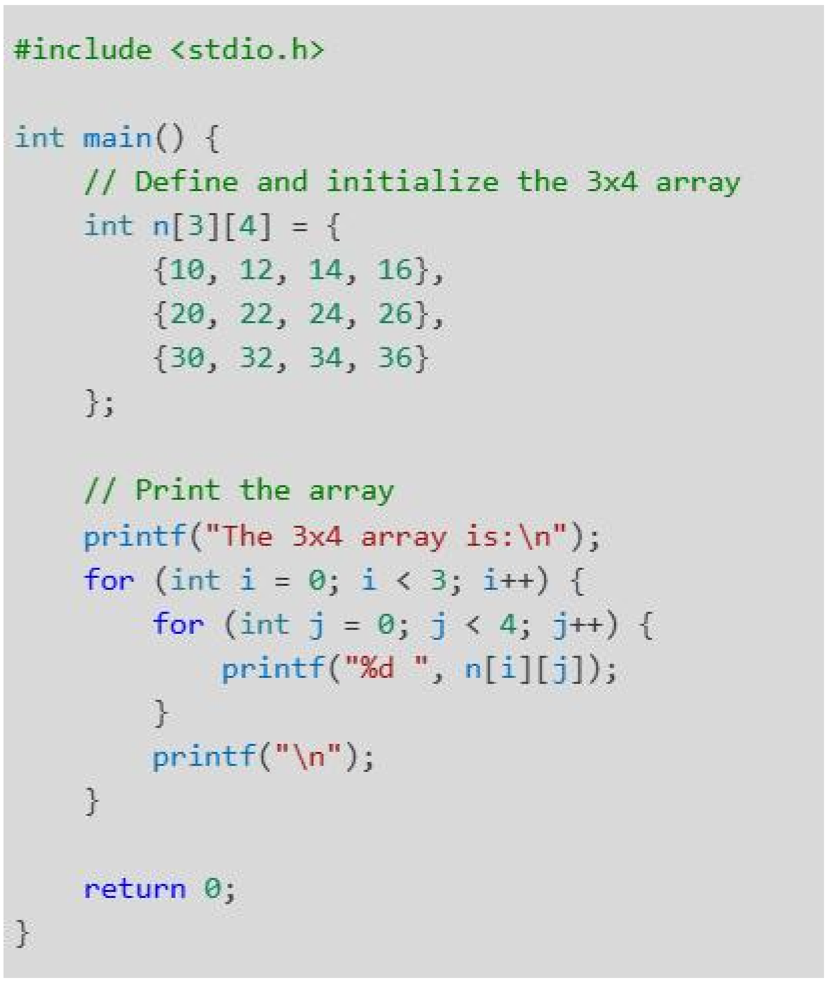
```c
#include <stdio.h>

#define SIZE 10  // Define matrix size

int main() {
    int matrix[SIZE][SIZE], i, j;

    // Input matrix elements
    printf("Enter the elements of the 10 × 10 matrix:\n");
    for (i = 0; i < SIZE; i++) {
        for (j = 0; j < SIZE; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    // Display primary diagonal
    printf("\nPrimary Diagonal Elements:\n");
    for (i = 0; i < SIZE; i++) {
        printf("%d ", matrix[i][i]);
    }

    // Display secondary diagonal
    printf("\n\nSecondary Diagonal Elements:\n");
    for (i = 0; i < SIZE; i++) {
        printf("%d ", matrix[i][SIZE - i - 1]);
    }

    printf("\n");
    return 0;
}
```

16. Define a 3 × 4 two-dimensional integer array called n. Assign the following values to its elements.

10 12 14 16

20 22 24 26

30 32 34 36



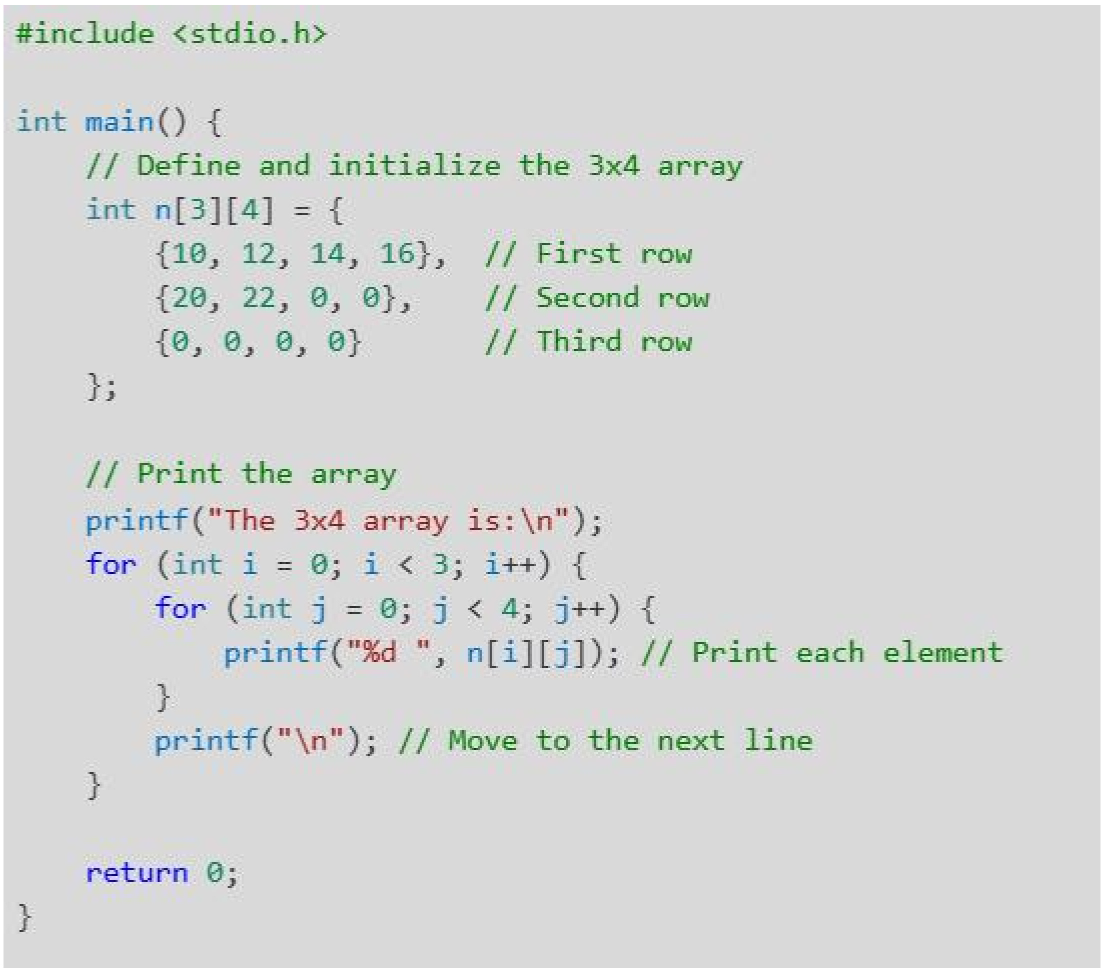
```c
#include <stdio.h>

int main() {
    // Define and initialize the 3x4 array
    int n[3][4] = {
        {10, 12, 14, 16},
        {20, 22, 24, 26},
        {30, 32, 34, 36}
    };

    // Print the array
    printf("The 3x4 array is:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%d ", n[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

17. Define a 3 × 4 two-dimensional integer array called n. Assign the following values to its elements.

10 12 14 16

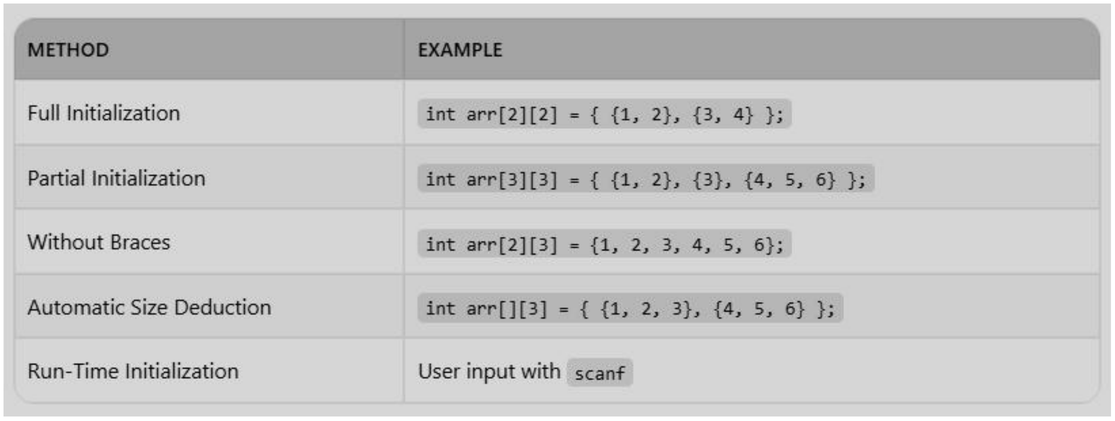
20 22 0 0

0 0 0 0



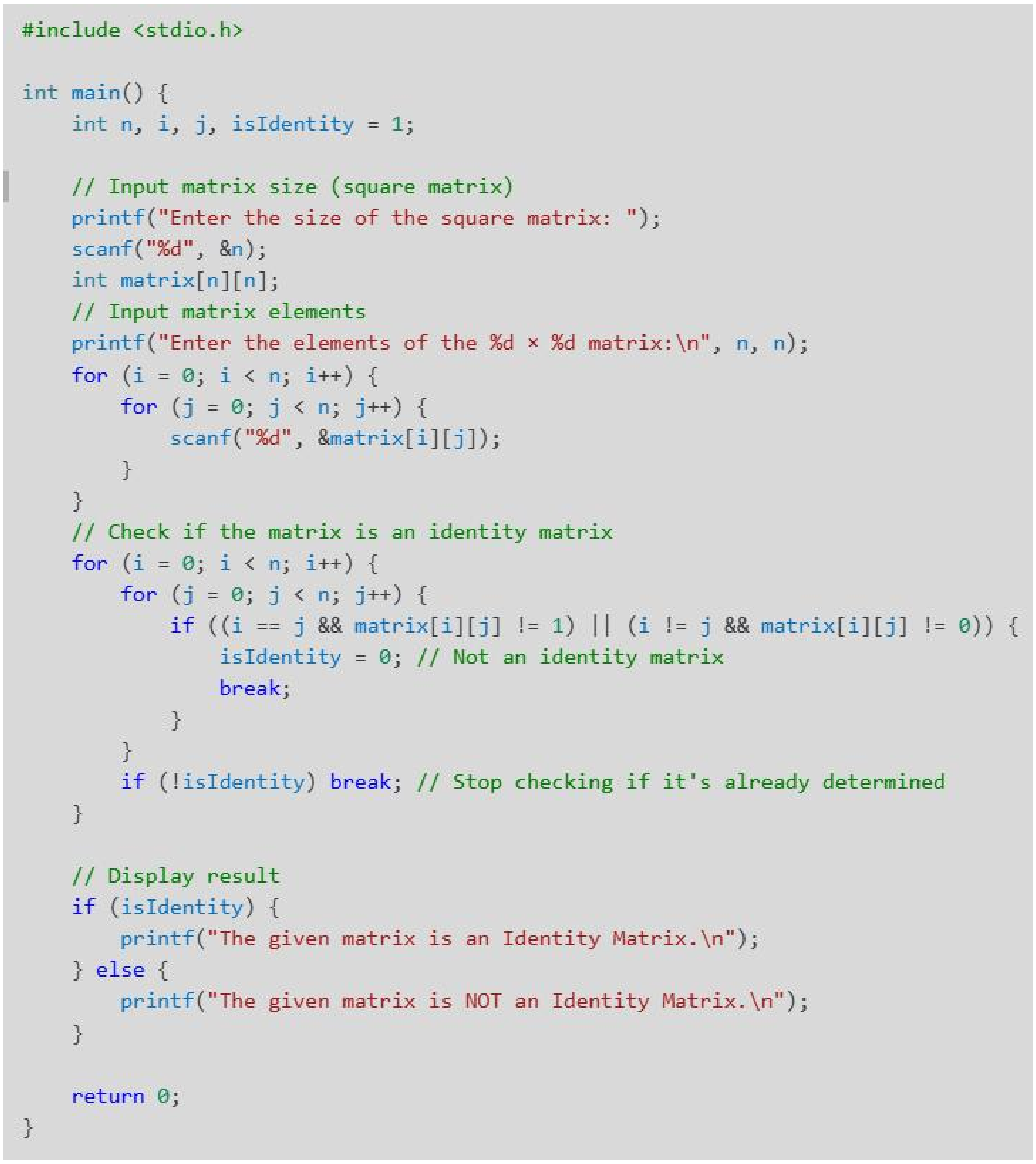
```c
#include <stdio.h>

int main() {
    // Define and initialize the 3x4 array
    int n[3][4] = {
        {10, 12, 14, 16},   // First row
        {20, 22, 0, 0},     // Second row
        {0, 0, 0, 0}        // Third row
    };

    // Print the array
    printf("The 3x4 array is:\n");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%d ", n[i][j]); // Print each element
        }
        printf("\n"); // Move to the next line
    }

    return 0;
}
```

## 18. Explain different ways of initializing a 2D array with examples.



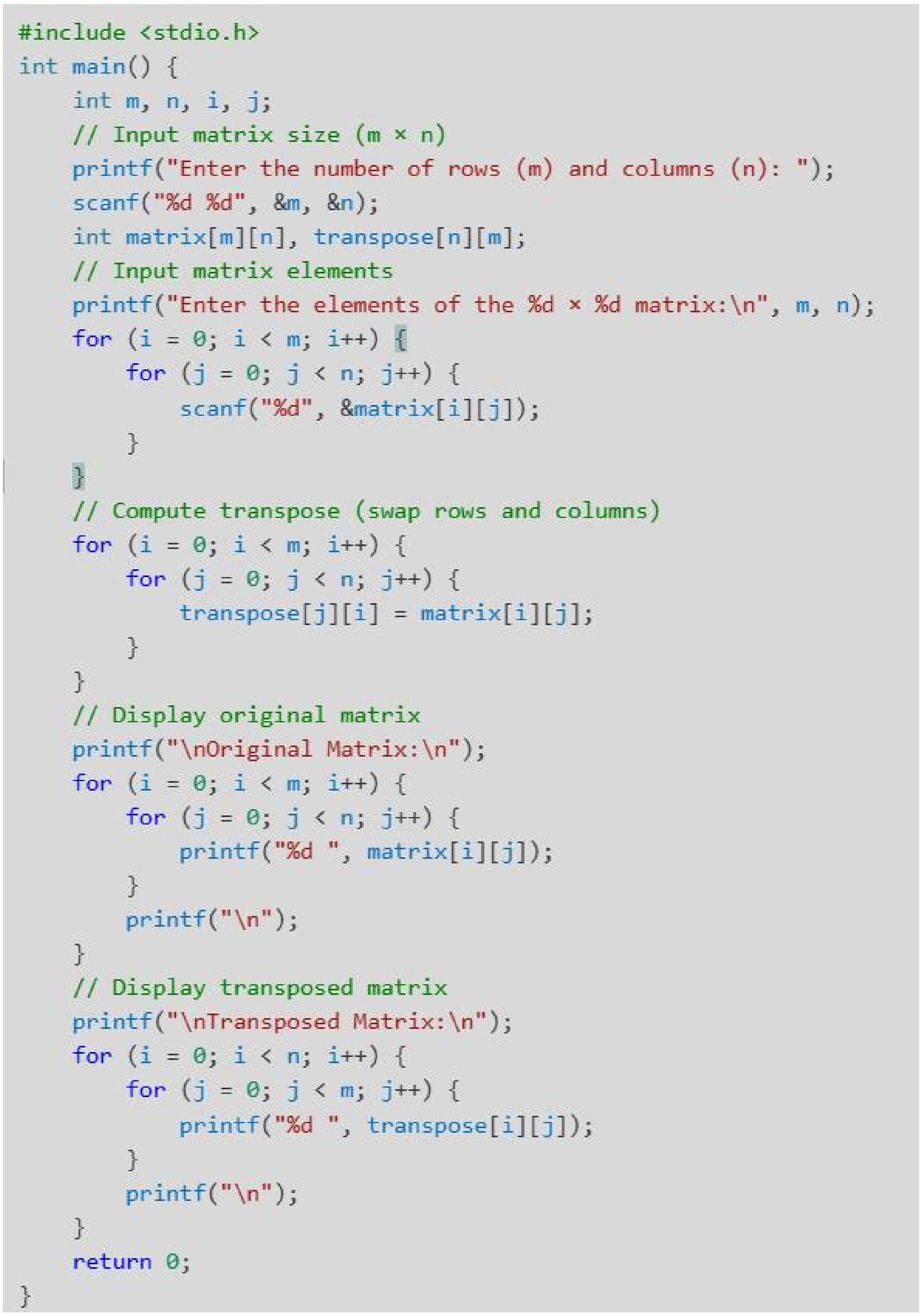
| METHOD | EXAMPLE |
|---|---|
| Full Initialization | `int arr[2][2] = { {1, 2}, {3, 4} };` |
| Partial Initialization | `int arr[3][3] = { {1, 2}, {3}, {4, 5, 6} };` |
| Without Braces | `int arr[2][3] = {1, 2, 3, 4, 5, 6};` |
| Automatic Size Deduction | `int arr[][3] = { {1, 2, 3}, {4, 5, 6} };` |
| Run-Time Initialization | User input with `scanf` |

19. Write a C program to check whether a given square matrix is an identity matrix. (All diagonal elements must be 1 and all non-diagonal elements must be 0.)



```c
#include <stdio.h>

int main() {
    int n, i, j, isIdentity = 1;

    // Input matrix size (square matrix)
    printf("Enter the size of the square matrix: ");
    scanf("%d", &n);
    int matrix[n][n];
    // Input matrix elements
    printf("Enter the elements of the %d × %d matrix:\n", n, n);
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    // Check if the matrix is an identity matrix
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if ((i == j && matrix[i][j] != 1) || (i != j && matrix[i][j] != 0)) {
                isIdentity = 0; // Not an identity matrix
                break;
            }
        }
        if (!isIdentity) break; // Stop checking if it's already determined
    }

    // Display result
    if (isIdentity) {
        printf("The given matrix is an Identity Matrix.\n");
    } else {
        printf("The given matrix is NOT an Identity Matrix.\n");
    }

    return 0;
}
```

20. Write a C program to read a 2D integer matrix of size m × n and compute its transpose.

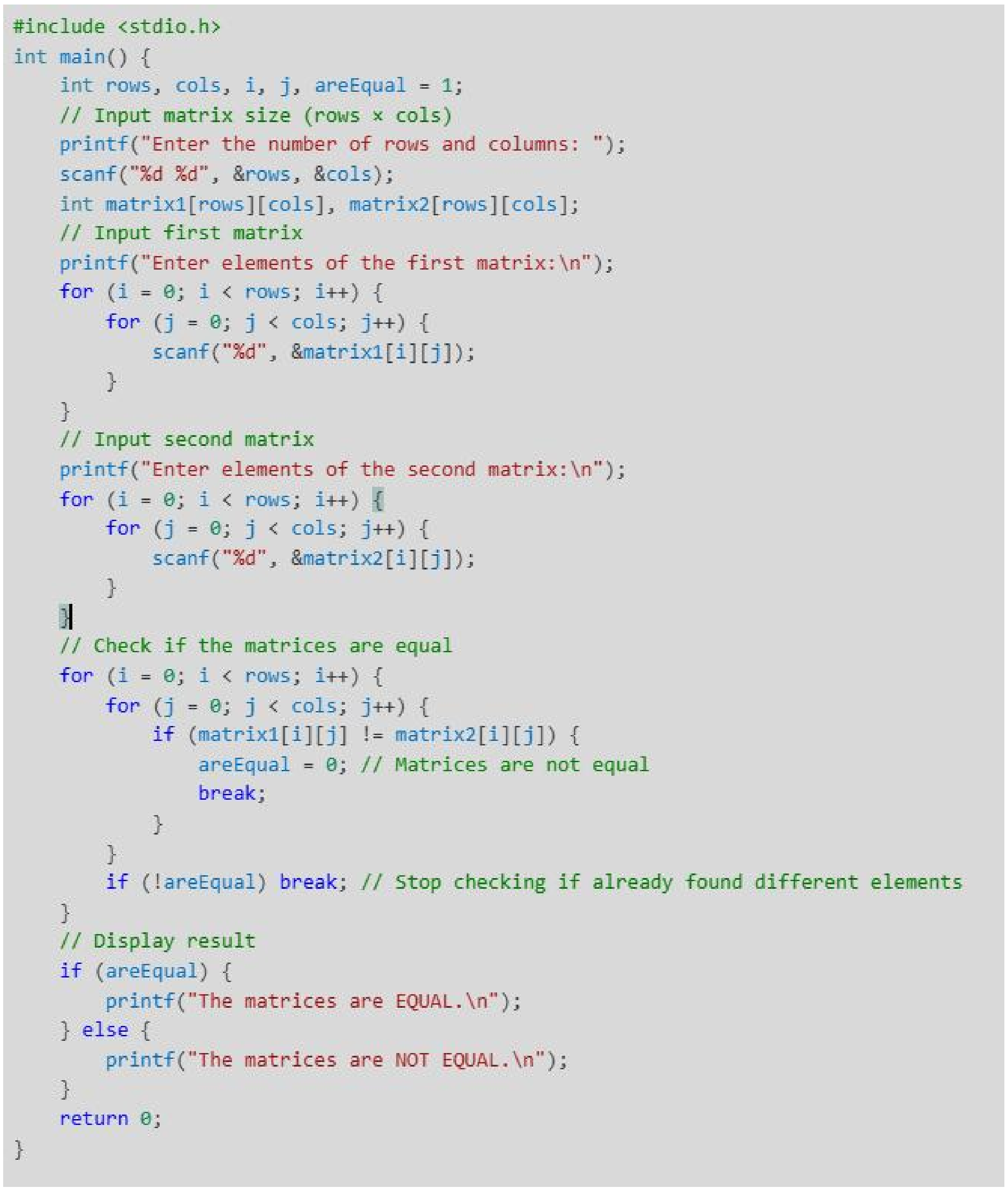


```c
#include <stdio.h>
int main() {
    int m, n, i, j;
    // Input matrix size (m × n)
    printf("Enter the number of rows (m) and columns (n): ");
    scanf("%d %d", &m, &n);
    int matrix[m][n], transpose[n][m];
    // Input matrix elements
    printf("Enter the elements of the %d × %d matrix:\n", m, n);
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    // Compute transpose (swap rows and columns)
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }
    // Display original matrix
    printf("\nOriginal Matrix:\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
    // Display transposed matrix
    printf("\nTransposed Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

21. Write a C program to read two 2D matrices, perform matrix addition, and print the sum matrix.

```c
#include <stdio.h>
int main() {
    int rows, cols, i, j;
    // Input number of rows and columns
    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &rows, &cols);
    int matrix1[rows][cols], matrix2[rows][cols], sumMatrix[rows][cols];
    // Input first matrix
    printf("Enter elements of the first matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }
    // Input second matrix
    printf("Enter elements of the second matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }
    // Perform matrix addition
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            sumMatrix[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }
    // Display the sum matrix
    printf("\nSum Matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("%d ", sumMatrix[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

22. Write a C program to check whether two matrices are equal.

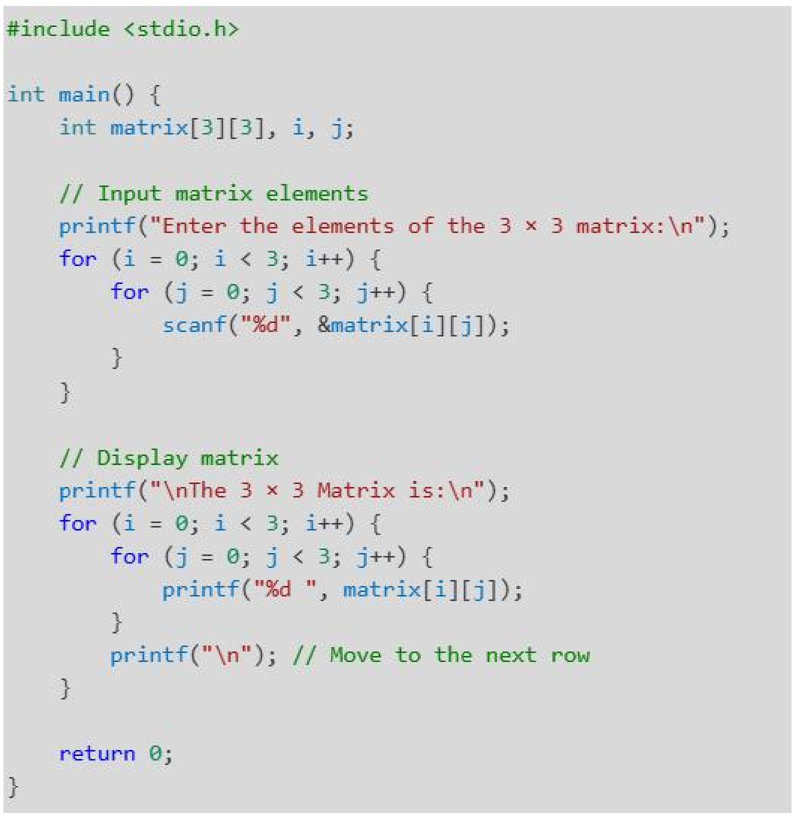


```c
#include <stdio.h>
int main() {
    int rows, cols, i, j, areEqual = 1;
    // Input matrix size (rows × cols)
    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &rows, &cols);
    int matrix1[rows][cols], matrix2[rows][cols];
    // Input first matrix
    printf("Enter elements of the first matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }
    // Input second matrix
    printf("Enter elements of the second matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }
    // Check if the matrices are equal
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            if (matrix1[i][j] != matrix2[i][j]) {
                areEqual = 0; // Matrices are not equal
                break;
            }
        }
        if (!areEqual) break; // Stop checking if already found different elements
    }
    // Display result
    if (areEqual) {
        printf("The matrices are EQUAL.\n");
    } else {
        printf("The matrices are NOT EQUAL.\n");
    }
    return 0;
}
```

23. Write a C statement to access the element in the second row and third column of a matrix.

```
#include <stdio.h>

int main() {

    int matrix[3][3] = {

        {10, 20, 30},

        {40, 50, 60},

        {70, 80, 90}

    };

    // Accessing the element in the second row and third column

    printf("Element at second row, third column: %d\n", matrix[1][2]);

    return 0;

}
```

24. Write a C program to read and display a 3x3 matrix.



```
#include <stdio.h>

int main() {
    int matrix[3][3], i, j;

    // Input matrix elements
    printf("Enter the elements of the 3 × 3 matrix:\n");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    // Display matrix
    printf("\nThe 3 × 3 Matrix is:\n");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n"); // Move to the next row
    }

    return 0;
}
```

25 a)Write a C program that uses an enum to represent the days of the week

```
#include <stdio.h>

// Define an enum for days of the week
enum Days {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY};

int main() {
    // Declare a variable of type enum Days
    enum Days today;

    // Assign a value (Example: Wednesday)
    today = WEDNESDAY;

    // Print the day
    printf("The value of today (Wednesday) in enum is: %d\n", today);

    return 0;
}
```

b)Define an enum for the four seasons: Spring, Summer, Autumn, and Winter. Write a C program that initializes a season and prints a corresponding message using a switch statement.

```
#include <stdio.h>

// Define an enum for the four seasons
enum Season {SPRING, SUMMER, AUTUMN, WINTER};

int main() {
    // Declare and initialize a season
    enum Season currentSeason = AUTUMN;

    // Print a message based on the season using switch
    switch (currentSeason) {
        case SPRING:
            printf("It's Spring! Flowers are blooming.\n");
            break;
        case SUMMER:
            printf("It's Summer! The weather is hot and sunny.\n");
            break;
        case AUTUMN:
            printf("It's Autumn! Leaves are falling.\n");
            break;
        case WINTER:
            printf("It's Winter! Time for snow and warm clothes.\n");
            break;
        default:
            printf("Invalid season!\n");
    }

    return 0;
}
```

26. What are enumerated data types? Explain how ordinal values are assigned to their members with examples. Compare enum with typedef.

## Enumerated Data Types

An **enumerated data type (enum)** is a user-defined data type in C and other programming languages that consists of a set of named integral constants. Enums improve code readability and make it easier to manage sets of related constants.

## Syntax of enum in C:

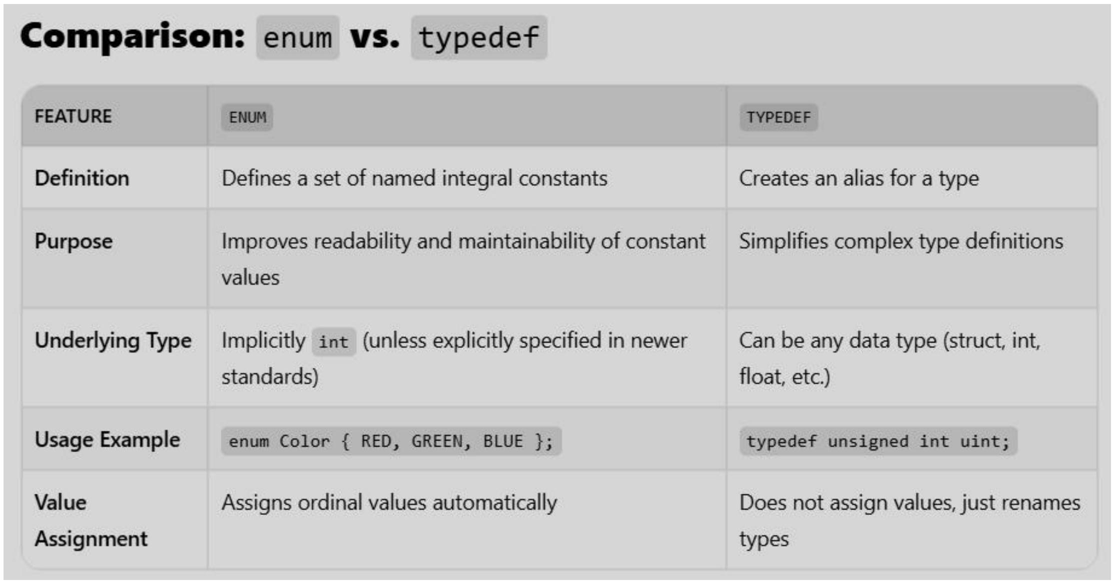
```
enum Color { RED, GREEN, BLUE };
```

## Ordinal Value Assignment in Enums

By default, the first enumerator is assigned the value 0, the second 1, the third 2, and so on. However, you can manually assign specific values.

## Example:

```
#include <stdio.h>

enum Days { SUNDAY = 1, MONDAY, TUESDAY, WEDNESDAY,
THURSDAY, FRIDAY, SATURDAY };

int main() {
    printf("Monday = %d\n", MONDAY);
    printf("Wednesday = %d\n", WEDNESDAY);
    printf("Saturday = %d\n", SATURDAY);
    return 0;
}
```



**Comparison: `enum` vs. `typedef`**

| FEATURE | ENUM | TYPEDEF |
|---|---|---|
| **Definition** | Defines a set of named integral constants | Creates an alias for a type |
| **Purpose** | Improves readability and maintainability of constant values | Simplifies complex type definitions |
| **Underlying Type** | Implicitly `int` (unless explicitly specified in newer standards) | Can be any data type (struct, int, float, etc.) |
| **Usage Example** | `enum Color { RED, GREEN, BLUE };` | `typedef unsigned int uint;` |
| **Value Assignment** | Assigns ordinal values automatically | Does not assign values, just renames types |

27. Explain typedef and enum data types in detail with examples.

**1. typedef (Type Definition)**

**Definition:**

- typedef is a keyword used to create a new name (alias) for an existing data type.
- It improves code readability, especially when dealing with complex data types such as pointers and structures.

**Syntax:**

```
typedef existing_datatype new_name;
```

**Example 1: Using typedef with Basic Data Types**

```
#include <stdio.h>
// Creating an alias for unsigned inttypedef unsigned int uint;
int main() {
    uint age = 25;  // Now, uint is another name for unsigned int
    printf("Age: %u\n", age);
    return 0;
}
```

**Example 2: Using typedef with Structures**

```
#include <stdio.h>
// Defining a structuretypedef struct {
    char name[50];
    int age;
} Person;
int main() {
    Person p1 = {"John Doe", 30};
    printf("Name: %s, Age: %d\n", p1.name, p1.age);
    return 0;
}
```

**Benefits of** typedef**:**

- Reduces code complexity.
- Makes the code more readable.
- Useful for platform-independent programming.

## 2. enum (Enumerated Data Type)

**Definition:**

- enum (enumeration) is a user-defined data type that consists of integral constants.
- It assigns names to integer values, making the code more readable.

**Syntax:**

```
enum enum_name { constant1, constant2, constant3, ... };
```

**Example 1: Basic enum Usage**

```
#include <stdio.h>
// Defining an enumerationenum Color { RED, GREEN, BLUE };
int main() {
    enum Color myColor = GREEN;  // Assigning an enum value
    printf("The value of GREEN is: %d\n", myColor);
    return 0;
}
```

**Explanation:**

- By default, RED = 0, GREEN = 1, BLUE = 2.

**Example 2: Assigning Custom Values**

```
#include <stdio.h>
enum Status { SUCCESS = 1, FAILURE = -1, PENDING = 5 };
int main() {
    enum Status taskStatus = PENDING;
    printf("Task Status: %d\n", taskStatus);
    return 0;
}
```

**Key Points:**

- The first constant defaults to 0 unless assigned a value.
- The next constants increment from the previous value unless explicitly defined.
- Enumerators can have non-sequential values.

28. Write a simple C program to declare and use an enum.

```c
#include <stdio.h>

// Define an enum for traffic light signals
enum TrafficLight {RED, YELLOW, GREEN};

int main() {
    // Declare an enum variable
    enum TrafficLight signal;

    // Assign a value
    signal = GREEN;

    // Display the signal status
    printf("Current Traffic Light Signal: ");

    // Use a switch statement to print corresponding messages
    switch (signal) {
        case RED:
            printf("STOP! The light is RED.\n");
            break;
        case YELLOW:
            printf("WAIT! The light is YELLOW.\n");
            break;
        case GREEN:
            printf("GO! The light is GREEN.\n");
            break;
        default:
            printf("Invalid Signal!\n");
    }

    return 0;
}
```

## 29. Explain the purpose of the typedef construct in C.

The typedef keyword in C is used to create a new name (alias) for an existing data type, making code more readable and easier to manage.

**Key Purposes:**

1. **Improves Code Readability** – Simplifies complex data types, making the code more understandable.
2. **Enhances Code Portability** – Allows easy modification of data types without changing multiple lines of code.
3. **Reduces Complexity in Declarations** – Useful for defining shorter names for structs, pointers, and arrays.

## 30. Explain enum data type in C with an example.

The enum (enumeration) data type in C is a user-defined type that assigns names to integral constants, improving code readability and maintainability.

**Key Features:**

1. **Defines Named Integer Constants** – Each enumerator is assigned an integer value starting from 0 by default.
2. **Enhances Code Readability** – Replaces hardcoded numbers with meaningful names.
3. **Allows Custom Value Assignment** – Users can specify values explicitly.

**Example:**

```
#include <stdio.h>
enum Days { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
FRIDAY, SATURDAY };
int main() {
   enum Days today = WEDNESDAY;
   printf("Today is day number: %d\n", today);  // Output: 3
   return 0;
}
```

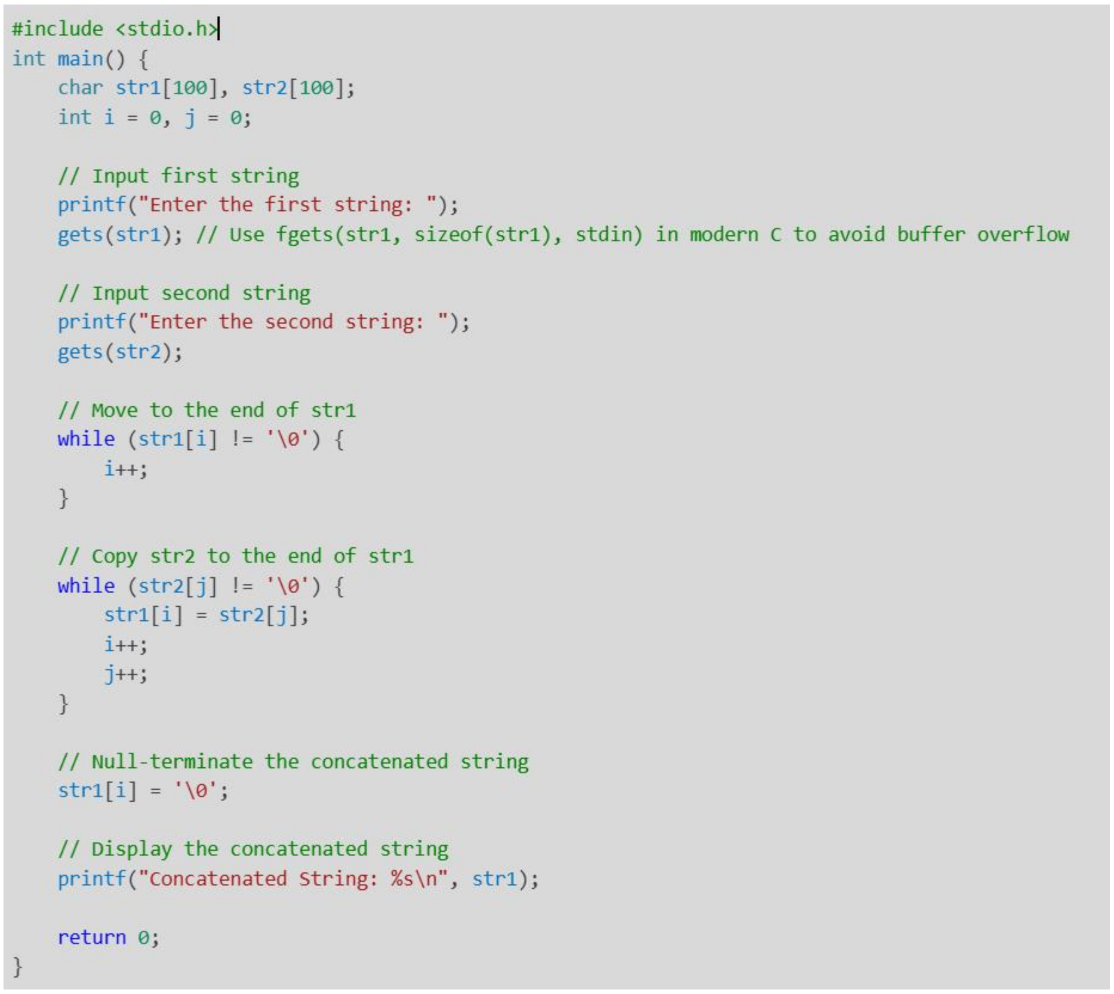
31 What are string handling functions? Write a C program to determine the length of a given string using both a built-in string function and a manual method without using string functions.

**String handling functions** in C are built-in functions provided by the <string.h> library to perform operations on strings, such as:

- strlen() - Get the length of a string.
- strcpy() - Copy one string into another.
- strcat() - Concatenate (join) two strings.
- strcmp() - Compare two strings.
- strrev() - Reverse a string (not available in standard C, but present in some compilers).
- strchr() - Find the first occurrence of a character in a string.
- strstr() - Find a substring in a string.

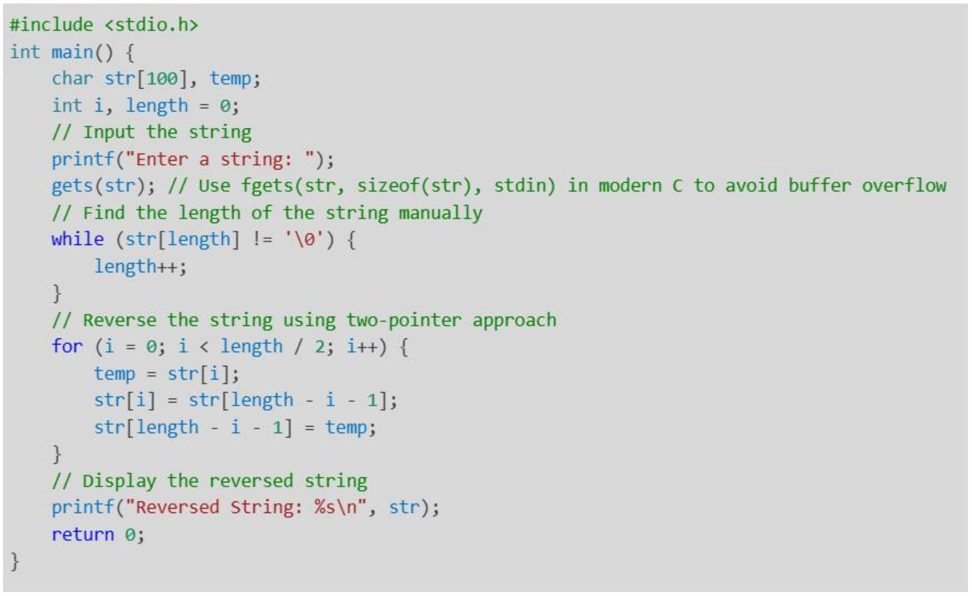
```
#include <stdio.h>
#include <string.h> // Required for strlen()

int main() {
    char str[100];
    int i, length = 0;

    // Input a string
    printf("Enter a string: ");
    gets(str); // Reads input (use fgets in modern C to avoid buffer overflow)

    // Using built-in strlen() function
    printf("Length using built-in strlen(): %lu\n", strlen(str));

    // Manual method (without using string functions)
    for (i = 0; str[i] != '\0'; i++) {
        length++;
    }
    printf("Length using manual method: %d\n", length);

    return 0;
}
```

## 32. Write a C program to concatenate two strings without using built in string functions.



```c
#include <stdio.h>
int main() {
    char str1[100], str2[100];
    int i = 0, j = 0;

    // Input first string
    printf("Enter the first string: ");
    gets(str1); // Use fgets(str1, sizeof(str1), stdin) in modern C to avoid buffer overflow

    // Input second string
    printf("Enter the second string: ");
    gets(str2);

    // Move to the end of str1
    while (str1[i] != '\0') {
        i++;
    }

    // Copy str2 to the end of str1
    while (str2[j] != '\0') {
        str1[i] = str2[j];
        i++;
        j++;
    }

    // Null-terminate the concatenated string
    str1[i] = '\0';

    // Display the concatenated string
    printf("Concatenated String: %s\n", str1);

    return 0;
}
```

## 33. Write a C program to reverse a string without using string handling functions.

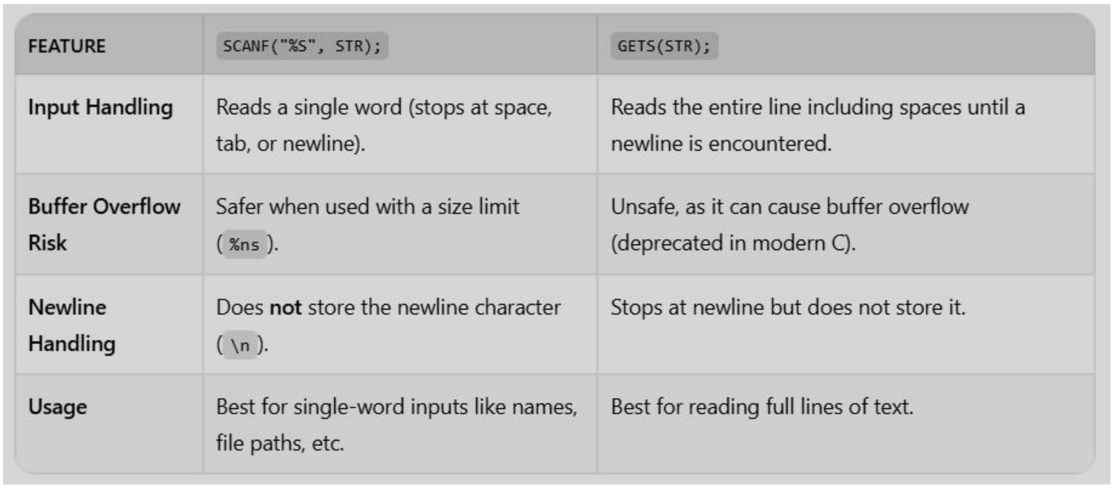


```c
#include <stdio.h>
int main() {
    char str[100], temp;
    int i, length = 0;
    // Input the string
    printf("Enter a string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C to avoid buffer overflow
    // Find the length of the string manually
    while (str[length] != '\0') {
        length++;
    }
    // Reverse the string using two-pointer approach
    for (i = 0; i < length / 2; i++) {
        temp = str[i];
        str[i] = str[length - i - 1];
        str[length - i - 1] = temp;
    }
    // Display the reversed string
    printf("Reversed String: %s\n", str);
    return 0;
}
```

## 34. Write a C program to print the number of vowels and consonants in a string.

```c
#include <stdio.h>

int main() {
    char str[100];
    int i, vowels = 0, consonants = 0;

    // Input the string
    printf("Enter a string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C to avoid buffer overflow

    // Traverse the string character by character
    for (i = 0; str[i] != '\0'; i++) {
        // Check if the character is a letter
        if ((str[i] >= 'A' && str[i] <= 'Z') || (str[i] >= 'a' && str[i] <= 'z')) {
            // Check if it is a vowel
            if (str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O' || str[i] == 'U' ||
                str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u') {
                vowels++;
            } else {
                consonants++;
            }
        }
    }

    // Display results
    printf("Number of vowels: %d\n", vowels);
    printf("Number of consonants: %d\n", consonants);

    return 0;
}
```

## 35. What is the difference between scanf("%s", str); and gets(str);?

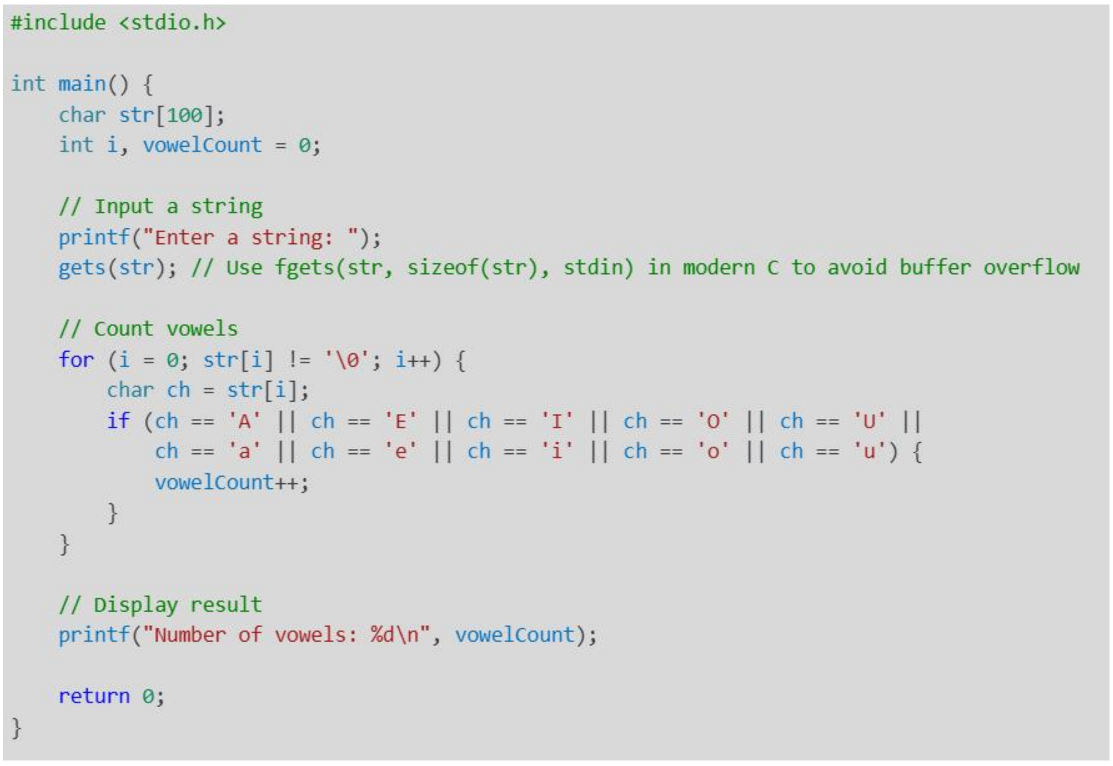


| FEATURE | `SCANF("%S", STR);` | `GETS(STR);` |
|---|---|---|
| **Input Handling** | Reads a single word (stops at space, tab, or newline). | Reads the entire line including spaces until a newline is encountered. |
| **Buffer Overflow Risk** | Safer when used with a size limit ( `%ns` ). | Unsafe, as it can cause buffer overflow (deprecated in modern C). |
| **Newline Handling** | Does **not** store the newline character ( `\n` ). | Stops at newline but does not store it. |
| **Usage** | Best for single-word inputs like names, file paths, etc. | Best for reading full lines of text. |

36. Write a C program to read your name from the console and display it.

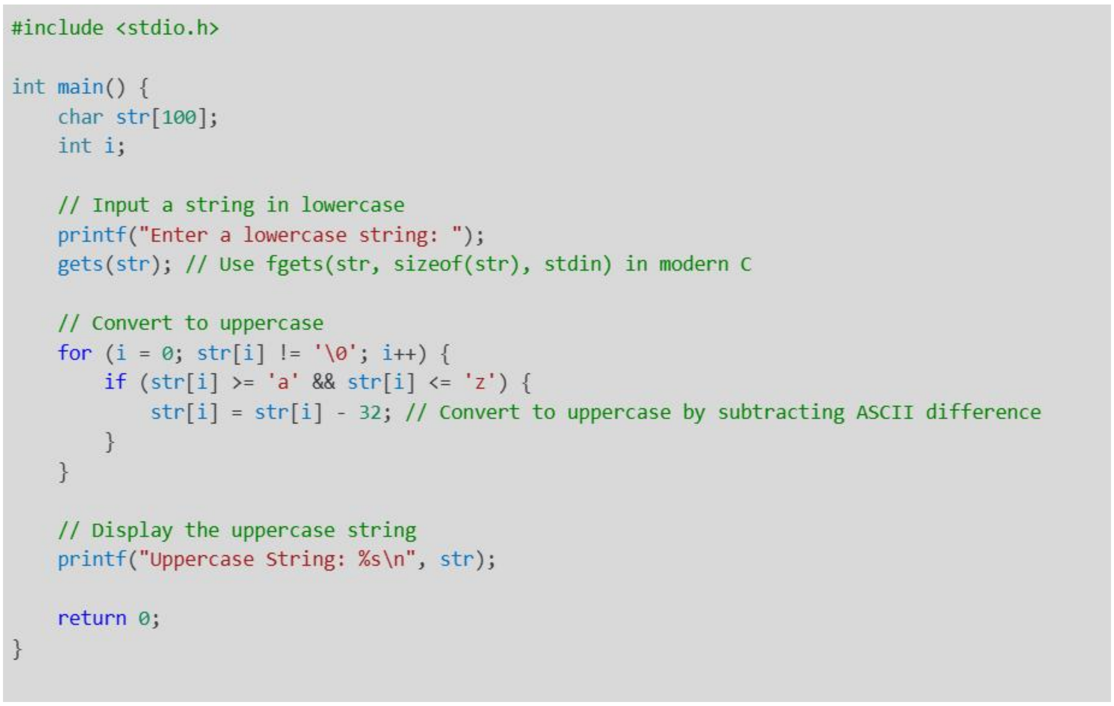
```
#include <stdio.h>

int main() {
    char name[100];

    // Prompt the user to enter their name
    printf("Enter your name: ");

    // Read the name (including spaces) using fgets
    fgets(name, sizeof(name), stdin);

    // Display the name
    printf("Your name is: %s", name);

    return 0;
}
```

37. a)Write a C program to count number of vowels in a given string.(5 Marks)



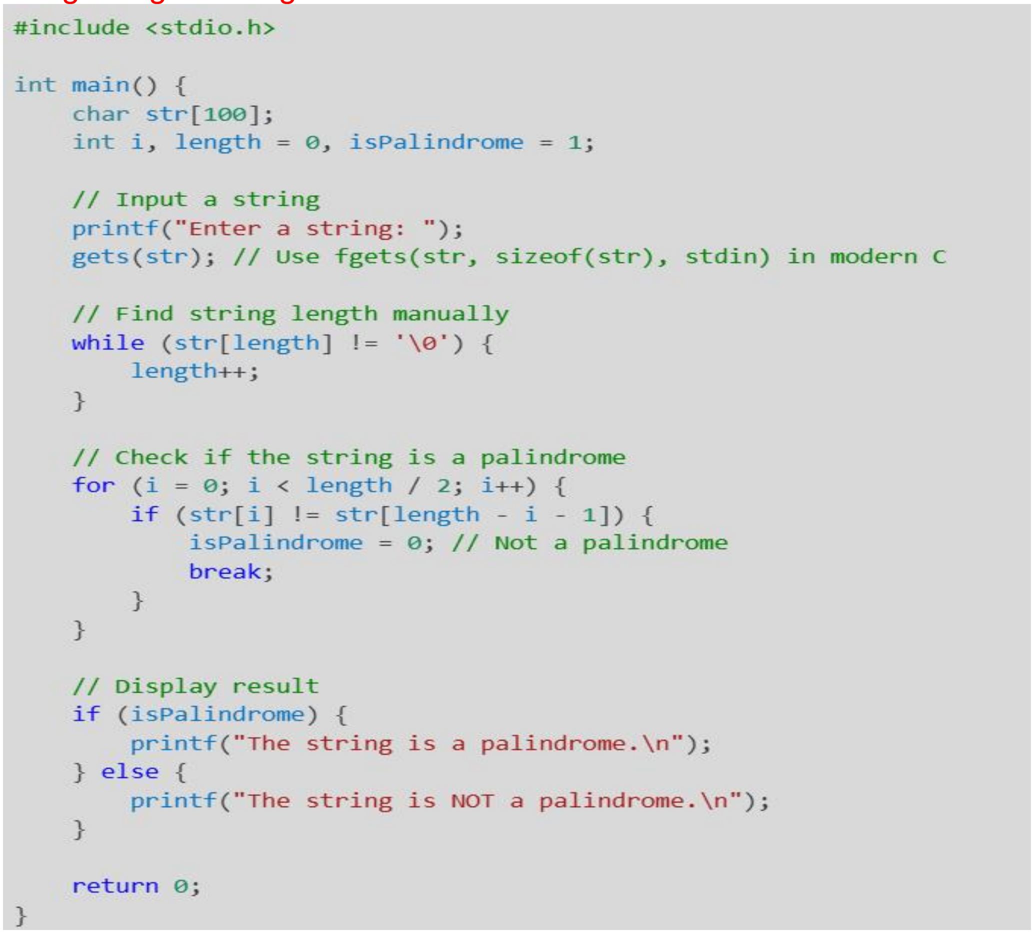
```
#include <stdio.h>

int main() {
    char str[100];
    int i, vowelCount = 0;

    // Input a string
    printf("Enter a string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C to avoid buffer overflow

    // Count vowels
    for (i = 0; str[i] != '\0'; i++) {
        char ch = str[i];
        if (ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ||
            ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
            vowelCount++;
        }
    }

    // Display result
    printf("Number of vowels: %d\n", vowelCount);

    return 0;
}
```

b)Write a C program to read a string in lowercase and convert it into upper case.(4 Marks)



```c
#include <stdio.h>

int main() {
    char str[100];
    int i;

    // Input a string in lowercase
    printf("Enter a lowercase string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C

    // Convert to uppercase
    for (i = 0; str[i] != '\0'; i++) {
        if (str[i] >= 'a' && str[i] <= 'z') {
            str[i] = str[i] - 32; // Convert to uppercase by subtracting ASCII difference
        }
    }

    // Display the uppercase string
    printf("Uppercase String: %s\n", str);

    return 0;
}
```

38. Write a C program to check whether a string is palindrome or not without using string handling functions.



```c
#include <stdio.h>

int main() {
    char str[100];
    int i, length = 0, isPalindrome = 1;

    // Input a string
    printf("Enter a string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C

    // Find string length manually
    while (str[length] != '\0') {
        length++;
    }

    // Check if the string is a palindrome
    for (i = 0; i < length / 2; i++) {
        if (str[i] != str[length - i - 1]) {
            isPalindrome = 0; // Not a palindrome
            break;
        }
    }

    // Display result
    if (isPalindrome) {
        printf("The string is a palindrome.\n");
    } else {
        printf("The string is NOT a palindrome.\n");
    }

    return 0;
}
```

39. List any 4 string handling functions in C programming and explain with examples.

**1. strlen() – String Length**

- Returns the number of characters in a string (excluding the null character \0).

**Example:**

```
#include <stdio.h>#include <string.h>
int main() {
    char str[] = "Hello, C!";
    printf("Length of string: %lu\n", strlen(str));
    return 0;
}
```

**Output:**

```
Length of string: 9
```

**2. strcpy() – String Copy**

- Copies the content of one string into another.

**Example:**

```
#include <stdio.h>#include <string.h>
int main() {
    char source[] = "C Programming";
    char destination[20];

    strcpy(destination, source);
    printf("Copied String: %s\n", destination);
    return 0;
}
```

**Output:**

```
Copied String: C Programming
```

**3. strcat() – String Concatenation**

- Appends (concatenates) one string to another.

**Example:**

```
#include <stdio.h>#include <string.h>
int main() {
    char str1[30] = "Hello, ";
    char str2[] = "World!";

    strcat(str1, str2);
    printf("Concatenated String: %s\n", str1);
    return 0;
}
```

**Output:**

```
Concatenated String: Hello, World!
```

**4. strcmp() – String Comparison**

- Compares two strings lexicographically (alphabetical order).
- Returns:
    - 0 if both strings are equal
    - A positive value if the first string is greater
    - A negative value if the second string is greater

**Example:**

```
#include <stdio.h>#include <string.h>
int main() {
    char str1[] = "Apple";
    char str2[] = "Banana";

    int result = strcmp(str1, str2);
    if (result == 0)
        printf("Strings are equal.\n");
    else if (result > 0)
        printf("String 1 is greater.\n");
    else
        printf("String 2 is greater.\n");

    return 0;
}
```

**Output:**

```
String 2 is greater.
```

40. Write a C program to find the length of a string using a string handling function.

```c
#include <stdio.h>
#include <string.h> // Required for strlen()

int main() {
    char str[100];

    // Input a string
    printf("Enter a string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C to avoid buffer overflow

    // Find the length using strlen()
    int length = strlen(str);

    // Display the length
    printf("The length of the string is: %d\n", length);

    return 0;
}
```

41. Explain any three string handling functions in C with examples and their syntax.

Repeat

Syntax

strlen()- size_t strlen(const char *str);
strcpy()- char *strcpy(char *destination, const char *source);
strcmp()- int strcmp(const char *str1, const char *str2);

42. Explain any three string handling functions in C with examples.

Repeat

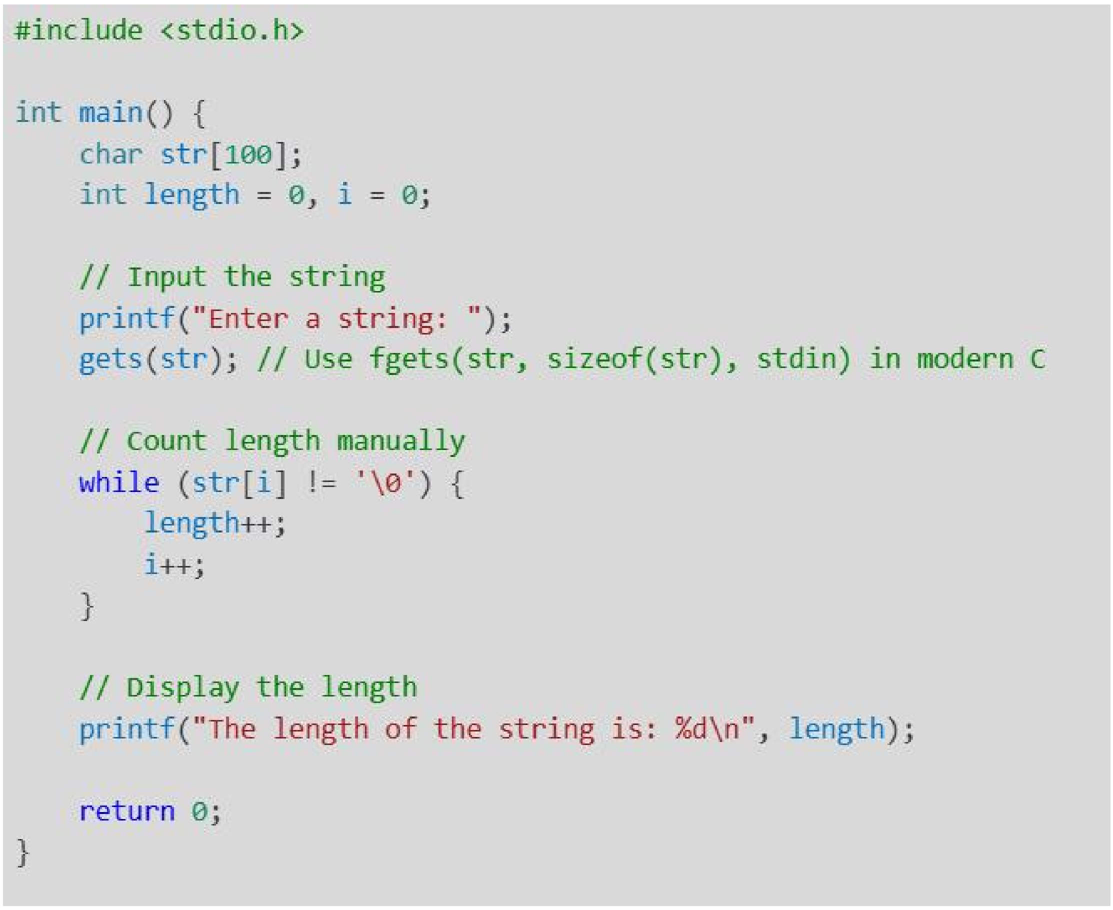
43 a)Write a C program to count the occurrence of a given character in a string.(5 Marks)

```
#include <stdio.h>

int main() {
    char str[100], ch;
    int i, count = 0;

    // Input the string
    printf("Enter a string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C

    // Input the character to count
    printf("Enter the character to count: ");
    scanf("%c", &ch);

    // Count occurrences of the given character
    for (i = 0; str[i] != '\0'; i++) {
        if (str[i] == ch) {
            count++;
        }
    }

    // Display the result
    printf("The character '%c' appears %d times in the string.\n", ch, count);

    return 0;
}
```

b)Write a program to count the number of spaces in a given string?(4 Marks)

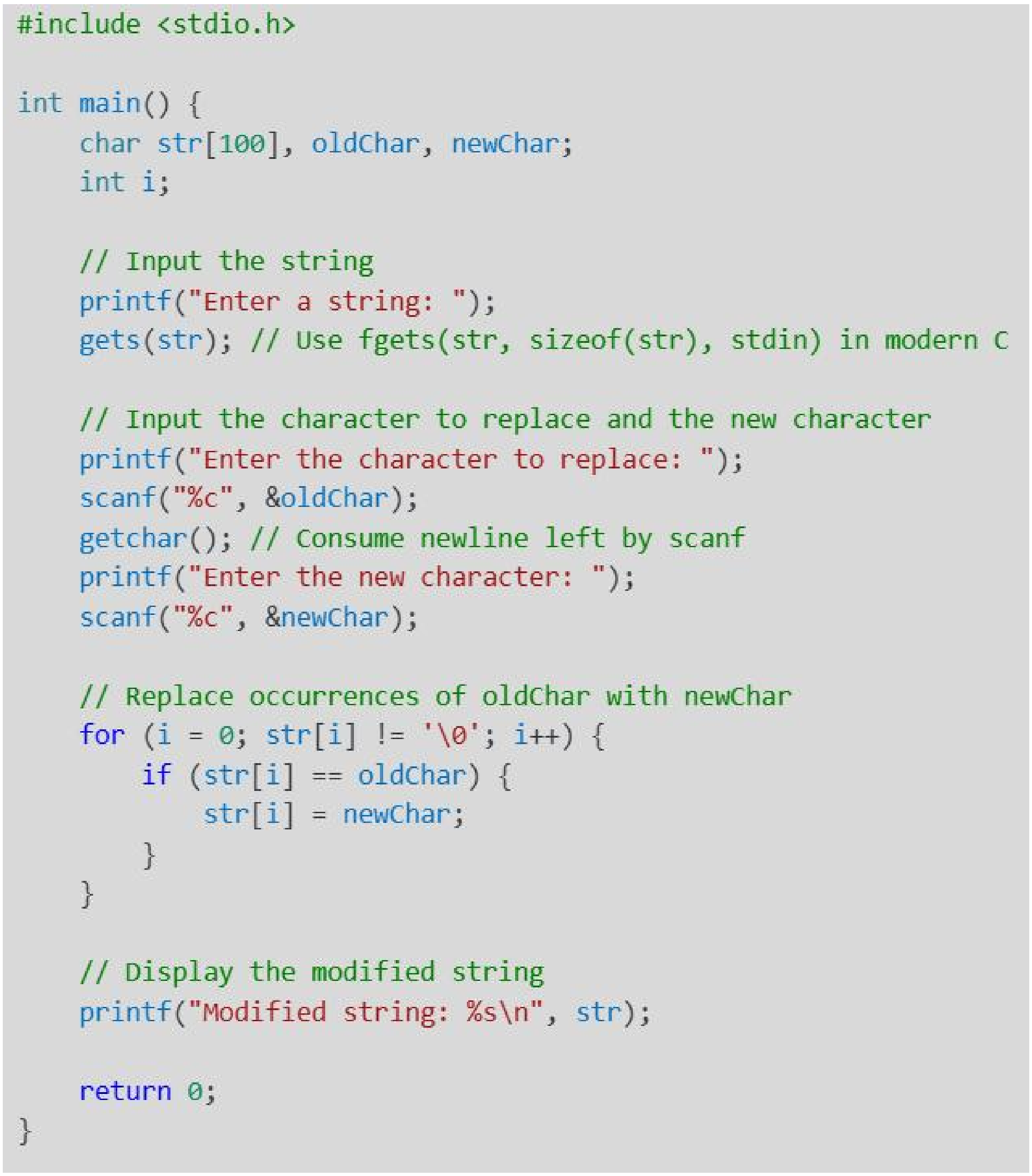
```
#include <stdio.h>

int main() {
    char str[100];
    int i, spaceCount = 0;

    // Input the string
    printf("Enter a string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C

    // Count spaces in the string
    for (i = 0; str[i] != '\0'; i++) {
        if (str[i] == ' ') {
            spaceCount++;
        }
    }

    // Display the result
    printf("The number of spaces in the string: %d\n", spaceCount);

    return 0;
}
```

44 a)Write a C program to find length of a string without using string handling functions.(5 Marks)



```c
#include <stdio.h>

int main() {
    char str[100];
    int length = 0, i = 0;

    // Input the string
    printf("Enter a string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C

    // Count length manually
    while (str[i] != '\0') {
        length++;
        i++;
    }

    // Display the length
    printf("The length of the string is: %d\n", length);

    return 0;
}
```

b)Write a C program to replace all occurrences of a given character in a string with another character.(4 Marks)



```c
#include <stdio.h>

int main() {
    char str[100], oldChar, newChar;
    int i;

    // Input the string
    printf("Enter a string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C

    // Input the character to replace and the new character
    printf("Enter the character to replace: ");
    scanf("%c", &oldChar);
    getchar(); // Consume newline left by scanf
    printf("Enter the new character: ");
    scanf("%c", &newChar);

    // Replace occurrences of oldChar with newChar
    for (i = 0; str[i] != '\0'; i++) {
        if (str[i] == oldChar) {
            str[i] = newChar;
        }
    }

    // Display the modified string
    printf("Modified string: %s\n", str);

    return 0;
}
```

45. a)Write a C program to compare two strings using the string handling function.(5 Marks)

```c
#include <stdio.h>
#include <string.h> // Required for strcmp()

int main() {
   char str1[100], str2[100];

   // Input two strings
   printf("Enter the first string: ");
   gets(str1); // Use fgets(str1, sizeof(str1), stdin) in modern C
```

```
    printf("Enter the second string: ");
    gets(str2); // Use fgets(str2, sizeof(str2), stdin) in modern C

    // Compare the two strings using strcmp()
    int result = strcmp(str1, str2);

    // Display the result
    if (result == 0) {
        printf("The strings are equal.\n");
    } else {
        printf("The strings are NOT equal.\n");
    }

    return 0;
}
```

b)Write a C program to read a string in uppercase and convert it into lower case.(4 Marks)

```
#include <stdio.h>
#include <string.h> // Required for string functions
#include <ctype.h>  // Required for tolower()

int main() {
    char str[100];
    int i;

    // Input the string
    printf("Enter an uppercase string: ");
    gets(str); // Use fgets(str, sizeof(str), stdin) in modern C

    // Convert each character to lowercase
    for (i = 0; str[i] != '\0'; i++) {
        str[i] = tolower(str[i]);
    }

    // Display the converted string
    printf("The lowercase string is: %s\n", str);

    return 0;
}
```

46. Write a C program to copy content of one string into another using the string handling function.

```
#include <stdio.h>
#include <string.h> // Required for strcpy()

int main() {
    char source[100], destination[100];
    // Input the source string
    printf("Enter the source string: ");
    gets(source); // Use fgets(source, sizeof(source), stdin) in modern C

    // Copy content of source string to destination using strcpy()
    strcpy(destination, source);

    // Display the copied string
    printf("The copied string is: %s\n", destination);

    return 0;
}
```

47. Write a C program to concatenate two strings using the string handling function.

```
#include <stdio.h>
#include <string.h> // Required for strcat()

int main() {
    char str1[100], str2[100];

    // Input the first string
    printf("Enter the first string: ");
    gets(str1); // Use fgets(str1, sizeof(str1), stdin) in modern C

    // Input the second string
    printf("Enter the second string: ");
    gets(str2); // Use fgets(str2, sizeof(str2), stdin) in modern C

    // Concatenate str2 to str1 using strcat()
    strcat(str1, str2);

    // Display the concatenated string
    printf("The concatenated string is: %s\n", str1);

    return 0;
}
```

48. Write a C program to compare two strings using the string handling function.

```
#include <stdio.h>
#include <string.h> // Required for strcmp()

int main() {
   char str1[100], str2[100];

   // Input the first string
   printf("Enter the first string: ");
   gets(str1); // Use fgets(str1, sizeof(str1), stdin) in modern C

   // Input the second string
   printf("Enter the second string: ");
   gets(str2); // Use fgets(str2, sizeof(str2), stdin) in modern C

   // Compare the two strings using strcmp()
   int result = strcmp(str1, str2);

   // Display the comparison result
   if (result == 0) {
      printf("The strings are equal.\n");
   } else {
      printf("The strings are NOT equal.\n");
   }

   return 0;
}
```