Bangladesh University of Business & Technology (BUBT)

Lab Report

Course Title : Structured Programming Language Lab
Course Code : CSE 102

Submitted by

Name : Makhzum-Bin-Harun
ID : 20254103279
Intake : 56
Section : 07
Program: B.Sc. Engg. in CSE

Submitted to

Name : Sourav Kundu
Designation: Lecturer
Department of Computer Science & Engineering
Bangladesh University of Business & Technology.

Date of Submission: November 26, 2025

Signature of Teacher

TABLE OF CONTENTS

Problem 07. Write a C program to print all negative elements in an array.

07.1. Algorithm: Algorithm to Factorial using For Loop

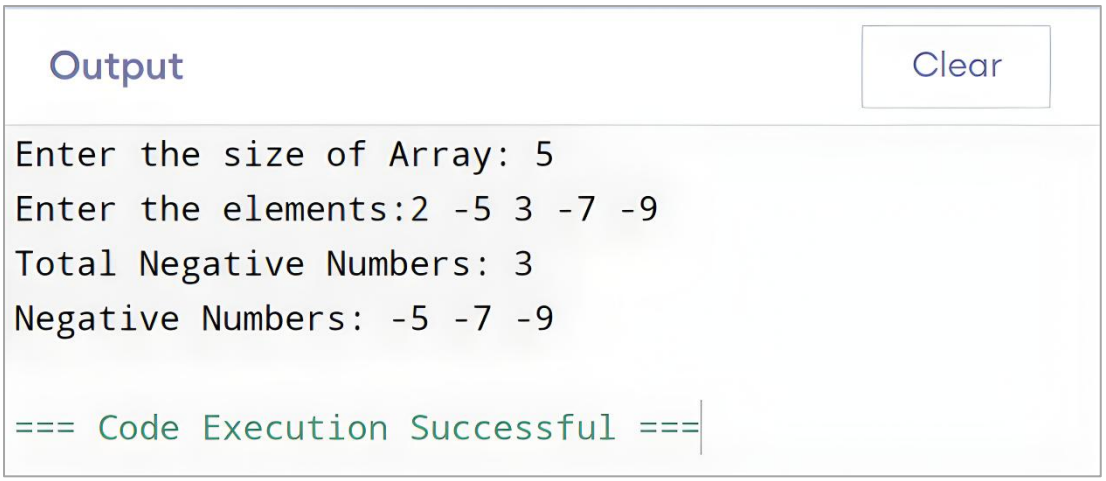
1. Start.
2. Input the size of the array (N).
3. Read N elements into the array.
4. Traverse the array and check each element.
5. If an element is less than 0, count it and print it.
6. Display the total number of negative elements.
7. End.

07.2. C Program Code:

```
#include <stdio.h>

int main(){
    int N, neg = 0;

    printf("Enter the size of the array: ");
    scanf("%d", &N);

    int arr[N];
    printf("Enter the elements: ");
    for(int i = 0; i < N; i++){
        scanf("%d", &arr[i]);
    }

    for(int i = 0; i < N; i++){
        if(arr[i] < 0) {
            neg++;
        }
    }

    printf("Total negative numbers: %d\n", neg);

    if(neg > 0){
        printf("Negative numbers: ");
        for(int i = 0; i < N; i++){
            if(arr[i] < 0){
                printf("%d ", arr[i]);
            }
        }
        printf("\n");
    }
```

```
      else{
         printf("No negative numbers found.\n");
    }

    return 0;
}
```

07.3. Output:



07.4. Discussion:

In this program, we read an array from the user and then check each element to find which values are negative. Negative elements are counted and printed separately. This program helped us practice array traversal, conditional checking, and basic input/output operations in C.

Problem 08. Write a C program to find second-largest number in an array.

08.1. Algorithm: Algorithm to Factorial using For Loop

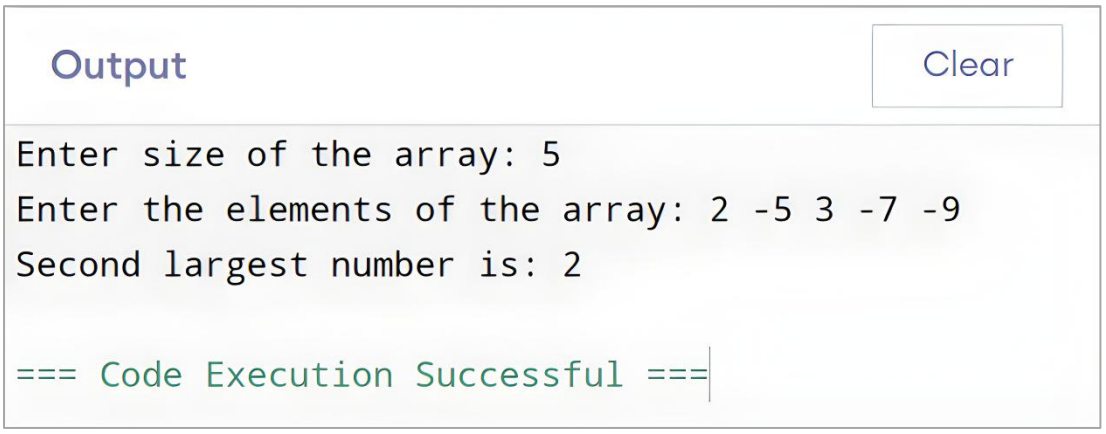
1. Start.
2. Input the size of the array (N).
3. Read N elements into the array.
4. Initialize lar1 and lar2 with the first element.
5. Traverse the array:
    - If current element > lar1:
        - Set `lar2 = lar1`
        - Set `lar1 = current element`
    - Else if current element > `lar2` and < `lar1`:
        - Update `lar2 = current element`
6. Print the value of `lar2` as the second largest number.
7. End.

08.2. C Program Code:

```
#include<stdio.h>

int main(){
    int N, lar1, lar2;

    printf("Enter size of the array: ");
    scanf("%d", & N);

    int arr[N];
    printf("Enter the elements of the array: ");
    for(int i = 0; i < N; ++i) {
        scanf("%d", & arr[i]);
    }

    lar1 = lar2 = arr[0];
    for(int i = 0; i < N; ++i) {
        if (arr[i] > lar1) {
            lar2 = lar1;
            lar1 = arr[i];
        }
        else if (arr[i] > lar2 && arr[i] < lar1) {
            lar2 = arr[i];
        }
    }
```

```
    printf("Second largest number is: %d", lar2);

    return 0;
}
```

08.3. Output:



08.4. Discussion:

This program finds the second largest number in an array without sorting. We first read all elements and initialize two variables to track the largest and second largest values. As we scan the array, we update these values based on comparisons. By the end of the loop, `lar1` holds the largest number and `lar2` holds the second largest. This method is efficient because it needs only one pass through the array.

Problem 09. Write a C program to insert an element in an array at a given position.

09.1. Algorithm: Algorithm to Factorial using For Loop

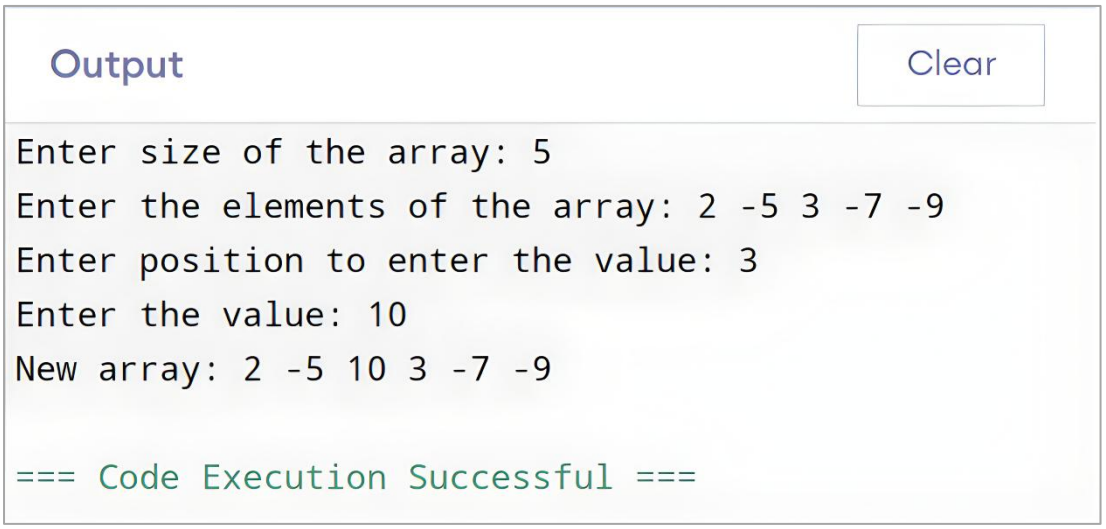
1. Start.
2. Input the size of the array (N) and its elements.
3. Input the position `pos` and value to insert.
4. Check if `pos` is valid ($1 \leq pos \leq N+1$).
    - If invalid → print error and stop.
5. Shift elements from position pos to the right to make space.
6. Insert the new value at `arr[pos-1]`.
7. Increment the size of the array.
8. Print the updated array.
9. End.

09.2. C Program Code:

```
#include <stdio.h>

int main(){
    int N, pos, value;

    printf("Enter size of the array: ");
    scanf("%d", &N);

    int arr[N + 1];

    printf("Enter the elements of the array: ");
    for(int i = 0; i < N; i++){
        scanf("%d", &arr[i]);
    }

    printf("Enter position to enter the value: ");
    scanf("%d", &pos);

    printf("Enter the value: ");
    scanf("%d", &value);

    if (pos < 1 || pos > N + 1){
        printf("Invalid position!\n");
    }
     else{
        for(int i = N; i >= pos; i--){
            arr[i] = arr[i - 1];
        }
```

```
        arr[pos - 1] = value;
        N++;

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

    return 0;
}
```

09.3. Output:



09.4. Discussion:

This program inserts an element at a specified position in an array. We first read the array and the desired position and value. If the position is valid, elements from that position onward are shifted right to create space for the new element. Finally, the new value is inserted, and the updated array is displayed. This exercise demonstrates array manipulation and element shifting in C.

Problem 10. Write a C program to delete an element from an array.

10.1. Algorithm: Algorithm to Factorial using For Loop

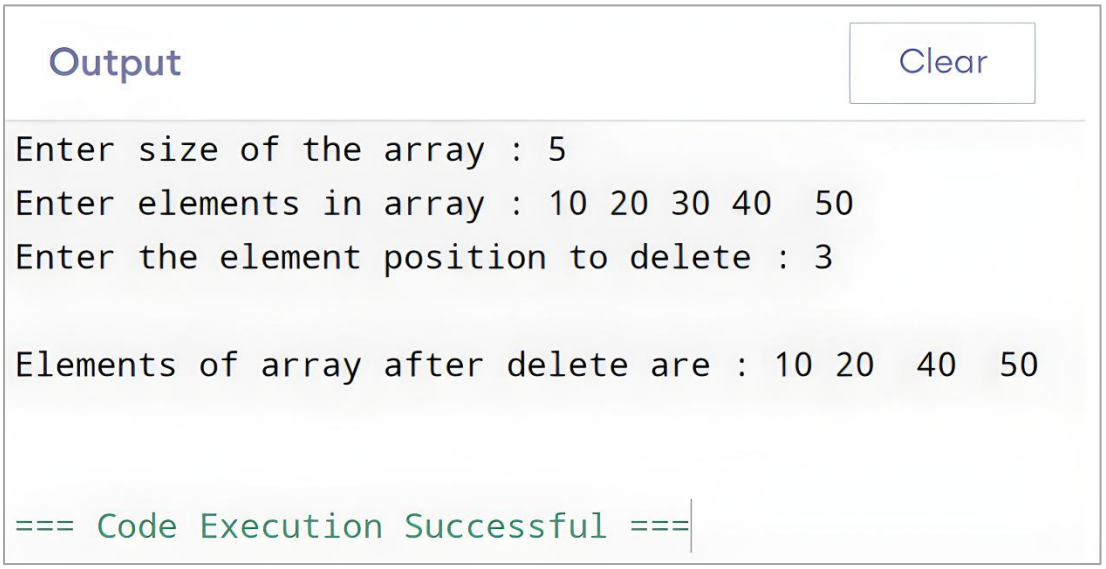
1. Start.
2. Input the size of the array and its elements.
3. Input the position pos of the element to delete.
4. Check if pos is valid ($1 \leq pos \leq size$).
    - If invalid → print error and stop.
5. Shift all elements after pos one place to the left.
6. Decrease the size of the array by 1.
7. Print the updated array.
8. End.

10.2. C Program Code:

```
#include <stdio.h>

int main(){
    int size, pos;

    printf("Enter size of the array: ");
    scanf("%d", &size);

    int arr[size];

    printf("Enter %d elements in array: ", size);
    for(int i = 0; i < size; i++){
        scanf("%d", &arr[i]);
    }

    printf("Enter the element position to delete: ");
    scanf("%d", &pos);

    if(pos < 1 || pos > size){
        printf("Invalid position! Please enter position between 1 to %d\n",
size);
    }
      else{
        for(int i = pos - 1; i < size - 1; i++){
            arr[i] = arr[i + 1];
        }
        size--;

        printf("Elements of array after delete are: ");
```

```
        for(int i = 0; i < size; i++){
            printf("%d\t", arr[i]);
        }
    }

    return 0;
}
```

10.3. Output:



10.4. Discussion:

This program deletes an element from a given position in an array. We first read the array and the position of the element to remove. If the position is valid, elements after the specified position are shifted left to overwrite the deleted element. The array size is reduced by one, and the updated array is displayed. This demonstrates array manipulation and element shifting in C.

Problem 11. Write a C program to reverse an array without using another array.

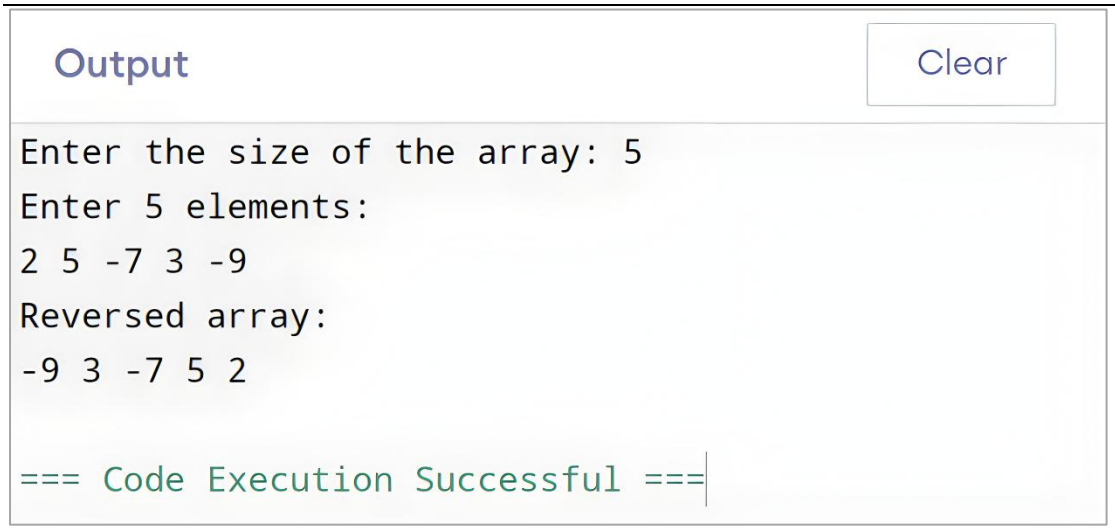
11.1. Algorithm: Algorithm to Factorial using For Loop

1. Start.
2. Input the size of the array (N).
3. Read N elements into the array.
4. Repeat from `i = 0` to `N/2`:
    - Swap `arr[i]` with `arr[N-1-i]`.
5. Print the reversed array.
6. End.

11.2. C Program Code:

```
#include <stdio.h>

int main(){
    int N;

    printf("Enter the size of the array: ");
    scanf("%d", &N);

    int arr[N];

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

    for(int i = 0; i < N / 2; i++){
        int temp = arr[i];
        arr[i] = arr[N - 1 - i];
        arr[N - 1 - i] = temp;
    }

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

    return 0;
}
```

11.3. Output:



11.4. Discussion:

In this program, the array is reversed **in-place**, meaning no extra array is used. We swap the first element with the last, the second with the second last, and continue this process until the middle of the array. This method is efficient because it uses only one loop and constant extra memory. The result is the original array reversed.

Problem 12. Write a C program to find the maximum and minimum elements using recursion.

12.1. Algorithm: Algorithm to Factorial using For Loop

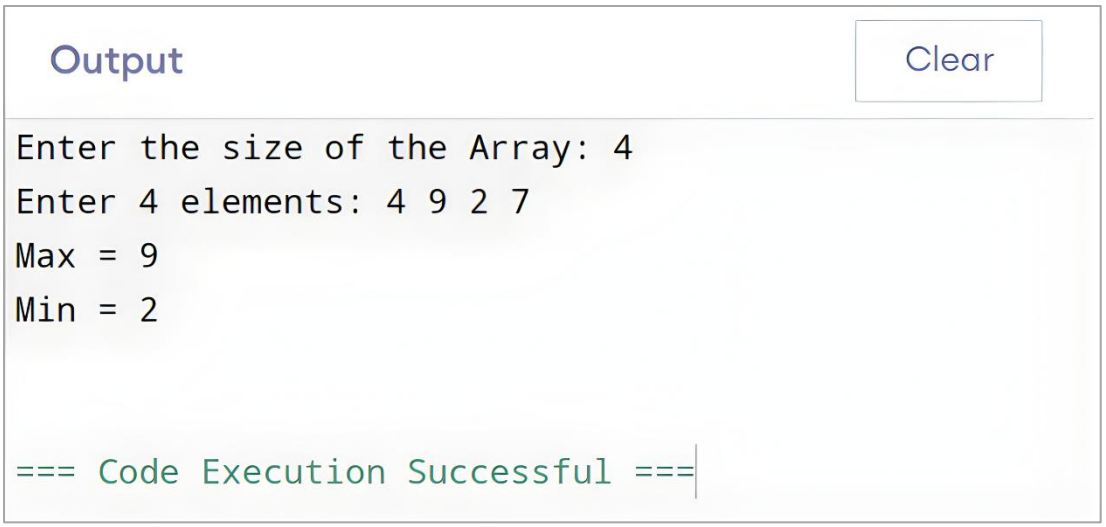
1. Start.
2. Input the size of the array (N) and read the elements.
3. Call the recursive function `findMax(arr, N)`:
    - If N = 1, return the only element.
    - Otherwise, compare the last element with the max of the remaining array.
4. Call the recursive function `findMin(arr, N)`:
    - If N = 1, return the only element.
    - Otherwise, compare the last element with the min of the remaining array.
5. Print the maximum and minimum values.
6. End.

12.2. C Program Code:

```
#include <stdio.h>

int findMax(int arr[], int N){
    if (N == 1)
        return arr[0];

    int max = findMax(arr, N - 1);
    return (arr[N - 1] > max) ? arr[N - 1] : max;
}

int findMin(int arr[], int N){
    if (N == 1)
        return arr[0];

    int min = findMin(arr, N - 1);
    return (arr[N - 1] < min) ? arr[N - 1] : min;
}

int main(){
    int N;
    printf("Enter the size of the Array: ");
    scanf("%d", &N);

    int arr[N];
    printf("Enter %d elements: ", N);
    for(int i = 0; i < N; i++){
        scanf("%d", &arr[i]);
    }
```

```
    printf("Max = %d\n", findMax(arr, N));
    printf("Min = %d\n", findMin(arr, N));

    return 0;
}
```

12.3. Output:



12.4. Discussion:

This program finds the maximum and minimum elements of an array using recursion. Each recursive function works by reducing the problem size: it finds the max/min of the first N−1 elements, then compares it with the last element to get the final result. When only one element remains, the function returns it. This experiment helped us understand how recursion can replace loops in array processing.

Problem 13. Write a C program to put even and odd elements of an array in two separate arrays.

13.1. Algorithm: Algorithm to Factorial using For Loop

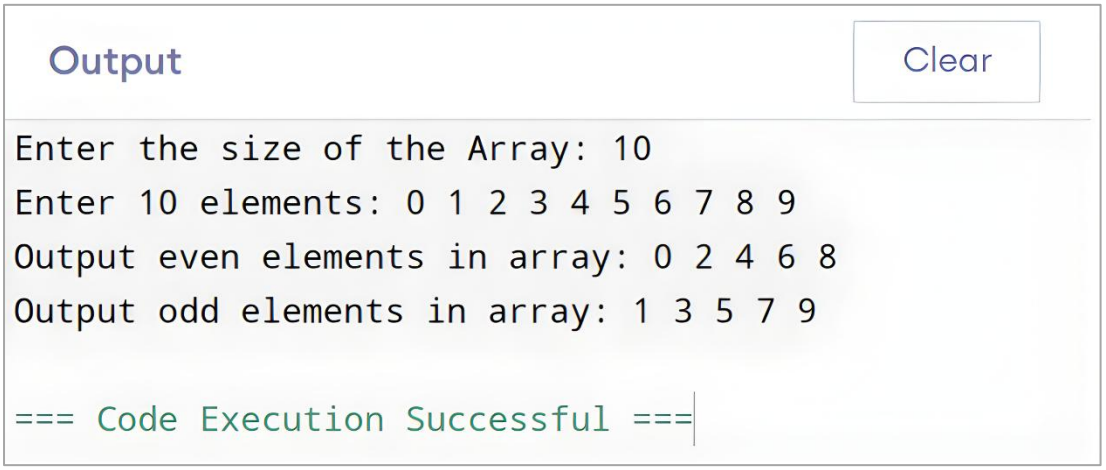
1. Start.
2. Input the size of the array (N).
3. Read N elements into `arr`.
4. Initialize two counters: `e = 0` for even, `o = 0` for odd.
5. Loop through the array from 0 to N−1:
    - If the element is even → store in `evenarr[e++]`.
    - Else → store in `oddarr[o++]`.
6. Print all elements in `evenarr`.
7. Print all elements in `oddarr`.
8. End.

13.2. C Program Code:

```
#include <stdio.h>

int main(){
    int N;
    int e = 0, o = 0;

    printf("Enter the size of the Array: ");
    scanf("%d", &N);

    int arr[N], evenarr[N], oddarr[N];

    printf("Enter %d elements: ", N);
    for(int i = 0; i < N; i++){
        scanf("%d", &arr[i]);
    }

    for(int i = 0; i < N; i++){
        if(arr[i] % 2 == 0) {
            evenarr[e++] = arr[i];
        } else {
            oddarr[o++] = arr[i];
        }
    }

    printf("Output even elements in array: ");
    for(int i = 0; i < e; i++){
        printf("%d ", evenarr[i]);
    }
```

```
    printf("\nOutput odd elements in array: ");
    for(int i = 0; i < o; i++){
        printf("%d ", oddarr[i]);
    }

    return 0;
}
```

13.3. Output:



13.4. Discussion:

In this program, the array elements are separated into two new arrays: one for even numbers and another for odd numbers. As each element is checked, it is placed in the appropriate array using counters to track positions. This approach helps practice array traversal, conditional checking, and storing values in separate arrays based on a condition.

Problem 14. Write a C program to multiply two matrices.

14.1. Algorithm: Algorithm to Factorial using For Loop

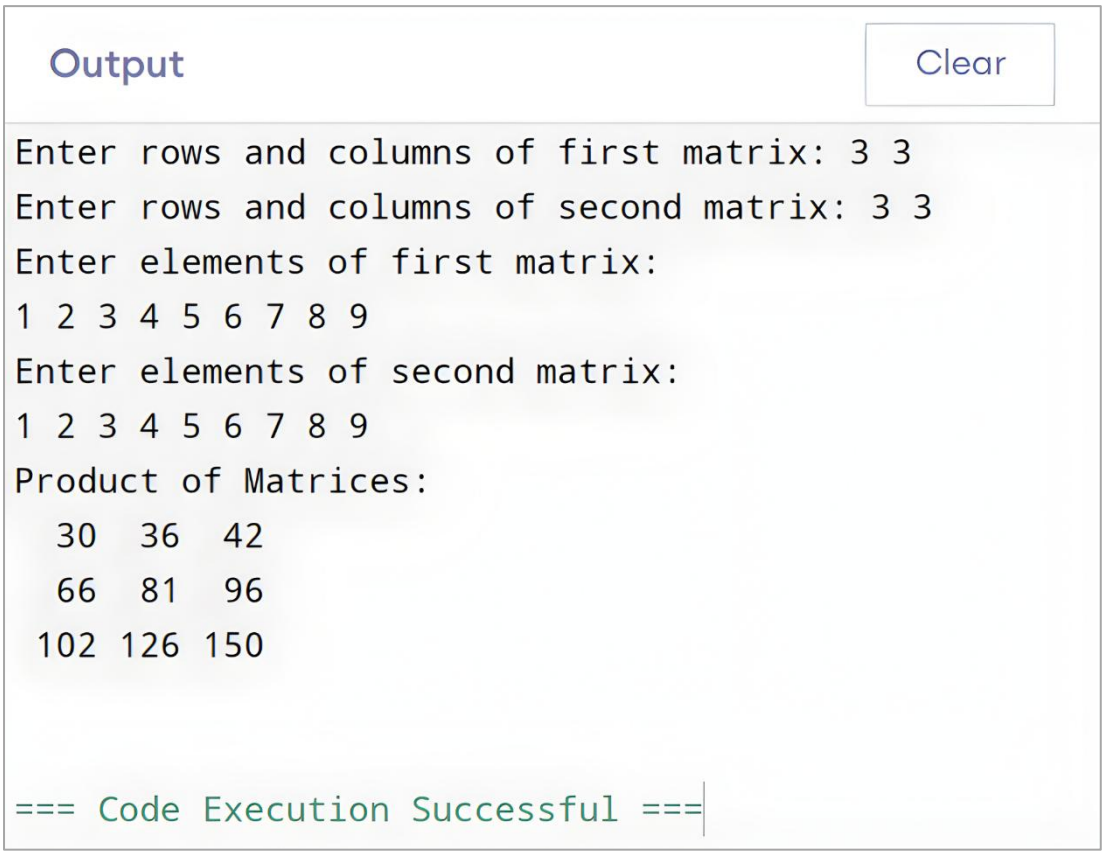
1. Start.
2. Input the rows and columns of both matrices (R1, C1, R2, C2).
3. Check if **C1 ≠ R2** → multiplication not possible, stop.
4. Input elements of both matrices.
5. Initialize product matrix `PMAT` with zeros.
6. For each element `PMAT[i][j]`:
   - Compute the sum of `MAT1[i][k] * MAT2[k][j]` for all `k` from 0 to C1−1.
7. Print the product matrix.
8. End.

14.2. C Program Code:

```
#include <stdio.h>

int main(){
    int R1, C1, R2, C2;

    printf("Enter rows and columns of first matrix: ");
    scanf("%d %d", &R1, &C1);

    printf("Enter rows and columns of second matrix: ");
    scanf("%d %d", &R2, &C2);

    if (C1 != R2){
        printf("Matrix multiplication not possible\n");
        return 0;
    }

    int MAT1[R1][C1], MAT2[R2][C2], PMAT[R1][C2];

    printf("Enter elements of first matrix:\n");
    for(int i = 0; i < R1; i++){
        for(int j = 0; j < C1; j++){
            scanf("%d", &MAT1[i][j]);
        }
    }

    printf("Enter elements of second matrix:\n");
    for(int i = 0; i < R2; i++){
        for(int j = 0; j < C2; j++){
            scanf("%d", &MAT2[i][j]);
```

```
        }
    }


    for(int i = 0; i < R1; i++){
        for(int j = 0; j < C2; j++){
            PMAT[i][j] = 0;
            for(int k = 0; k < C1; k++){
                PMAT[i][j] += MAT1[i][k] * MAT2[k][j];
            }
        }
    }

    printf("Product of Matrices:\n");
    for (int i = 0; i < R1; i++) {
        for (int j = 0; j < C2; j++) {
            printf("%4d", PMAT[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

14.3. Output:



14.4. Discussion:

This program multiplies two matrices using the standard matrix multiplication rule. Each element of the resulting matrix is calculated as the sum of products of corresponding elements from a row of the first matrix and a column of the second matrix. The program first checks if multiplication is possible by comparing dimensions, then performs the multiplication efficiently using nested loops. This exercise helps understand matrix operations and nested loop handling in C.

Problem 15. Write a C program to count the frequency of each character in a string using a loop.

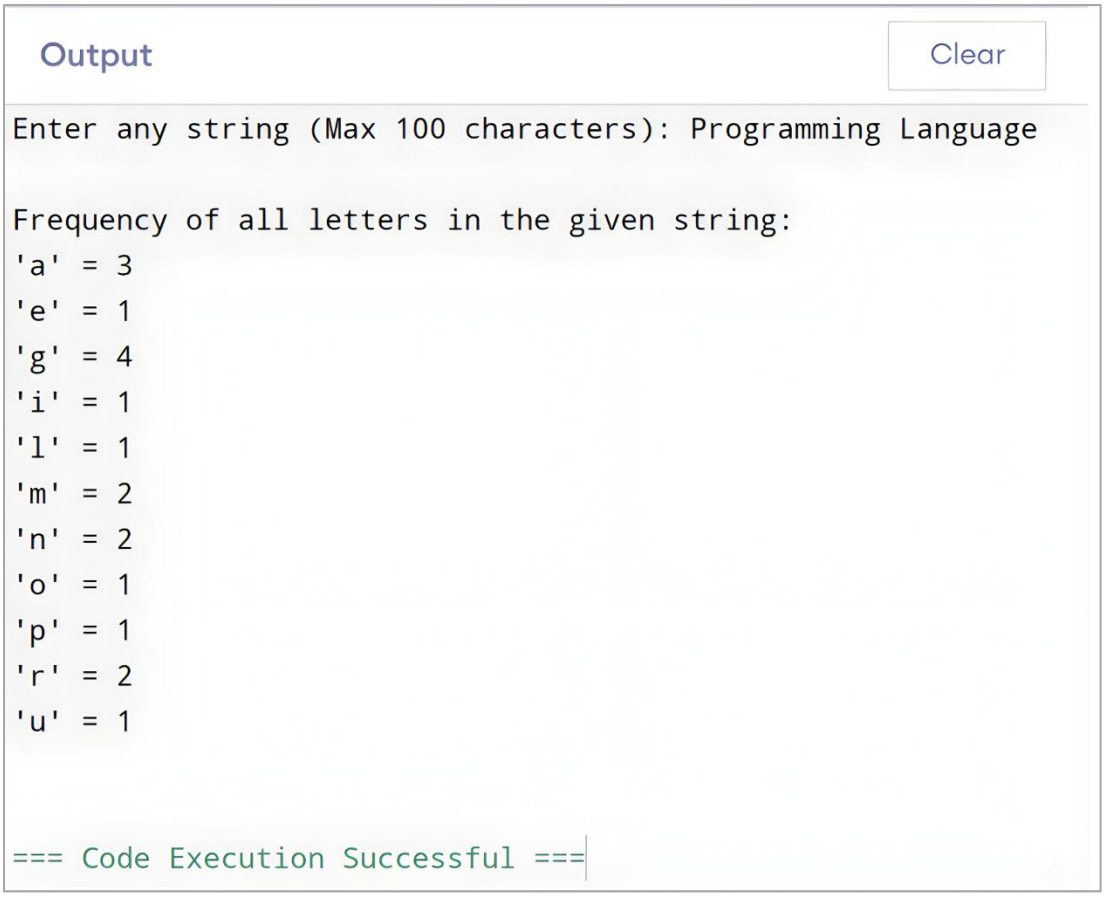
15.1. Algorithm: Algorithm to Factorial using For Loop

1. Start.
2. Input a string (maximum 100 characters).
3. Initialize an array `freq[26]` to zero for storing letter frequencies.
4. Traverse each character of the string:
   - If it is uppercase → increment `freq[char - 'A']`.
   - If it is lowercase → increment `freq[char - 'a']`.
5. Loop through `freq` array and print the letters with non-zero frequency.
6. End.

15.2. C Program Code:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[101];
    int freq[26] = {0};

    printf("Enter any string (Max 100 characters): ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';

    for(int i = 0; str[i] != '\0'; i++){
        if(str[i] >= 'A' && str[i] <= 'Z'){
            freq[str[i] - 'A']++;
        }
        else if(str[i] >= 'a' && str[i] <= 'z'){
            freq[str[i] - 'a']++;
        }
    }

    printf("\nFrequency of all letters in the given string:\n");
    for(int i = 0; i < 26; i++){
        if(freq[i] > 0){
            printf("'%c' = %d\n", i + 'a', freq[i]);
        }
    }

    return 0;
}
```

15.3. Output:



15.4. Discussion:

The This program counts the frequency of each alphabet letter in a string. Uppercase and lowercase letters are treated equally by converting them to indices in the `freq` array. By using a simple loop and array indexing, the program efficiently calculates and displays how many times each letter occurs in the input string. This exercise demonstrates string traversal, character handling, and frequency counting in C.

Problem 16. Write a C program to check whether a string is a palindrome or not without using a 'for' loop.

16.1. Algorithm: Algorithm to Factorial using For Loop

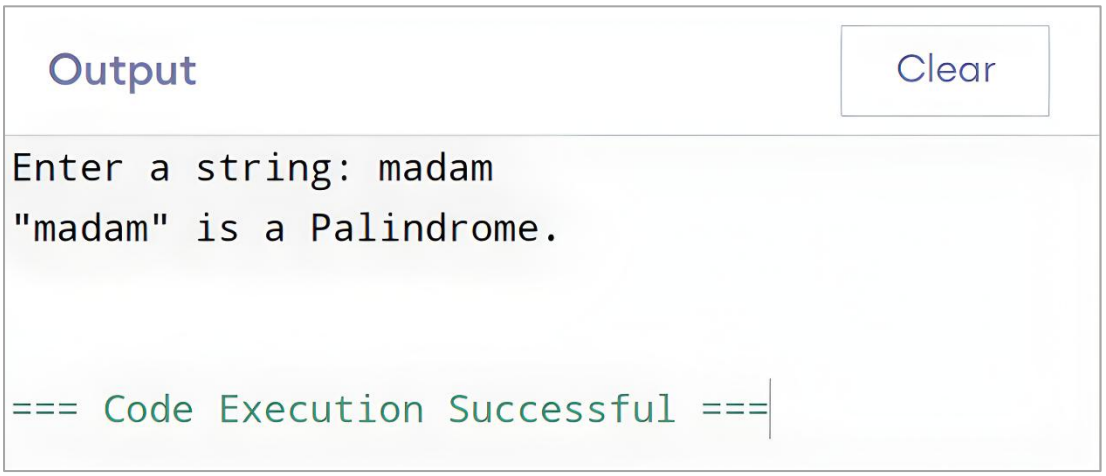
1. Start.
2. Input a string.
3. Find the length of the string.
4. Initialize `i = 0` and `isPalindrome = 1`.
5. While `i < length/2`:
    - Compare `str[i]` with `str[length-1-i]`.
    - If not equal → set `isPalindrome = 0` and break.
    - Increment `i`.
6. If `isPalindrome = 1` → print "`Palindrome`".
7. Else → print "`Not a Palindrome`".
8. End.

16.2. C Program Code:

```
#include <stdio.h>
#include <string.h>

int main(){
    char str[500];

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';

    int len = strlen(str);
    int i = 0;
    int isPalindrome = 1;

    while(i < len / 2){
        if(str[i] != str[len - 1 - i]){
            isPalindrome = 0;
            break;
        }
        i++;
    }

    if(isPalindrome)
        printf("\"%s\" is a Palindrome.\n", str);
    else
        printf("\"%s\" is not a Palindrome.\n", str);
```

```
    return 0;
}
```

16.3. Output:



16.4. Discussion:

The This program checks if a string is a palindrome without using a `for` loop. It compares characters from the start and end moving toward the center. If any pair of characters differ, the string is not a palindrome. Using a `while` loop simplifies the process and avoids extra array manipulation. This demonstrates string traversal and conditional checking in C.

Problem 17. Write a C program to enter elements in two matrices and check whether both matrices are equal or not.

17.1. Algorithm: Algorithm to Check two matrices

1. Start.
2. Input R1, C1 and R2, C2.
3. If R1 ≠ R2 or C1 ≠ C2 → print "Matrices cannot be compared" and stop.
4. Read elements of Matrix 1.
5. Read elements of Matrix 2.
6. Compare both matrices element by element.
7. If all elements match → print "Matrices are equal".
8. Otherwise → print "Matrices are not equal".
9. End.

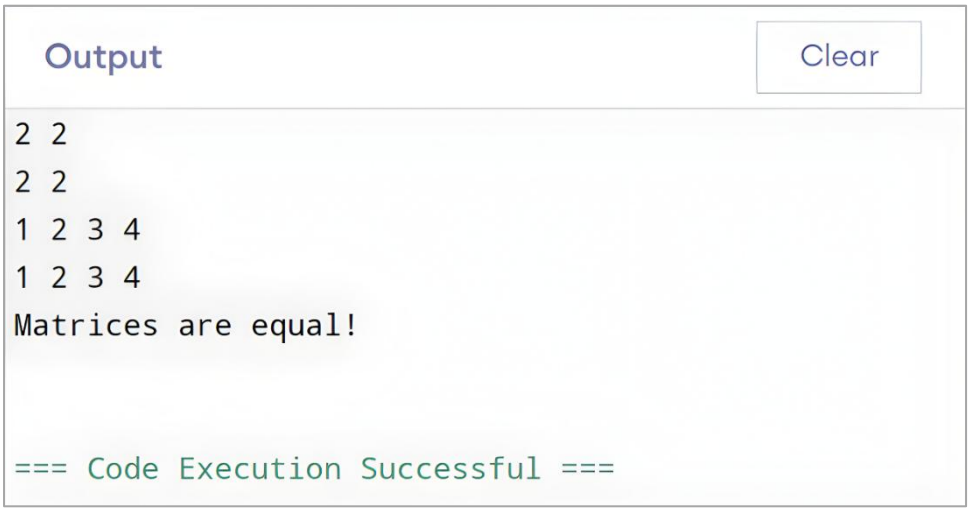
17.2. C Program Code:

```
#include <stdio.h>

int main(){
    int R1, C1, R2, C2;
    scanf("%d %d", &R1, &C1);
    scanf("%d %d", &R2, &C2);

    if(C1!=R2) {
        printf("Matrices cannot be compared for equality (different
dimensions)\n");
        return 0;
    }

    int MAT1[R1][C1], MAT2[R2][C2];
    for(int i=0; i<R1; i++) {
        for(int j=0; j<C1; j++) {
            scanf("%d", &MAT1[i][j]);
        }
    }
    for(int i=0; i<R2; i++) {
        for(int j=0; j<C2; j++) {
            scanf("%d", &MAT2[i][j]);
        }
    }

     int equal=1;
    for(int i=0; i<R1; i++) {
        for(int j=0; j<C1; j++) {
```

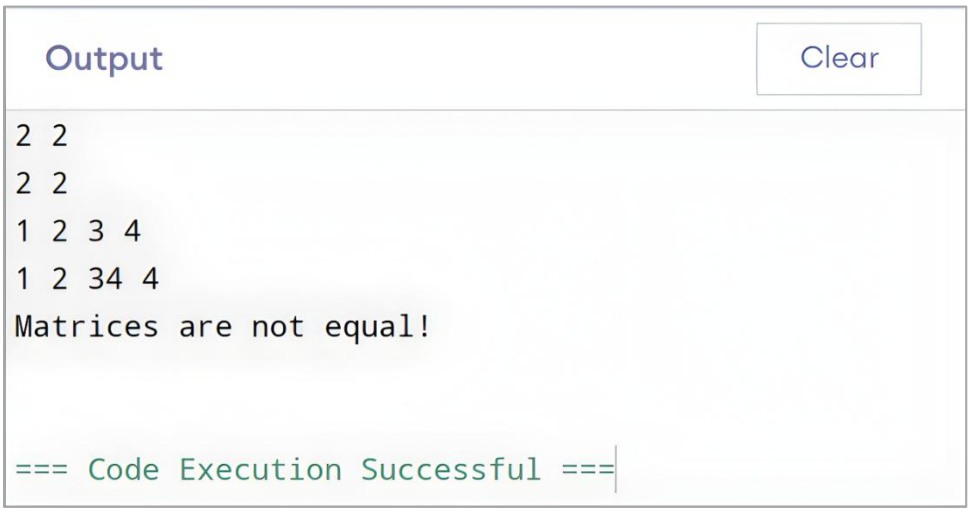
```
            if(MAT1[i][j]!=MAT2[i][j]) {
                equal=0;
                break;
            }
        }
        if(equal==0) break;
    }

    if(equal)
        printf("Matrices are equal!\n");
    else
        printf("Matrices are not equal!\n");

    return 0;
}
```

17.3. Output:

Case 01:



Case 02:



17.4. Discussion:

In this program, we check if two matrices are equal. First, the program verifies whether both matrices have the same dimensions. If not, they cannot be equal. If dimensions match, the program reads the elements of both matrices and compares each corresponding element. If all values are the same, the matrices are equal; otherwise, they are not. This experiment helped us understand matrix input, 2D arrays, and element-wise comparison in C.

Problem 18. Write a C program to compare two strings using a loop, character by character.

18.1. Algorithm: Algorithm to Factorial using For Loop

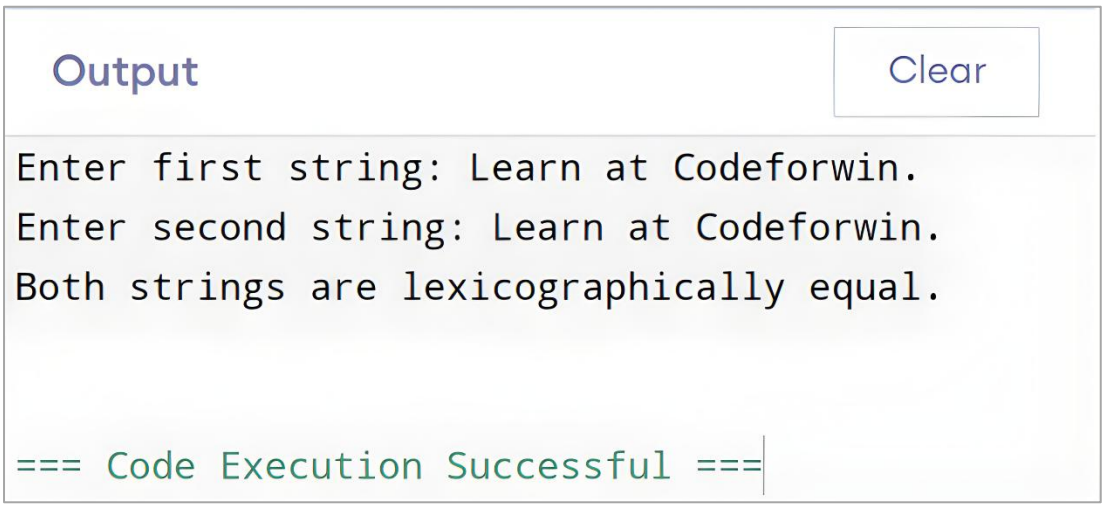
1. Start.
2. Input two strings `str1` and `str2`.
3. Initialize `i = 0`.
4. While neither string ends:
   - Compare `str1[i]` and `str2[i]`.
   - If characters differ → break.
   - Increment `i`.
5. If both strings end at the same position → print "`equal`".
6. Else, compare ASCII values to determine which is lexicographically larger.
7. End.

18.2. C Program Code:

```
#include <stdio.h>
#include <string.h>

int main() {
    char str1[500], str2[500];
    int i = 0;

    printf("Enter first string: ");
    fgets(str1, sizeof(str1), stdin);
    str1[strcspn(str1, "\n")] = '\0';

    printf("Enter second string: ");
    fgets(str2, sizeof(str2), stdin);
    str2[strcspn(str2, "\n")] = '\0';

    while(str1[i] != '\0' && str2[i] != '\0'){
        if (str1[i] != str2[i]) {
            break;
        }
        i++;
    }

    if(str1[i] == str2[i])
        printf("Both strings are lexicographically equal.\n");
    else if(str1[i] > str2[i])
        printf("First string is lexicographically larger.\n");
    else
        printf("Second string is lexicographically larger.\n");
```

```
    return 0;
}
```

18.3. Output:



18.4. Discussion:

The This program compares two strings **character by character** using a loop. It checks each corresponding character until a difference is found or the strings end. Based on the first mismatch or string length, it determines whether the strings are equal or which one is lexicographically larger. This method avoids using built-in string comparison functions and demonstrates manual string handling in C.

Problem 19. Write a C program to convert a string from lowercase to an uppercase string using a loop.

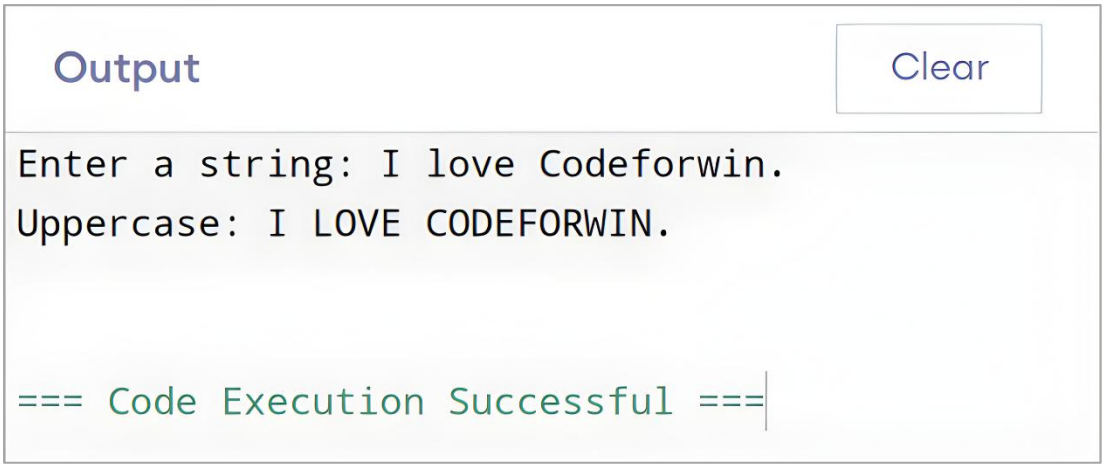
19.1. Algorithm: Algorithm to Factorial using For Loop

1. Start.
2. Input a string.
3. Loop through each character of the string:
    - If character is lowercase ('a' to 'z') → subtract 32 to convert to uppercase.
4. Print the modified string.
5. End.

19.2. C Program Code:

```
#include <stdio.h>
#include <string.h>
int main(){
    char str[500];

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';

    for(int i = 0; str[i] != '\0'; i++){
        if(str[i] >= 'a' && str[i] <= 'z'){
            str[i] -= 32;
        }
    }

    printf("Uppercase: %s\n", str);
    return 0;
}
```

19.3. Output:



19.4. Discussion:

This program converts a lowercase string to uppercase using a loop. Each character is checked, and if it is lowercase, its ASCII value is adjusted to get the corresponding uppercase character. This demonstrates string traversal and character manipulation in C without using built-in functions like `toupper()`, `strupr()`.

Problem 20. Write a C program to find the reverse of a given string using a loop.

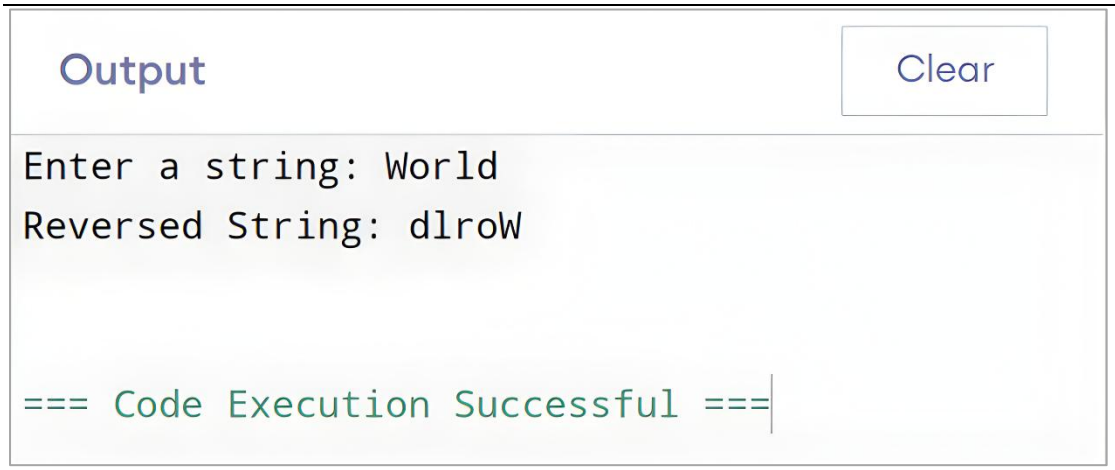
20.1. Algorithm: Algorithm to Factorial using For Loop

1. Start.
2. Input a string.
3. Find the length of the string.
4. Loop from `i = 0` to `length/2`:
    - Swap `str[i]` with `str[length - 1 - i]`.
5. Print the reversed string.
6. End.

20.2. C Program Code:

```
#include <stdio.h>
#include <string.h>

int main(){
    char str[500];

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';

    int len = strlen(str);

    for(int i = 0; i < len / 2; i++){
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }

    printf("Reversed String: %s\n", str);

    return 0;
}
```

20.3. Output:



20.4. Discussion:

This program reverses a string using a loop. By swapping characters from the start and end, moving toward the center, the string is reversed **in-place** without using an extra array. This demonstrates string manipulation and the use of loops for processing character arrays in C.

Problem 21. Write a C program to perform arithmetic operations on numbers using pointers

21.1. Algorithm: Algorithm to Factorial using For Loop

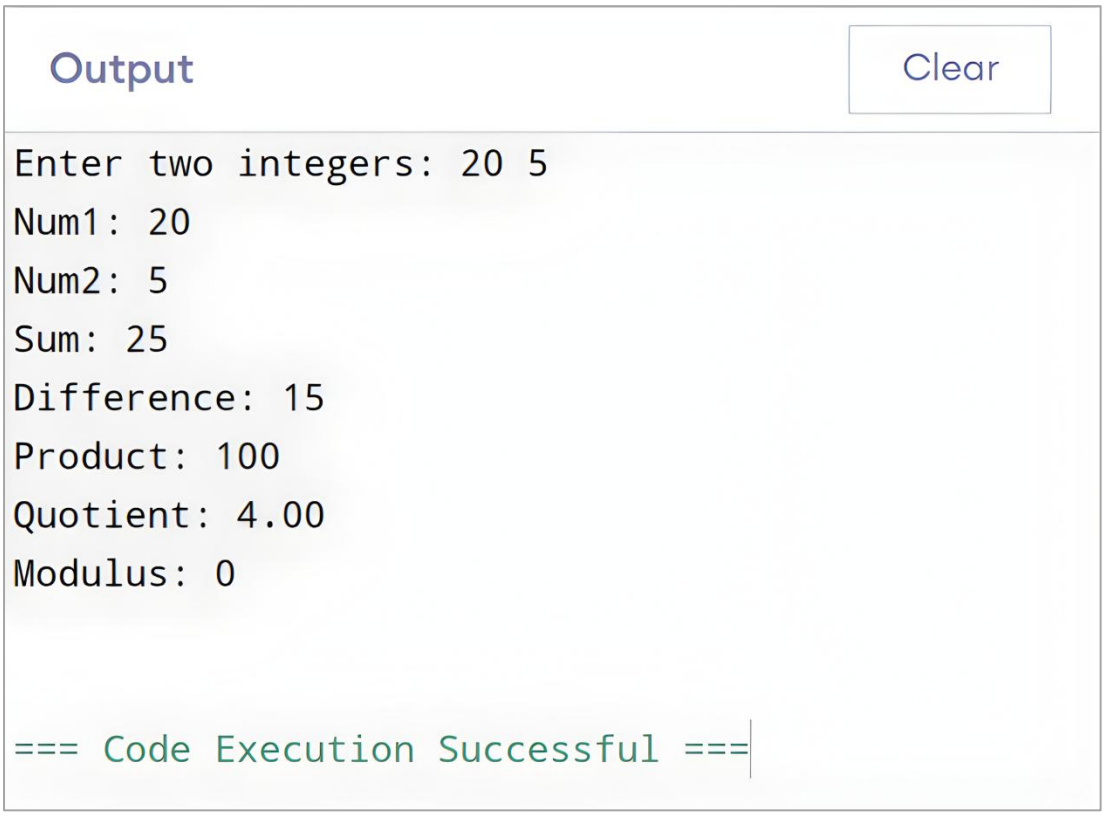
1. Start.
2. Input two integers `A` and `B`.
3. Create pointers `ptr1` and `ptr2` pointing to `A` and `B`.
4. Perform arithmetic operations using the pointers:
    - Sum: `*ptr1 + *ptr2`
    - Difference: `*ptr1 - *ptr2`
    - Product: `*ptr1 * *ptr2`
    - If `*ptr2 != 0` → calculate Quotient and Modulus.
5. Print all results.
6. End.

21.2. C Program Code:

```
#include <stdio.h>

int main() {
    int A, B;
    int *ptr1 = &A, *ptr2 = &B;
    int sum, diff, prod, mod;
    float quo;

    printf("Enter two integers: ");
    scanf("%d %d", &A, &B);

    sum = *ptr1 + *ptr2;
    diff = *ptr1 - *ptr2;
    prod = (*ptr1) * (*ptr2);

    printf("Num1: %d\nNum2: %d\n", *ptr1, *ptr2);
    printf("Sum: %d\n", sum);
    printf("Difference: %d\n", diff);
    printf("Product: %d\n", prod);

    if(*ptr2 != 0){
        quo = (float)(*ptr1) / (*ptr2);
        mod = (*ptr1) % (*ptr2);
        printf("Quotient: %.2f\n", quo);
        printf("Modulus: %d\n", mod);
    }
    else{
        printf("Division or modulus by zero is not allowed.\n");
```

```
    }

    return 0;
}
```

21.3. Output:



21.4. Discussion:

This program demonstrates performing arithmetic operations using pointers. Pointers `ptr1` and `ptr2` store the addresses of the variables, and the arithmetic is done by dereferencing the pointers. Using pointers helps understand memory access and variable manipulation in C. The program also safely handles division and modulus by zero.

Problem 22. Write a C program to input elements in an array and print the array using pointers.

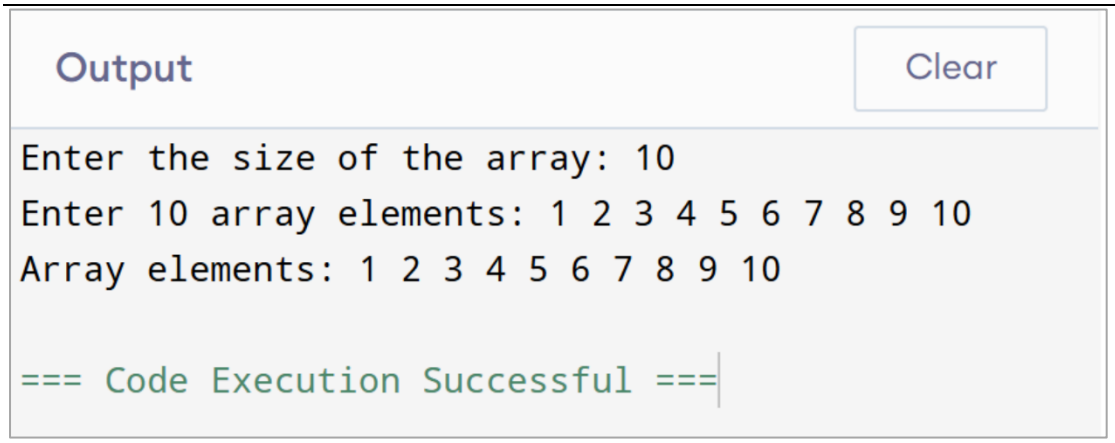
22.1. Algorithm: Algorithm to Factorial using For Loop

1. Start.
2. Input the size of the array `N`.
3. Declare an array `arr[N]` and a pointer `ptr` pointing to `arr`.
4. Loop from `i = 0` to `N-1`:
    - Input array elements using `ptr + i`.
5. Loop from `i = 0` to `N-1`:
    - Print elements using `*(ptr + i)`.
6. End.

22.2. C Program Code:

```
#include <stdio.h>

int main(){
    int N;
    printf("Enter the size of the array: ");
    scanf("%d", &N);

    int arr[N];
    int *ptr = arr;

    printf("Enter %d array elements: ", N);
    for (int i = 0; i < N; i++){
        scanf("%d", ptr + i);
    }

    printf("Array elements: ");
    for (int i = 0; i < N; i++){
        printf("%d ", *(ptr + i));
    }

    return 0;
}
```

22.3. Output:



22.4. Discussion:

This program demonstrates array input and output using pointers. The pointer `ptr` points to the base of the array, and array elements are accessed by pointer arithmetic. This approach helps understand memory addressing, pointer manipulation, and efficient array traversal in C.

Problem 23. Write a C program to input elements into an array and search for an element in the array using pointers.

23.1. Algorithm: Algorithm to Factorial using For Loop

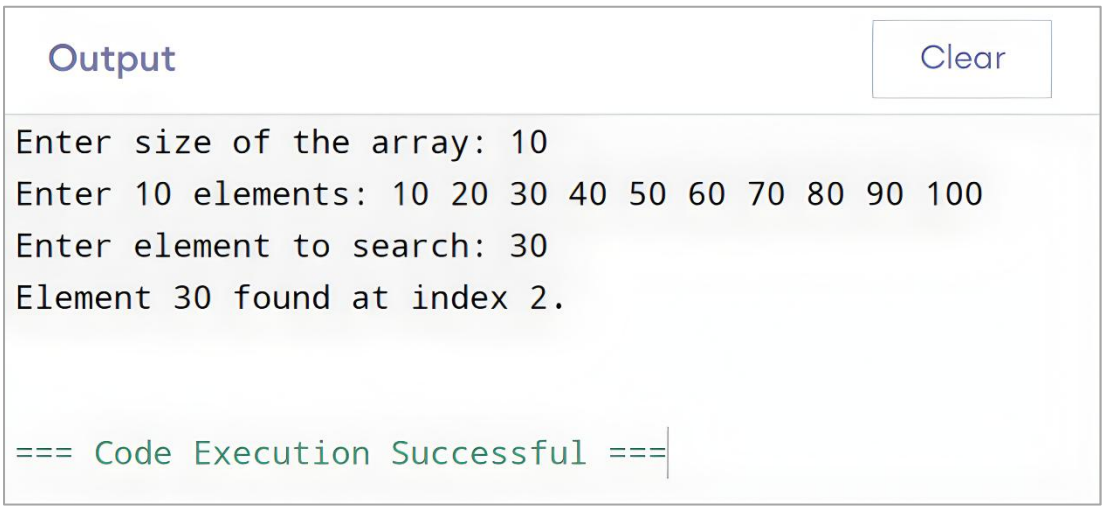
1. Start.
2. Input the size of the array `N` and read `N` elements into `arr`.
3. Initialize a pointer `ptr` to the array.
4. Input the element `S` to search.
5. Loop through the array using the pointer:
    - If `*ptr == S` → element found, print index, break.
6. If the element is not found → print "`Element not found.`"
7. End.

23.2. C Program Code:

```
#include <stdio.h>

int main(){
    int N;
    printf("Enter size of the array: ");
    scanf("%d", &N);

    int arr[N];
    printf("Enter %d elements: ", N);
    for(int i = 0; i < N; i++){
        scanf("%d", &arr[i]);
    }

    int *ptr = arr;
    int S, found = 0;
    printf("Enter element to search: ");
    scanf("%d", &S);

    for(int i = 0; i < N; i++, ptr++){
        if (*ptr == S) {
            found = 1;
            printf("Element %d found at index %d.\n", S, i);
            break;
        }
    }

    if (!found){
        printf("Element not found.\n");
    }
```

```
    return 0;
}
```

23.3. Output:



23.4. Discussion:

This program searches for a given element in an array using pointers. A pointer is used to traverse the array, comparing each element with the target. If a match is found, the program displays its index; otherwise, it indicates that the element is not present. This exercise helps understand pointer traversal, array access, and conditional checking in C.