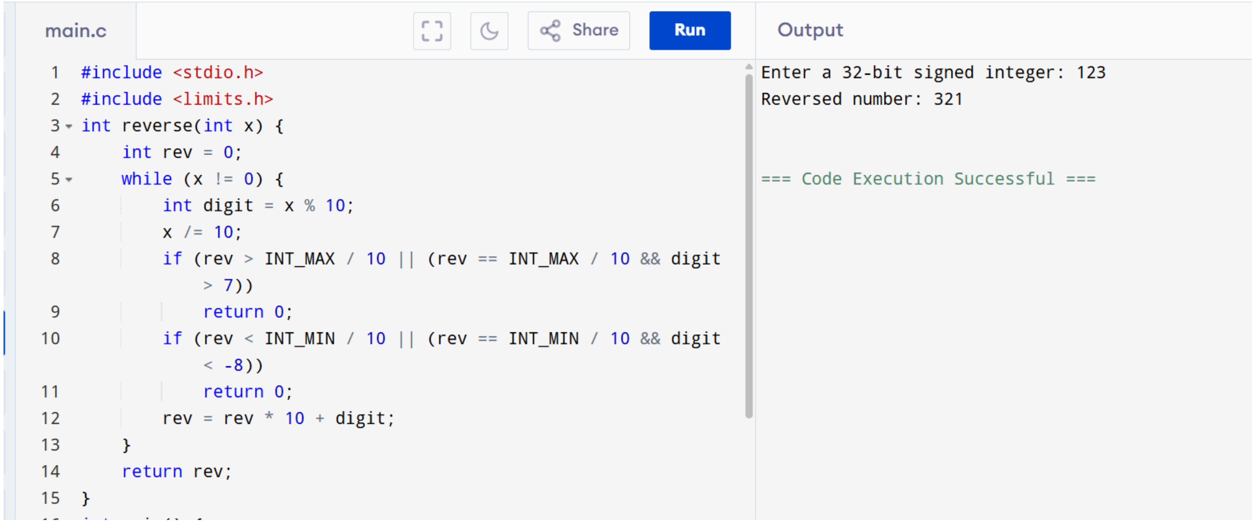
1.Write a Reversing a 32 bit signed integers by using c program

Aim: To reverse a 32-bit signed integer using C programming, ensuring the result remains within the 32-bit signed integer range (`[-2,147,483,648 to 2,147,483,647]`). If it overflows, return `0`



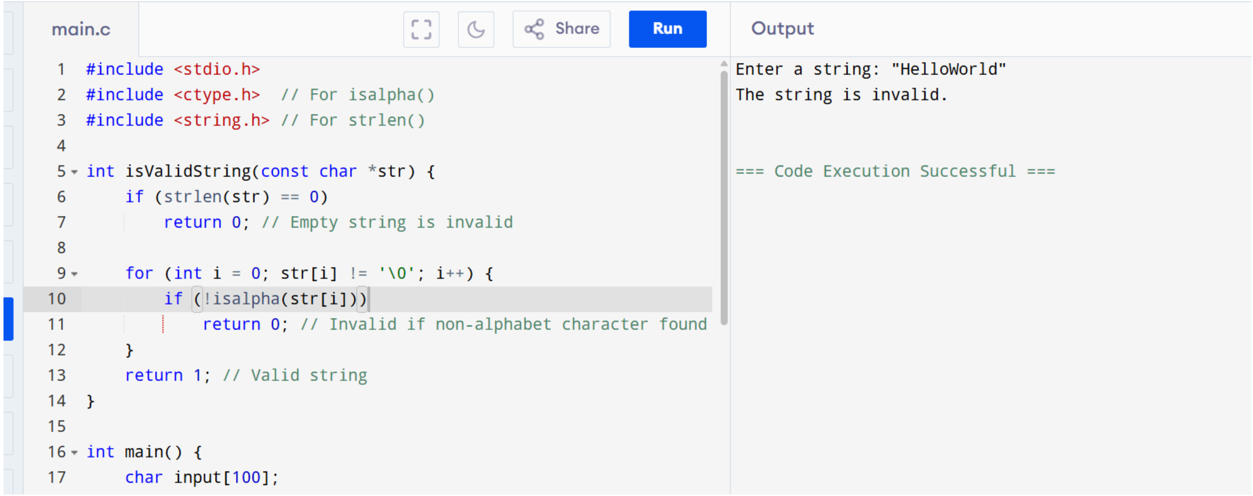
```c
#include <stdio.h>
#include <limits.h>
int reverse(int x) {
    int rev = 0;
    while (x != 0) {
        int digit = x % 10;
        x /= 10;
        if (rev > INT_MAX / 10 || (rev == INT_MAX / 10 && digit > 7))
            return 0;
        if (rev < INT_MIN / 10 || (rev == INT_MIN / 10 && digit < -8))
            return 0;
        rev = rev * 10 + digit;
    }
    return rev;
}
```

```
Enter a 32-bit signed integer: 123
Reversed number: 321


=== Code Execution Successful ===
```

`Result: code executed successfully and output is verified`

2. Write a Check for a valid String by using c programming

Aim:To check whether a given string is **valid** based on certain rules (e.g., contains only alphabetic characters, is not empty, etc.).



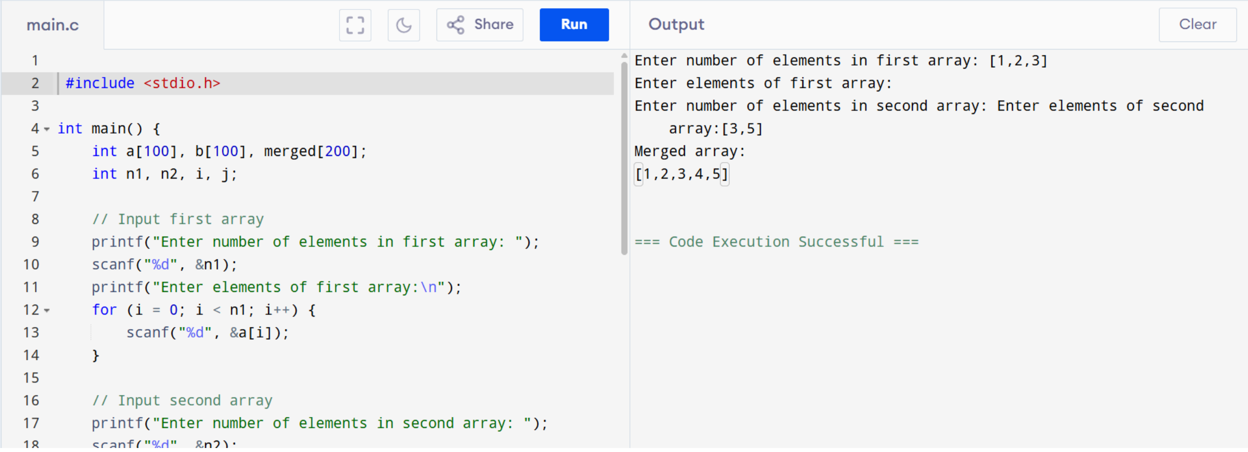
```c
#include <stdio.h>
#include <ctype.h>  // For isalpha()
#include <string.h> // For strlen()

int isValidString(const char *str) {
    if (strlen(str) == 0)
        return 0; // Empty string is invalid

    for (int i = 0; str[i] != '\0'; i++) {
        if (!isalpha(str[i]))
            return 0; // Invalid if non-alphabet character found
    }
    return 1; // Valid string
}

int main() {
    char input[100];
```

```
Enter a string: "HelloWorld"
The string is invalid.


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

3.Write a Merging two Arrays by c program

Aim:To merge two arrays into a single array in C. The merged array contains all elements of the first array followed by all elements of the second array.

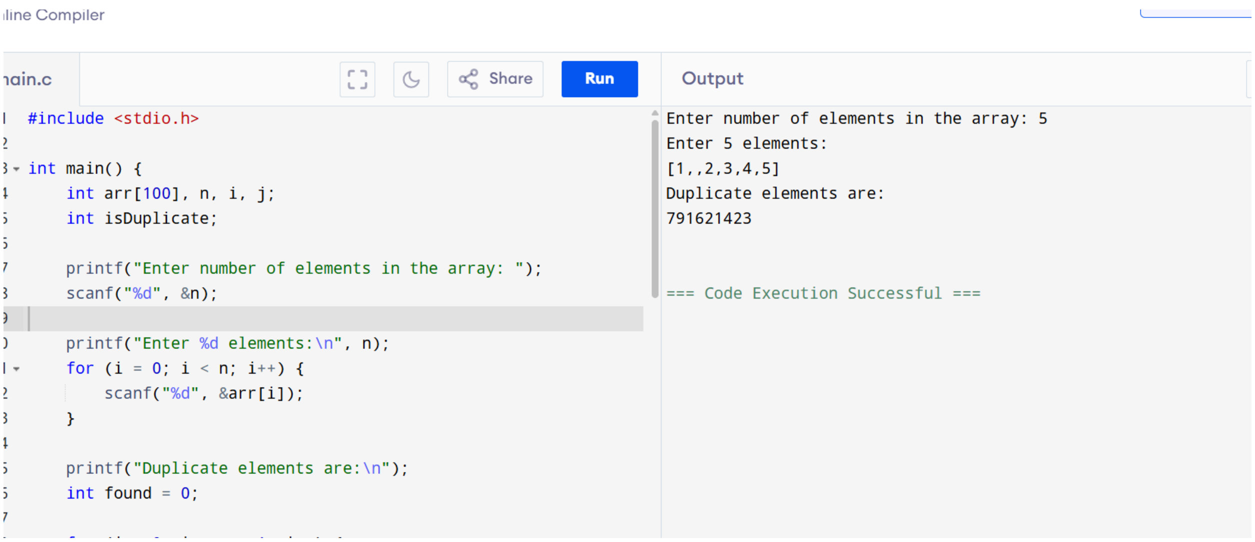


```c

#include <stdio.h>

int main() {
    int a[100], b[100], merged[200];
    int n1, n2, i, j;

    // Input first array
    printf("Enter number of elements in first array: ");
    scanf("%d", &n1);
    printf("Enter elements of first array:\n");
    for (i = 0; i < n1; i++) {
        scanf("%d", &a[i]);
    }

    // Input second array
    printf("Enter number of elements in second array: ");
    scanf("%d", &n2);
```

```
Enter number of elements in first array: [1,2,3]
Enter elements of first array:
Enter number of elements in second array: Enter elements of second
    array:[3,5]
Merged array:
[1,2,3,4,5]


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

4.Write a Given an array finding duplication values by using c programm
Aim:To find and display the duplicate elements present in a given array of integers.



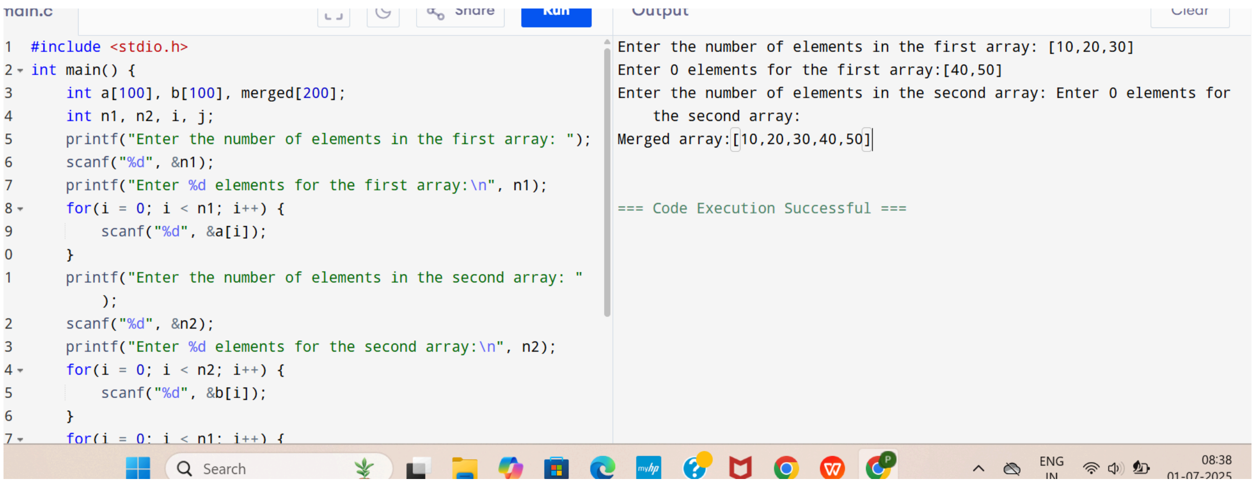
```c
#include <stdio.h>

int main() {
    int arr[100], n, i, j;
    int isDuplicate;

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

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

    printf("Duplicate elements are:\n");
    int found = 0;

```

```
Enter number of elements in the array: 5
Enter 5 elements:
[1,,2,3,4,5]
Duplicate elements are:
791621423


=== Code Execution Successful ===
```

Result: To find and display the duplicate elements present in a given array of integers.
Code executed successfully and output is verified

5. Write a Merging of list bu using c programming

Aim:To write a C program to merge two arrays (lists) into a third array.

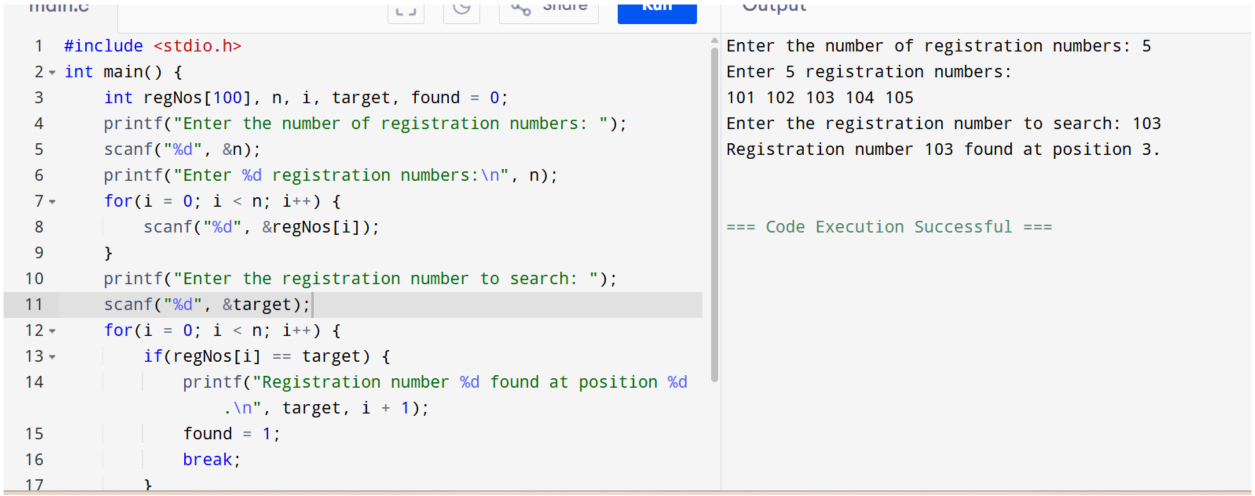


```c
#include <stdio.h>
int main() {
    int a[100], b[100], merged[200];
    int n1, n2, i, j;
    printf("Enter the number of elements in the first array: ");
    scanf("%d", &n1);
    printf("Enter %d elements for the first array:\n", n1);
    for(i = 0; i < n1; i++) {
        scanf("%d", &a[i]);
    }
    printf("Enter the number of elements in the second array: "
        );
    scanf("%d", &n2);
    printf("Enter %d elements for the second array:\n", n2);
    for(i = 0; i < n2; i++) {
        scanf("%d", &b[i]);
    }
    for(i = 0; i < n1; i++) {
```

```
Enter the number of elements in the first array: [10,20,30]
Enter 0 elements for the first array:[40,50]
Enter the number of elements in the second array: Enter 0 elements for
    the second array:
Merged array:[10,20,30,40,50]


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

6.Write a Given array of reg nos need to search for particular reg no by using c programming
Aim:To write a C program that searches for a particular registration number in a given array of registration numbers.



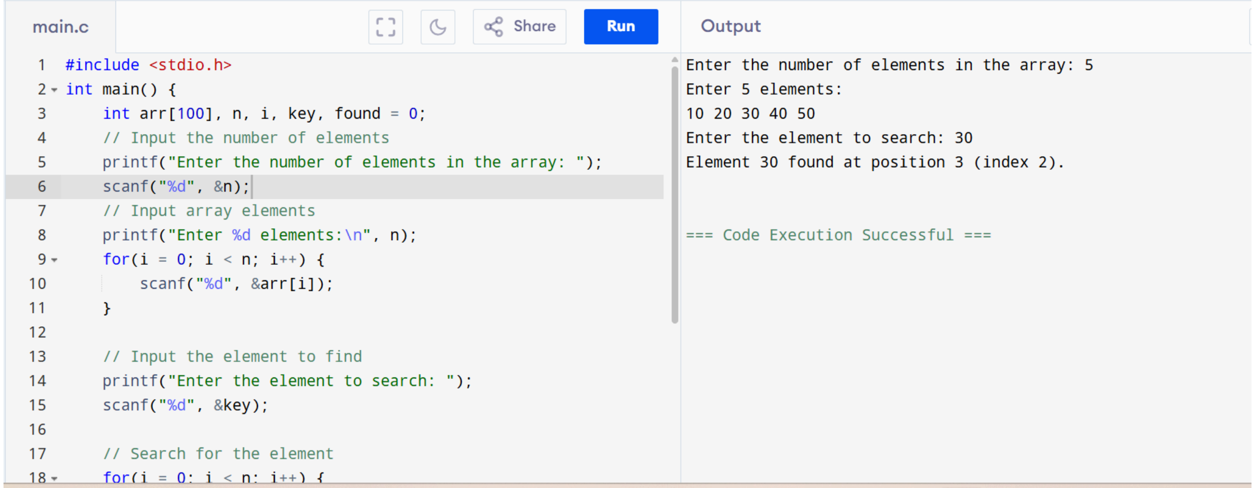
```c
#include <stdio.h>
int main() {
    int regNos[100], n, i, target, found = 0;
    printf("Enter the number of registration numbers: ");
    scanf("%d", &n);
    printf("Enter %d registration numbers:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &regNos[i]);
    }
    printf("Enter the registration number to search: ");
    scanf("%d", &target);
    for(i = 0; i < n; i++) {
        if(regNos[i] == target) {
            printf("Registration number %d found at position %d
                .\n", target, i + 1);
            found = 1;
            break;
        }
```

```
Enter the number of registration numbers: 5
Enter 5 registration numbers:
101 102 103 104 105
Enter the registration number to search: 103
Registration number 103 found at position 3.


=== Code Execution Successful ===
```

Result:To write a C program that searches for a particular registration number in a given array of registration numbers code executed successfully and output is verified

7. Identify location of element in given array by using c programming

Aim: To write a C program to identify the location (index/position) of a given element in an array.



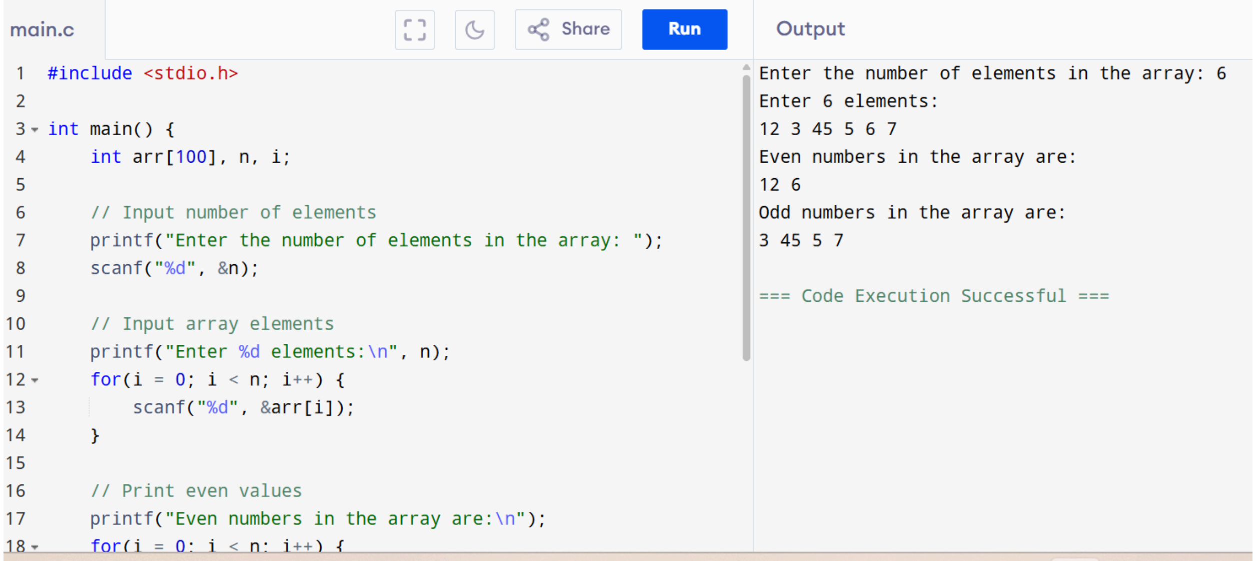
```c
#include <stdio.h>
int main() {
    int arr[100], n, i, key, found = 0;
    // Input the number of elements
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    // Input array elements
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

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

    // Search for the element
    for(i = 0; i < n; i++) {
```

```
Enter the number of elements in the array: 5
Enter 5 elements:
10 20 30 40 50
Enter the element to search: 30
Element 30 found at position 3 (index 2).


=== Code Execution Successful ===
```

Result: To write a C program to identify the location (index/position) of a given element in an array. Code executed successfully and output is verified

8. Write a Given array print odd and even values by using c programing

Aim:To write a C program that reads an array of integers and prints the **odd** and **even** values separately.



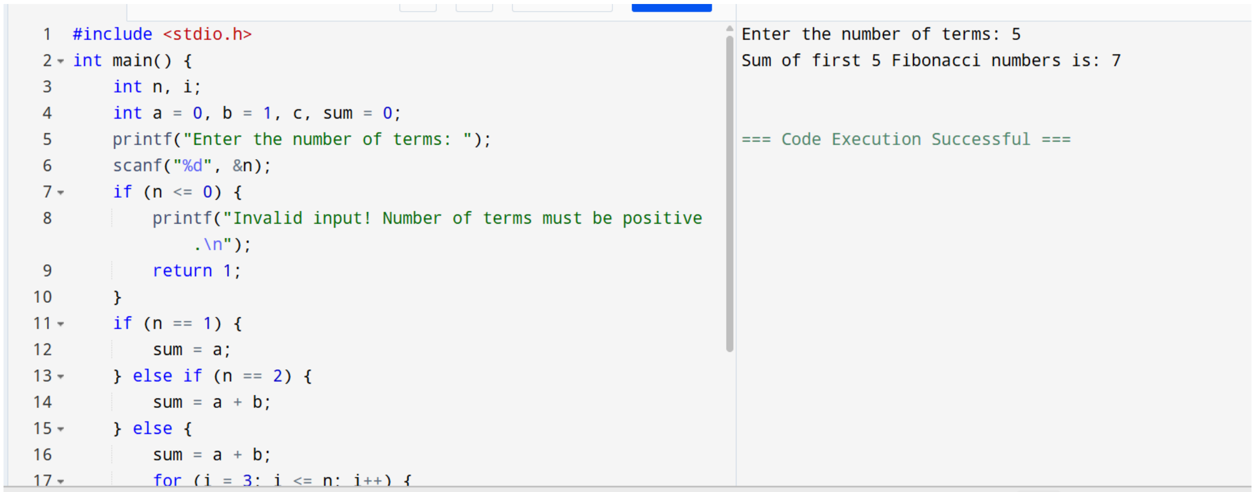
```c
#include <stdio.h>

int main() {
    int arr[100], n, i;

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

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

    // Print even values
    printf("Even numbers in the array are:\n");
    for(i = 0; i < n; i++) {
```

```
Enter the number of elements in the array: 6
Enter 6 elements:
12 3 45 5 6 7
Even numbers in the array are:
12 6
Odd numbers in the array are:
3 45 5 7

=== Code Execution Successful ===
```

Result:To write a C program that reads an array of integers and prints the **odd** and **even** values separately. Code executed successfully and output is verified

9. Write a sum of fibonacci series in a c programming language

Aim:To write a C program to calculate the **sum of the Fibonacci series** up to **n terms**.

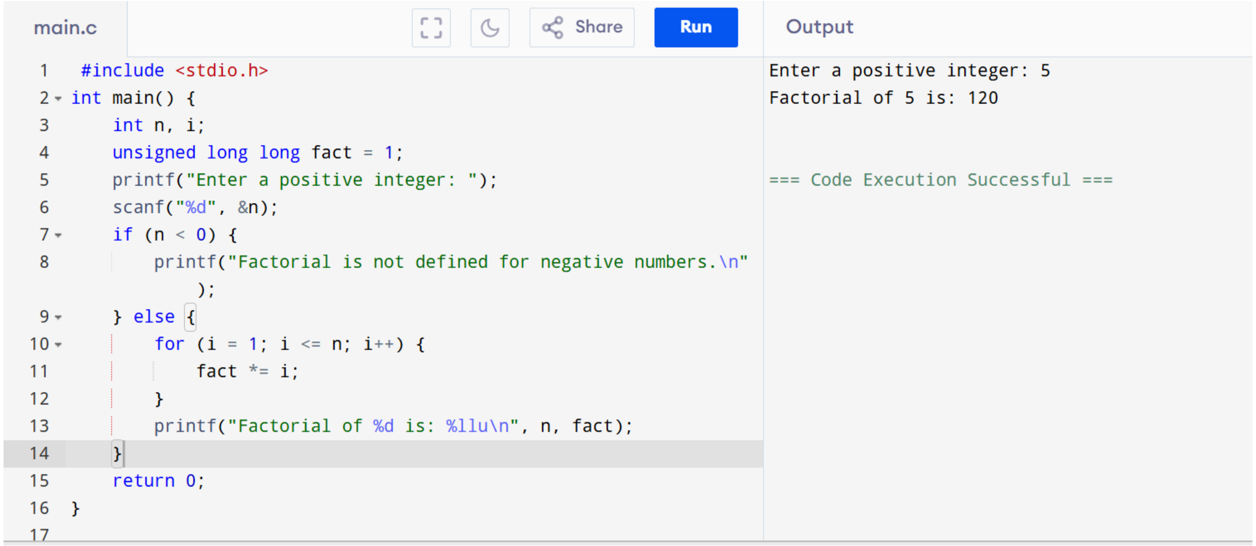


```c
#include <stdio.h>
int main() {
    int n, i;
    int a = 0, b = 1, c, sum = 0;
    printf("Enter the number of terms: ");
    scanf("%d", &n);
    if (n <= 0) {
        printf("Invalid input! Number of terms must be positive.\n");
        return 1;
    }
    if (n == 1) {
        sum = a;
    } else if (n == 2) {
        sum = a + b;
    } else {
        sum = a + b;
        for (i = 3; i <= n; i++) {
```

```
Enter the number of terms: 5
Sum of first 5 Fibonacci numbers is: 7


=== Code Execution Successful ===
```

Result:To write a C program to calculate the **sum of the Fibonacci series** up to **n terms**. Code executed successfully and output is verified

10. Finding factorial of a number by using c programming language
Aim:To write a C program to calculate the **factorial of a given number**.



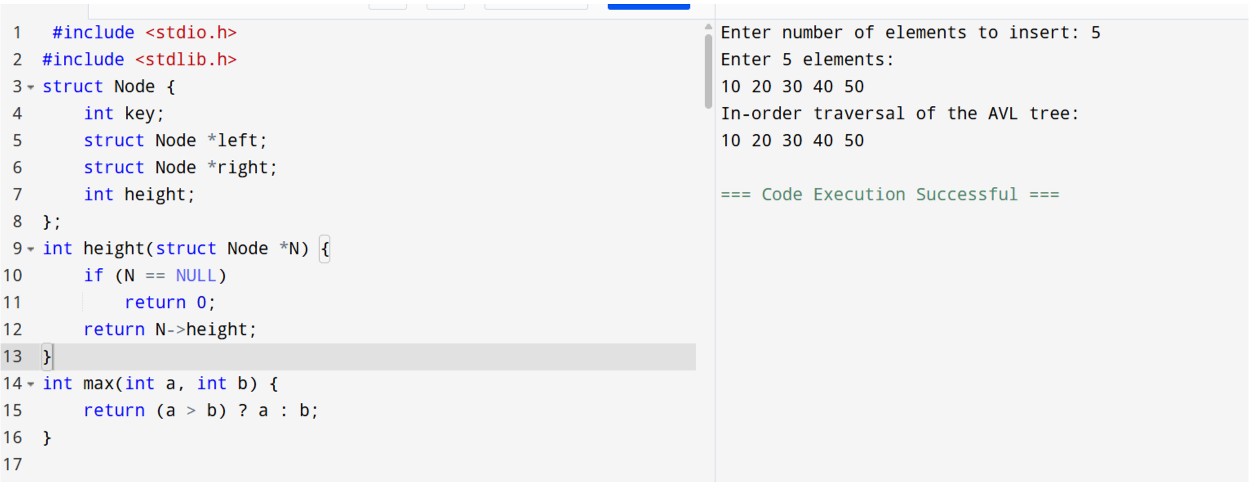
```c
#include <stdio.h>
int main() {
    int n, i;
    unsigned long long fact = 1;
    printf("Enter a positive integer: ");
    scanf("%d", &n);
    if (n < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    } else {
        for (i = 1; i <= n; i++) {
            fact *= i;
        }
        printf("Factorial of %d is: %llu\n", n, fact);
    }
    return 0;
}
```

```
Enter a positive integer: 5
Factorial of 5 is: 120


=== Code Execution Successful ===
```

Result:To write a C program to calculate the **factorial of a given number**. Is executed successfully and output is verified

11. Built a AVL tree by using c programming language

Aim:To write a C program to **implement an AVL Tree**, which performs balanced insertion of nodes to maintain height balance for efficient searching, insertion, and deletion.



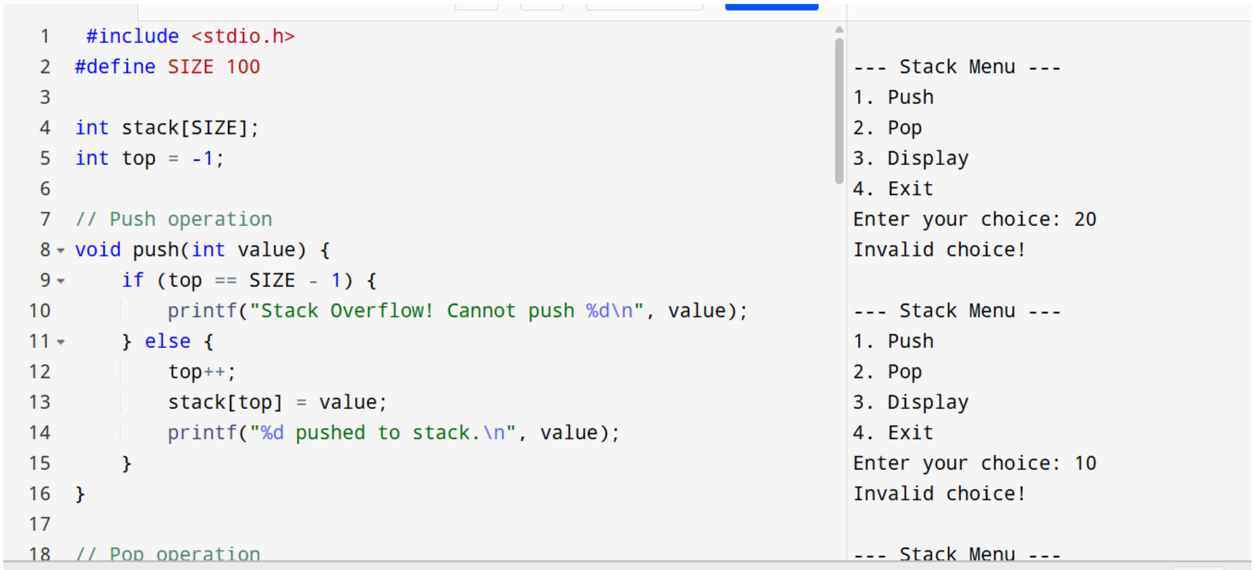
```c
 #include <stdio.h>
#include <stdlib.h>
struct Node {
    int key;
    struct Node *left;
    struct Node *right;
    int height;
};
int height(struct Node *N) {
    if (N == NULL)
        return 0;
    return N->height;
}
int max(int a, int b) {
    return (a > b) ? a : b;
}
```

```
Enter number of elements to insert: 5
Enter 5 elements:
10 20 30 40 50
In-order traversal of the AVL tree:
10 20 30 40 50

=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

12. Built Valid stack by using c programming language

Aim:To write a C program to implement and validate **stack operations** (push, pop, display), and check for **underflow** or **overflow** conditions.



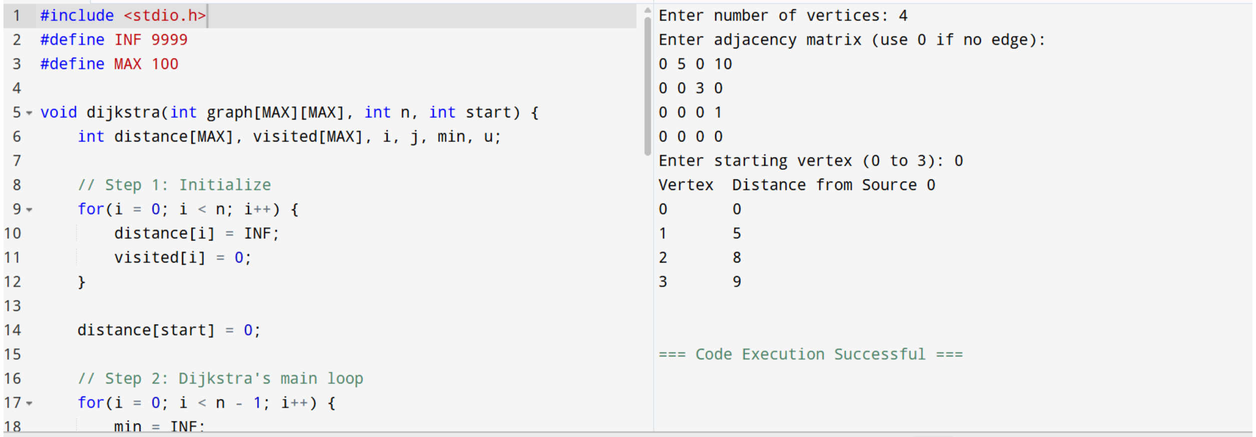
```c
 #include <stdio.h>
#define SIZE 100

int stack[SIZE];
int top = -1;

// Push operation
void push(int value) {
    if (top == SIZE - 1) {
        printf("Stack Overflow! Cannot push %d\n", value);
    } else {
        top++;
        stack[top] = value;
        printf("%d pushed to stack.\n", value);
    }
}

// Pop operation
```

```

--- Stack Menu ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 20
Invalid choice!

--- Stack Menu ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 10
Invalid choice!

--- Stack Menu ---
```

Result: code executed successfully and output is verified

13. Graph - shortest path by using c programming language

Aim:To write a C program to **find the shortest path** from a source node to all other nodes in a **graph using Dijkstra's algorithm**.



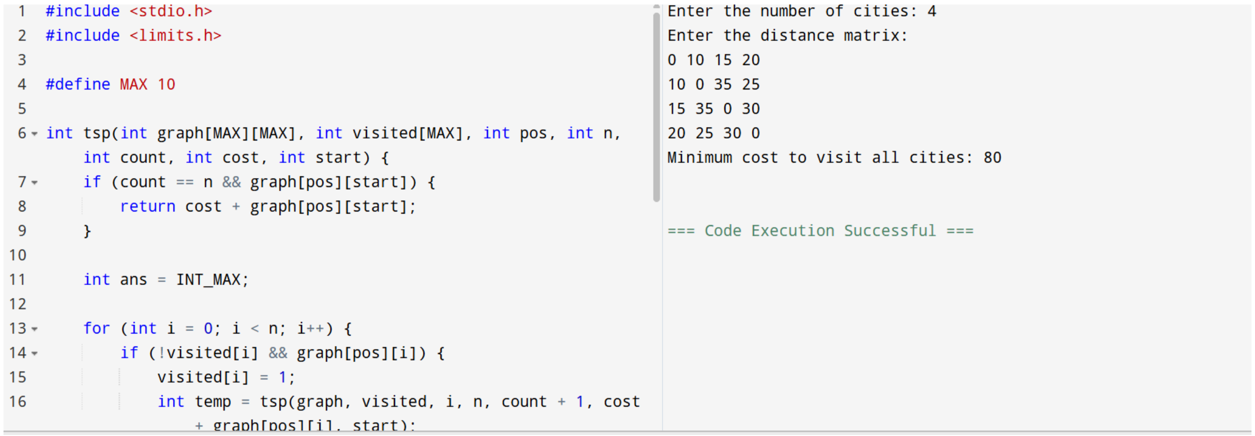
```c
#include <stdio.h>
#define INF 9999
#define MAX 100

void dijkstra(int graph[MAX][MAX], int n, int start) {
    int distance[MAX], visited[MAX], i, j, min, u;

    // Step 1: Initialize
    for(i = 0; i < n; i++) {
        distance[i] = INF;
        visited[i] = 0;
    }

    distance[start] = 0;

    // Step 2: Dijkstra's main loop
    for(i = 0; i < n - 1; i++) {
        min = INF;
```

```
Enter number of vertices: 4
Enter adjacency matrix (use 0 if no edge):
0 5 0 10
0 0 3 0
0 0 0 1
0 0 0 0
Enter starting vertex (0 to 3): 0
Vertex  Distance from Source 0
0       0
1       5
2       8
3       9


=== Code Execution Successful ===
```

Result:code is executed successfully and output is verified

14. Traveling Salesman Problem by using c programming language

Aim:To write a C program to solve the **Traveling Salesman Problem (TSP)** using a basic approach that finds the **minimum cost** path that visits every city exactly once and returns to the starting point.



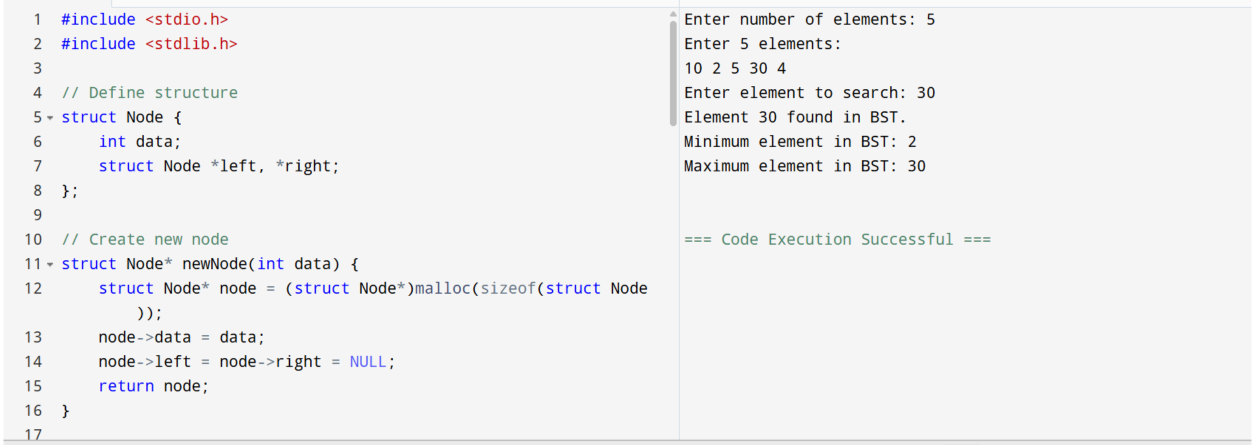
```c
#include <stdio.h>
#include <limits.h>

#define MAX 10

int tsp(int graph[MAX][MAX], int visited[MAX], int pos, int n,
    int count, int cost, int start) {
    if (count == n && graph[pos][start]) {
        return cost + graph[pos][start];
    }

    int ans = INT_MAX;

    for (int i = 0; i < n; i++) {
        if (!visited[i] && graph[pos][i]) {
            visited[i] = 1;
            int temp = tsp(graph, visited, i, n, count + 1, cost
                + graph[pos][i], start);
```

```
Enter the number of cities: 4
Enter the distance matrix:
0 10 15 20
10 0 35 25
15 35 0 30
20 25 30 0
Minimum cost to visit all cities: 80


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

15. ! Binary search tree - search for a element, min element and Max element in c program

Aim:To write a Binary search tree to search for a element and max element in c program



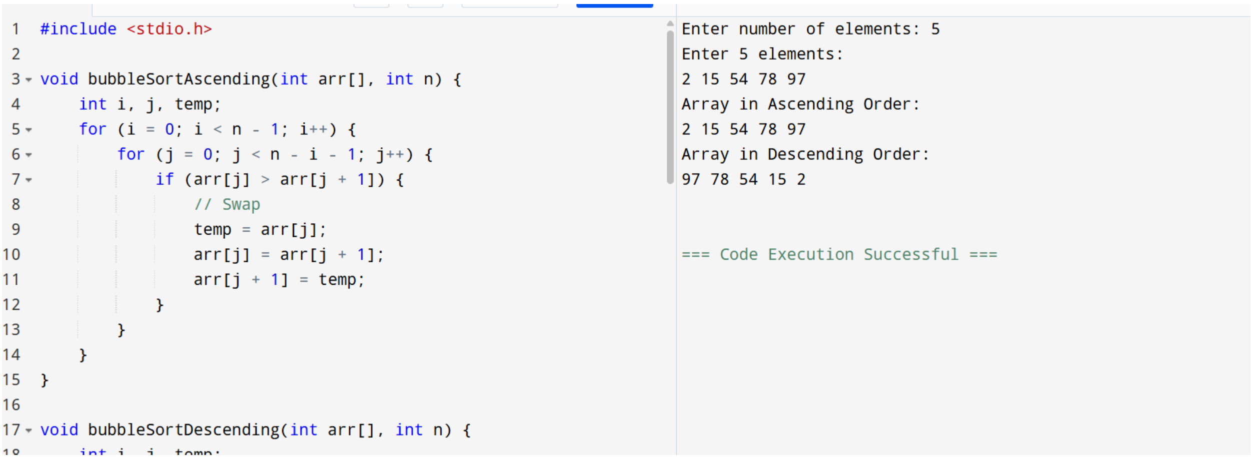
```c
#include <stdio.h>
#include <stdlib.h>

// Define structure
struct Node {
    int data;
    struct Node *left, *right;
};

// Create new node
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}
```

```
Enter number of elements: 5
Enter 5 elements:
10 2 5 30 4
Enter element to search: 30
Element 30 found in BST.
Minimum element in BST: 2
Maximum element in BST: 30


=== Code Execution Successful ===
```

Result:code execute successfully and output is verified

16. Array sort- ascending and descending by using c programming

Aim:To write a C program to **sort an array** in both **ascending** and **descending** order.



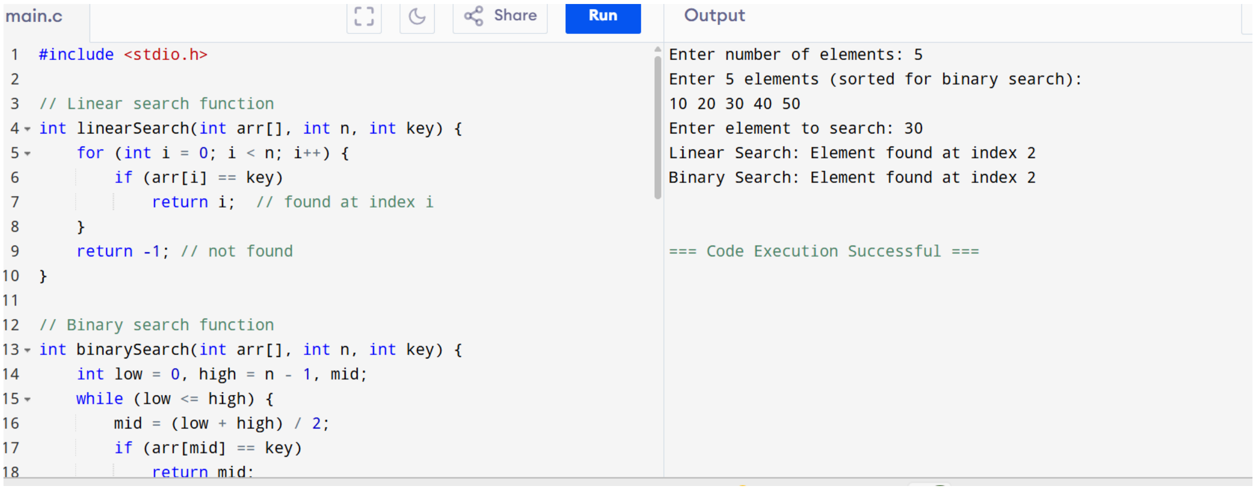
```c
#include <stdio.h>

void bubbleSortAscending(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
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void bubbleSortDescending(int arr[], int n) {
    int i, j, temp;
```

```
Enter number of elements: 5
Enter 5 elements:
2 15 54 78 97
Array in Ascending Order:
2 15 54 78 97
Array in Descending Order:
97 78 54 15 2


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

17 Array search - linear and binary by using c programming language

Aim:To write a C program to **search for an element in an array** using linear search tree and binary search tree



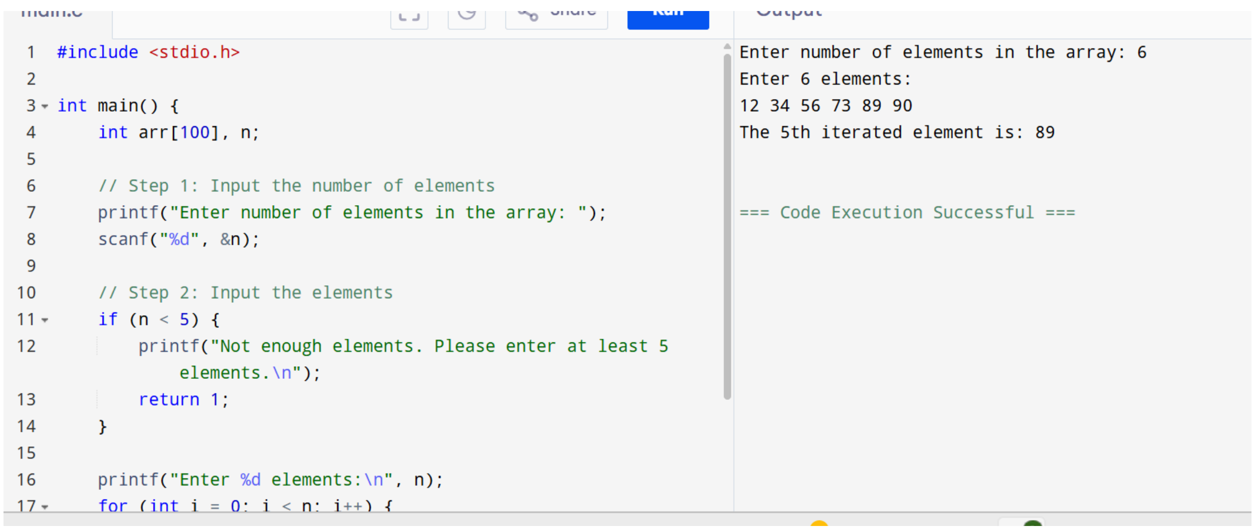
```c
#include <stdio.h>

// Linear search function
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key)
            return i;  // found at index i
    }
    return -1; // not found
}

// Binary search function
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] == key)
            return mid;
```

```
Enter number of elements: 5
Enter 5 elements (sorted for binary search):
10 20 30 40 50
Enter element to search: 30
Linear Search: Element found at index 2
Binary Search: Element found at index 2

=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

18. given set of Array elements - display 5th iterated element by using c program

Aim:To write a C program that takes a set of array elements and **displays the 5th element** in the array (i.e., the element at index 4, since arrays in C are 0-indexed).



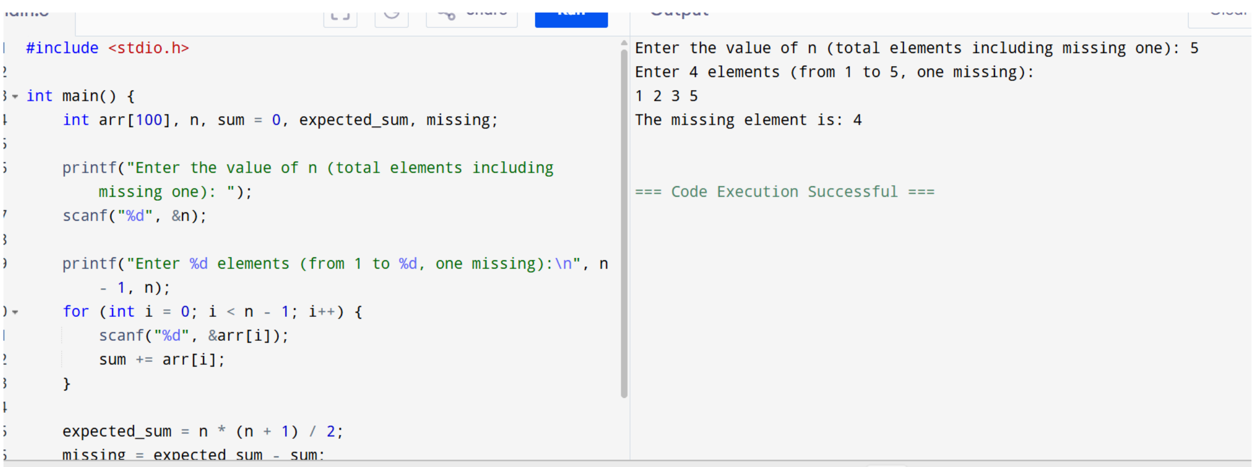
```c
#include <stdio.h>

int main() {
    int arr[100], n;

    // Step 1: Input the number of elements
    printf("Enter number of elements in the array: ");
    scanf("%d", &n);

    // Step 2: Input the elements
    if (n < 5) {
        printf("Not enough elements. Please enter at least 5 elements.\n");
        return 1;
    }

    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
```

```
Enter number of elements in the array: 6
Enter 6 elements:
12 34 56 73 89 90
The 5th iterated element is: 89

=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

19. Given unsorted array - Display missing element by using c programming

Aim:To write a C program to find the **missing number** in an **unsorted array** containing numbers from 1 to n, with exactly **one number missing**.



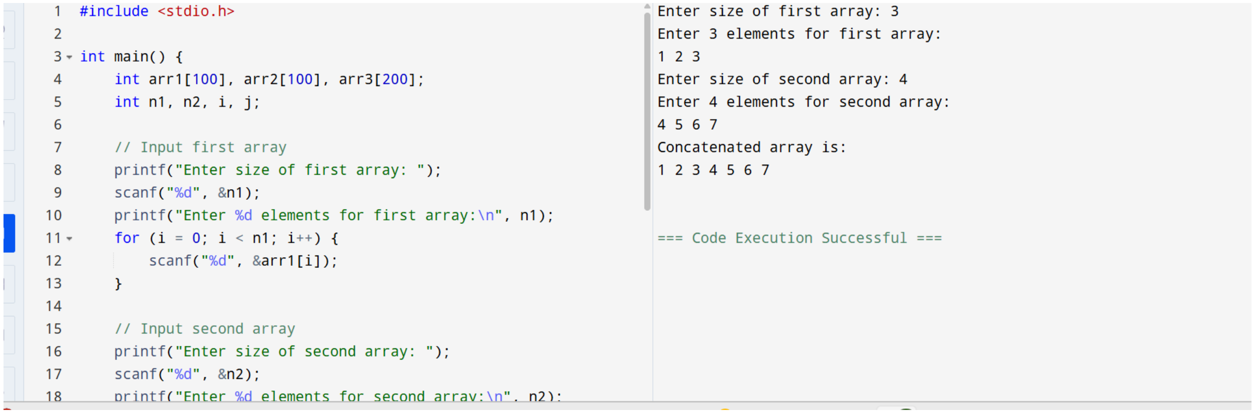
```
#include <stdio.h>

int main() {
    int arr[100], n, sum = 0, expected_sum, missing;

    printf("Enter the value of n (total elements including
        missing one): ");
    scanf("%d", &n);

    printf("Enter %d elements (from 1 to %d, one missing):\n", n
        - 1, n);
    for (int i = 0; i < n - 1; i++) {
        scanf("%d", &arr[i]);
        sum += arr[i];
    }

    expected_sum = n * (n + 1) / 2;
    missing = expected_sum - sum;
```

```
Enter the value of n (total elements including missing one): 5
Enter 4 elements (from 1 to 5, one missing):
1 2 3 5
The missing element is: 4


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

20. Array concatenation by using c programming language

Aim:To write a C program to **concatenate two arrays** and display the final merged array.



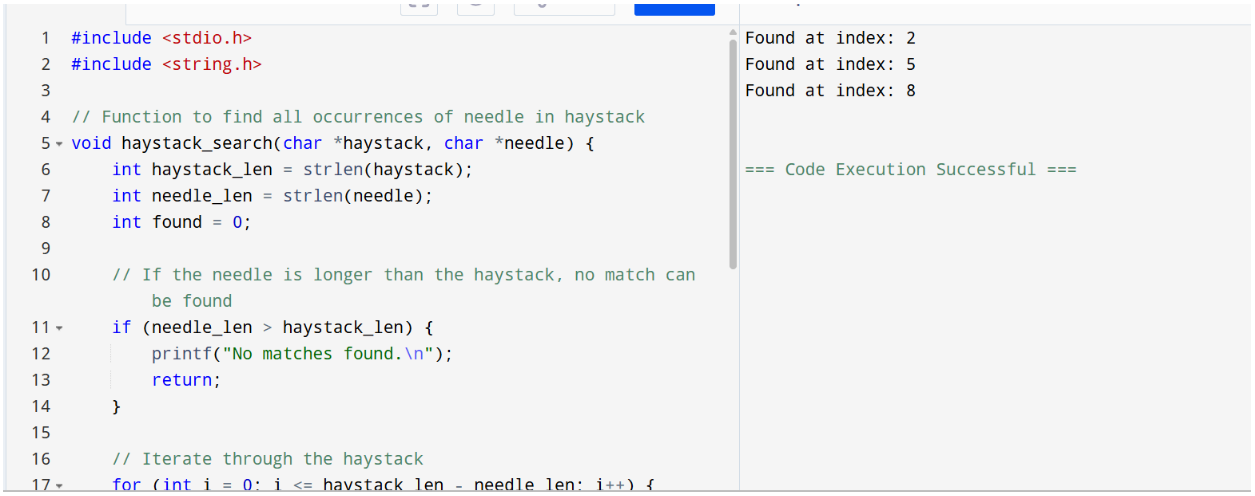
```
#include <stdio.h>

int main() {
    int arr1[100], arr2[100], arr3[200];
    int n1, n2, i, j;

    // Input first array
    printf("Enter size of first array: ");
    scanf("%d", &n1);
    printf("Enter %d elements for first array:\n", n1);
    for (i = 0; i < n1; i++) {
        scanf("%d", &arr1[i]);
    }

    // Input second array
    printf("Enter size of second array: ");
    scanf("%d", &n2);
    printf("Enter %d elements for second array:\n", n2);
```

```
Enter size of first array: 3
Enter 3 elements for first array:
1 2 3
Enter size of second array: 4
Enter 4 elements for second array:
4 5 6 7
Concatenated array is:
1 2 3 4 5 6 7


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

21. Haystack by using c programming language

Aim:The goal of the Haystack algorithm is to **find all occurrences** of a **needle** (a substring) within a **haystack** (a larger string). In other words, it helps to search a small string (needle) in a larger string (haystack) and return the index/indices where the needle is found.



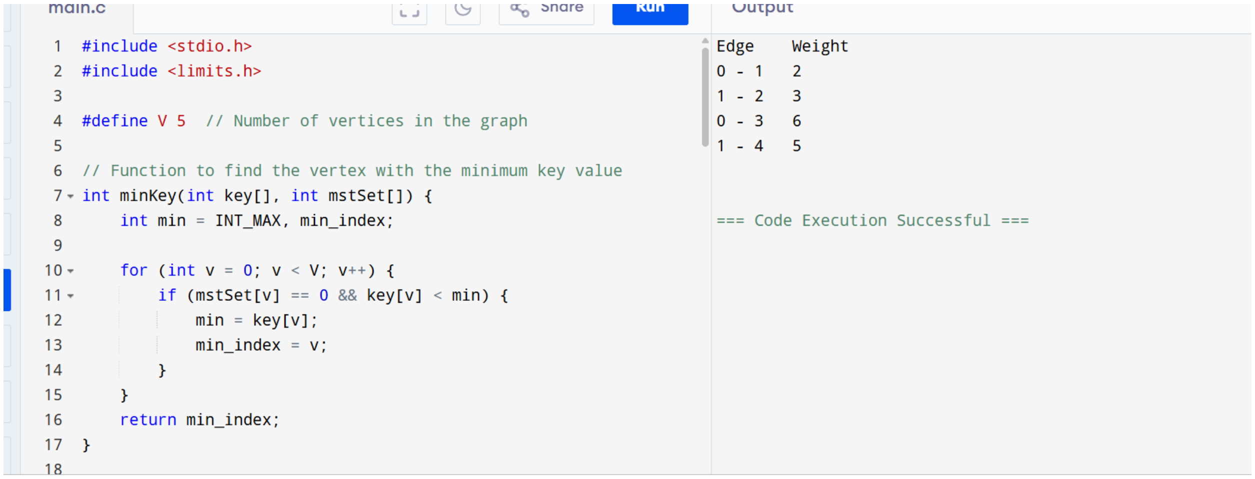
```c
#include <stdio.h>
#include <string.h>

// Function to find all occurrences of needle in haystack
void haystack_search(char *haystack, char *needle) {
    int haystack_len = strlen(haystack);
    int needle_len = strlen(needle);
    int found = 0;

    // If the needle is longer than the haystack, no match can be found
    if (needle_len > haystack_len) {
        printf("No matches found.\n");
        return;
    }

    // Iterate through the haystack
    for (int i = 0; i <= haystack_len - needle_len; i++) {
```

```
Found at index: 2
Found at index: 5
Found at index: 8


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

22. Given Graph convert to array and print minimum edges by using c programming

Aim:The aim of this program is to **implement Prim's Algorithm** to **find the Minimum Spanning Tree (MST)** of a graph represented as an adjacency matrix



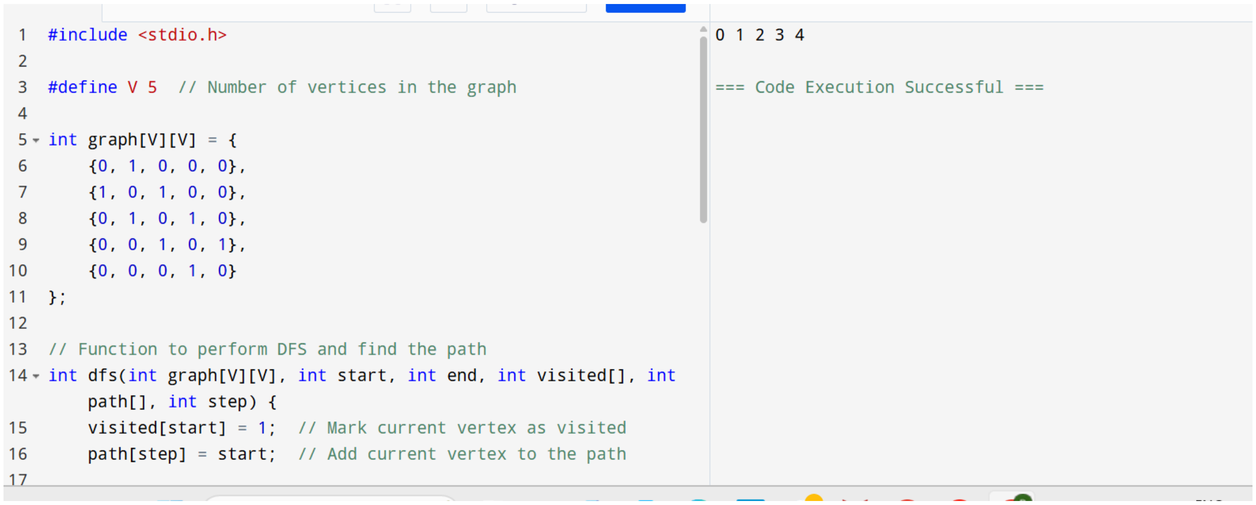
```c
#include <stdio.h>
#include <limits.h>

#define V 5  // Number of vertices in the graph

// Function to find the vertex with the minimum key value
int minKey(int key[], int mstSet[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++) {
        if (mstSet[v] == 0 && key[v] < min) {
            min = key[v];
            min_index = v;
        }
    }
    return min_index;
}

```

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

23. Given Graph - Print valid path by using c programming

Aim:The aim of this program is to **find and print a valid path** between two given vertices in a graph.

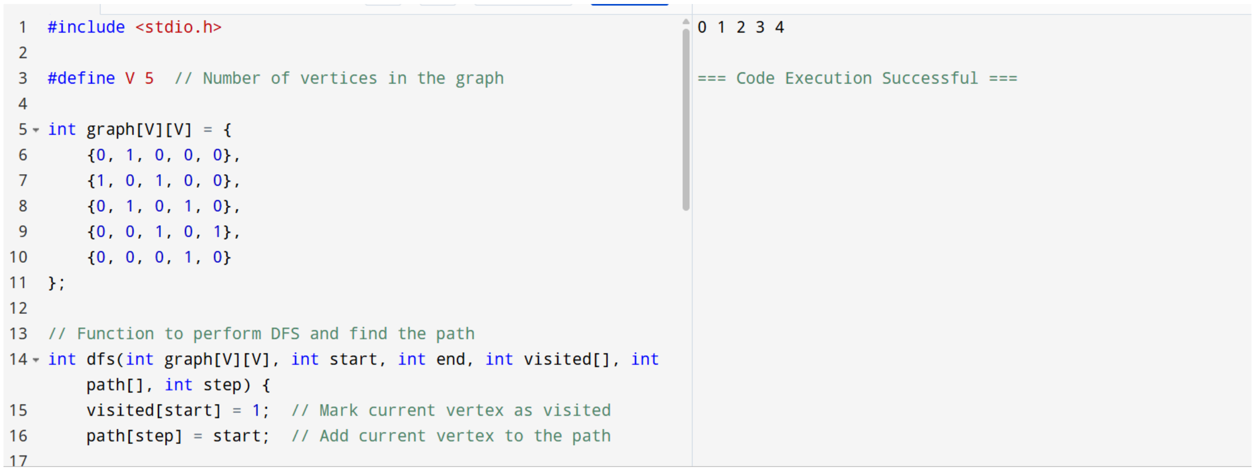


```c
#include <stdio.h>

#define V 5  // Number of vertices in the graph

int graph[V][V] = {
    {0, 1, 0, 0, 0},
    {1, 0, 1, 0, 0},
    {0, 1, 0, 1, 0},
    {0, 0, 1, 0, 1},
    {0, 0, 0, 1, 0}
};

// Function to perform DFS and find the path
int dfs(int graph[V][V], int start, int end, int visited[], int
    path[], int step) {
    visited[start] = 1;  // Mark current vertex as visited
    path[step] = start;  // Add current vertex to the path
```

```
0 1 2 3 4

=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

24, heap , merge, insertion and quick sort by using c programming language

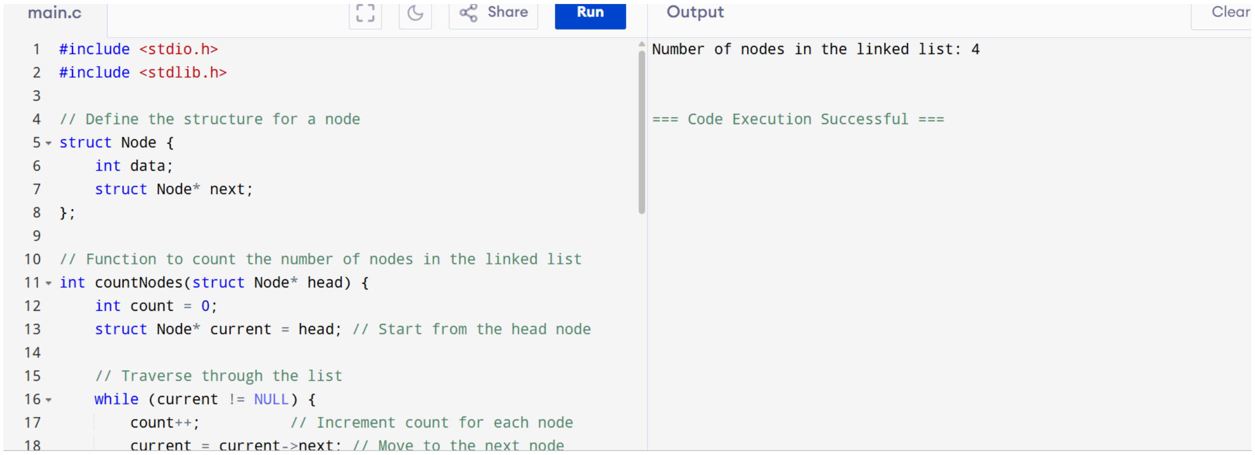


```c
#include <stdio.h>

#define V 5  // Number of vertices in the graph

int graph[V][V] = {
    {0, 1, 0, 0, 0},
    {1, 0, 1, 0, 0},
    {0, 1, 0, 1, 0},
    {0, 0, 1, 0, 1},
    {0, 0, 0, 1, 0}
};

// Function to perform DFS and find the path
int dfs(int graph[V][V], int start, int end, int visited[], int
    path[], int step) {
    visited[start] = 1;  // Mark current vertex as visited
    path[step] = start;  // Add current vertex to the path
```

```
0 1 2 3 4

=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

25. Print no of nodes in the given linked list by using c programming
Aim:The aim of this program is to **count and print the number of nodes** in a given singly linked list.



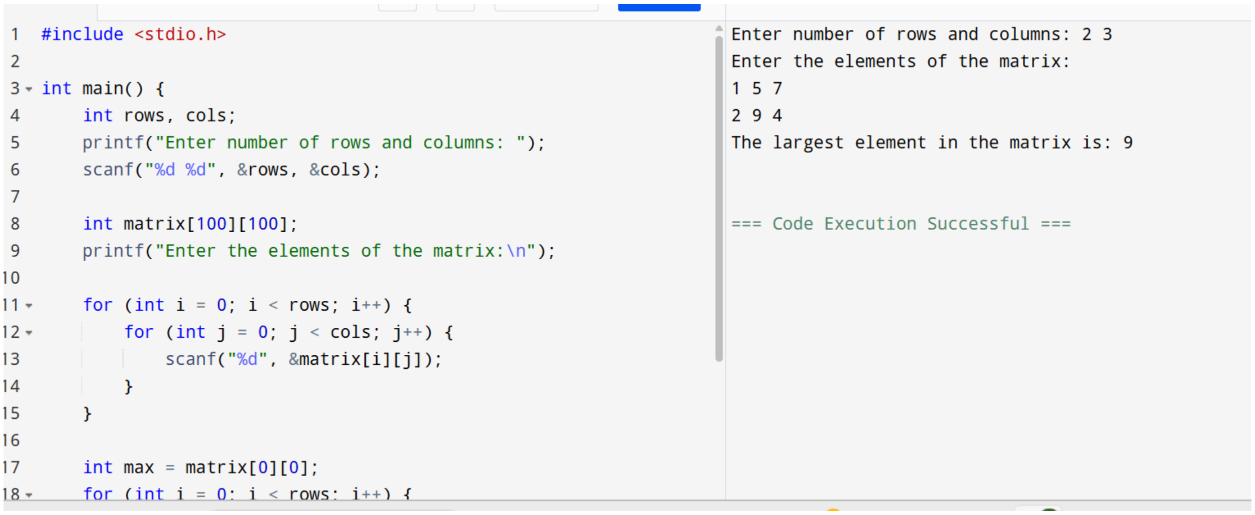
```c
#include <stdio.h>
#include <stdlib.h>

// Define the structure for a node
struct Node {
    int data;
    struct Node* next;
};

// Function to count the number of nodes in the linked list
int countNodes(struct Node* head) {
    int count = 0;
    struct Node* current = head; // Start from the head node

    // Traverse through the list
    while (current != NULL) {
        count++;          // Increment count for each node
        current = current->next; // Move to the next node
```

```
Number of nodes in the linked list: 4


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

26. Given 2 D matrix print largest element by using c programming

Aim:To **find and print the largest element** in a given 2D matrix by traversing all its elements and tracking the maximum value.



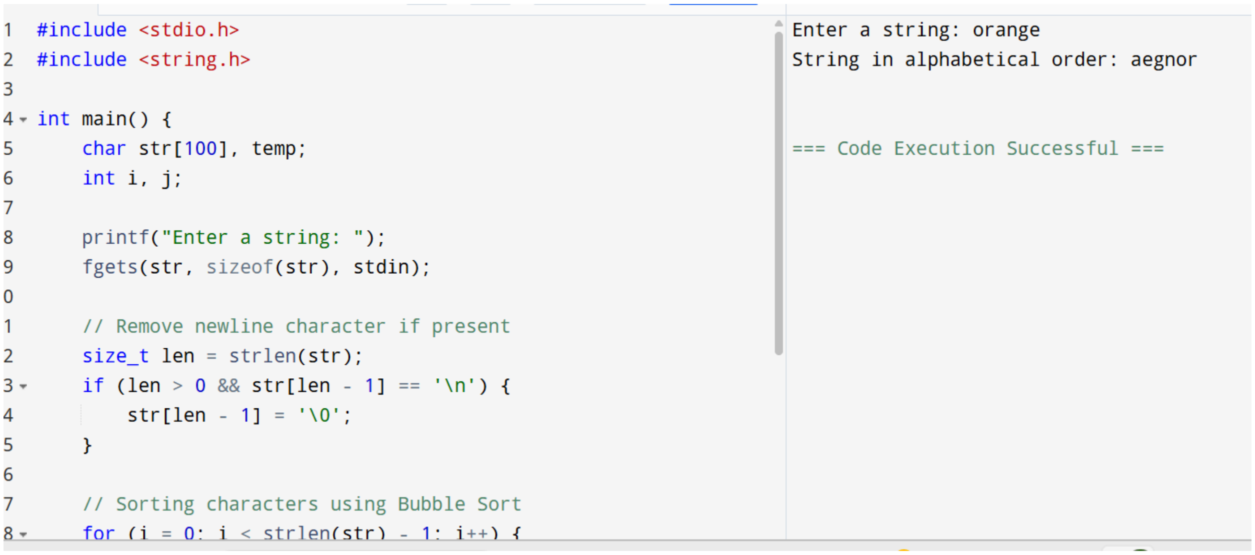
```c
#include <stdio.h>

int main() {
    int rows, cols;
    printf("Enter number of rows and columns: ");
    scanf("%d %d", &rows, &cols);

    int matrix[100][100];
    printf("Enter the elements of the matrix:\n");

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    int max = matrix[0][0];
    for (int i = 0; i < rows; i++) {
```

```
Enter number of rows and columns: 2 3
Enter the elements of the matrix:
1 5 7
2 9 4
The largest element in the matrix is: 9


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

27. Given a string - sort in alphabetical order by using c program

Aim:To write a C program to sort the characters of a given string in **alphabetical order** (ascending lexicographical order).



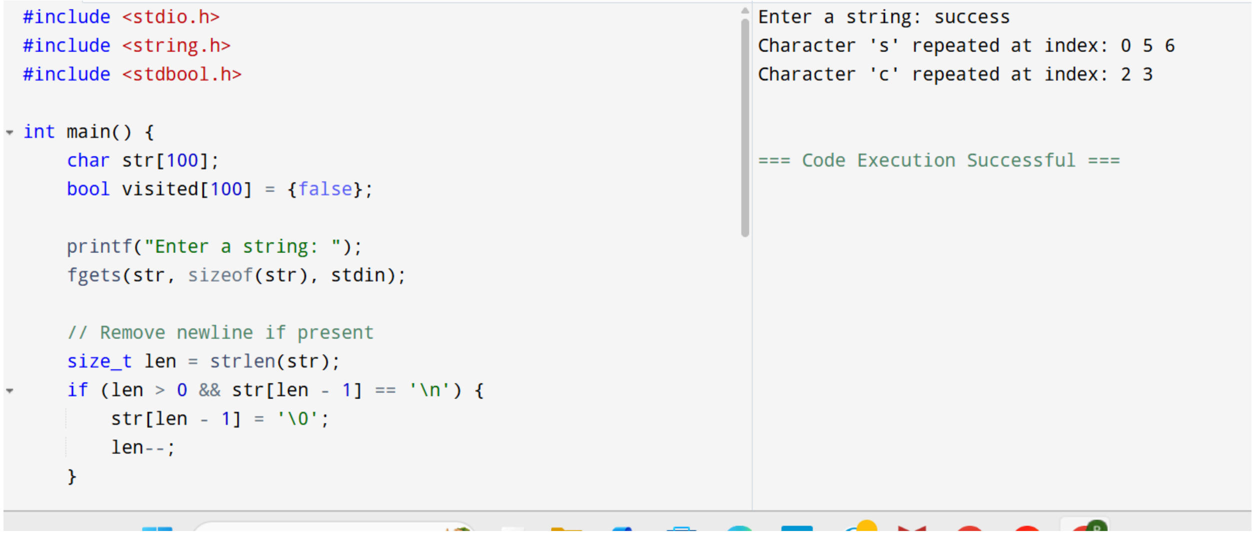
```c
#include <stdio.h>
#include <string.h>

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

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    // Remove newline character if present
    size_t len = strlen(str);
    if (len > 0 && str[len - 1] == '\n') {
        str[len - 1] = '\0';
    }

    // Sorting characters using Bubble Sort
    for (i = 0; i < strlen(str) - 1; i++) {
```

```
Enter a string: orange
String in alphabetical order: aegnor


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

28. Print the index of repeated characters given in an array

Aim:To write a C program that prints the **indexes of repeated characters** in a character array (string).



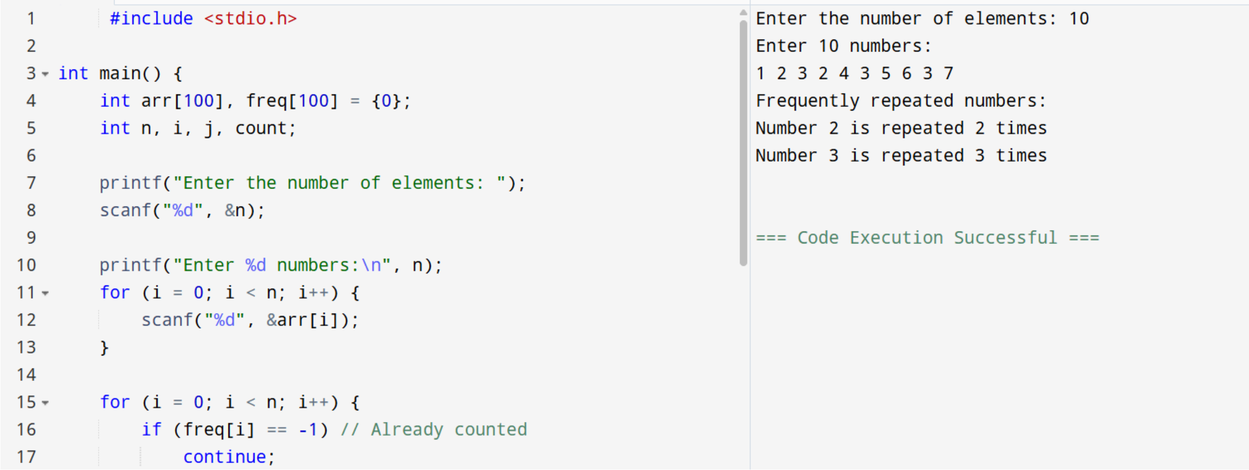
```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

int main() {
    char str[100];
    bool visited[100] = {false};

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    // Remove newline if present
    size_t len = strlen(str);
    if (len > 0 && str[len - 1] == '\n') {
        str[len - 1] = '\0';
        len--;
    }
```

```
Enter a string: success
Character 's' repeated at index: 0 5 6
Character 'c' repeated at index: 2 3


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

29. Print the frequently repeated numbers count from an array

Aim;To write a C program that counts and prints how many times each number is **repeated** in an integer array.



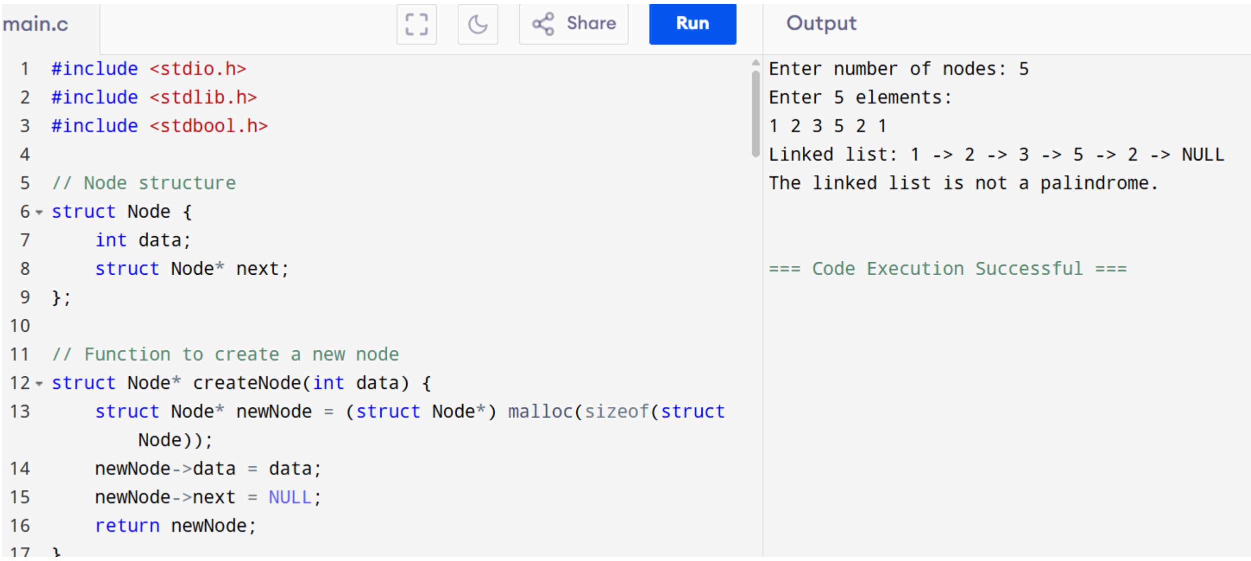
```c
#include <stdio.h>

int main() {
    int arr[100], freq[100] = {0};
    int n, i, j, count;

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

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

    for (i = 0; i < n; i++) {
        if (freq[i] == -1) // Already counted
            continue;
```

```
Enter the number of elements: 10
Enter 10 numbers:
1 2 3 2 4 3 5 6 3 7
Frequently repeated numbers:
Number 2 is repeated 2 times
Number 3 is repeated 3 times


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

30. Palindrome using SLL

Aim:To write a C program that checks whether a singly linked list is a **palindrome**.



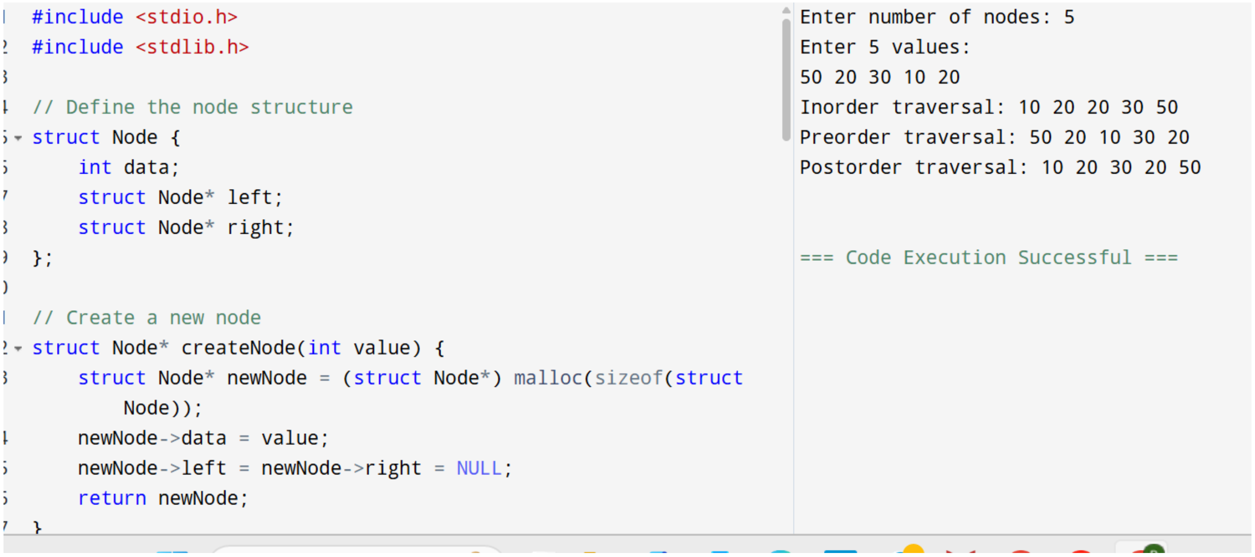
```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Node structure
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

```
Enter number of nodes: 5
Enter 5 elements:
1 2 3 5 2 1
Linked list: 1 -> 2 -> 3 -> 5 -> 2 -> NULL
The linked list is not a palindrome.


=== Code Execution Successful ===
```

Result: code executed successfully and output is verified

31. Binary tree by using c programming language

Aim:To implement a basic **Binary Tree** in C with the following operations:

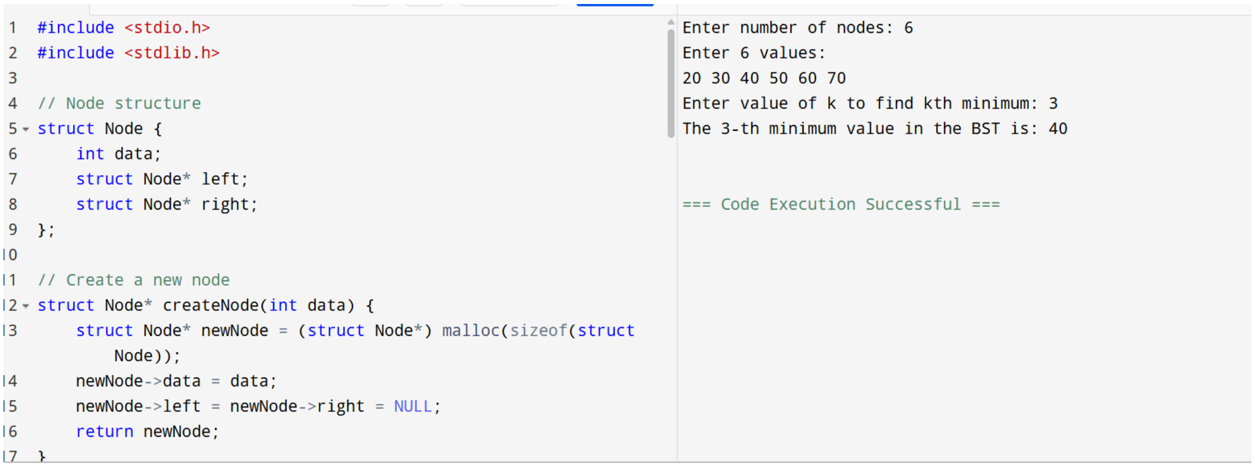


```c
#include <stdio.h>
#include <stdlib.h>

// Define the node structure
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct
        Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
```

```
Enter number of nodes: 5
Enter 5 values:
50 20 30 10 20
Inorder traversal: 10 20 20 30 50
Preorder traversal: 50 20 10 30 20
Postorder traversal: 10 20 30 20 50


=== Code Execution Successful ===
```

32. BST - kth min value by using c programming language

Aim:To implement a program in C to Construct a **Binary Search Tree (BST)**



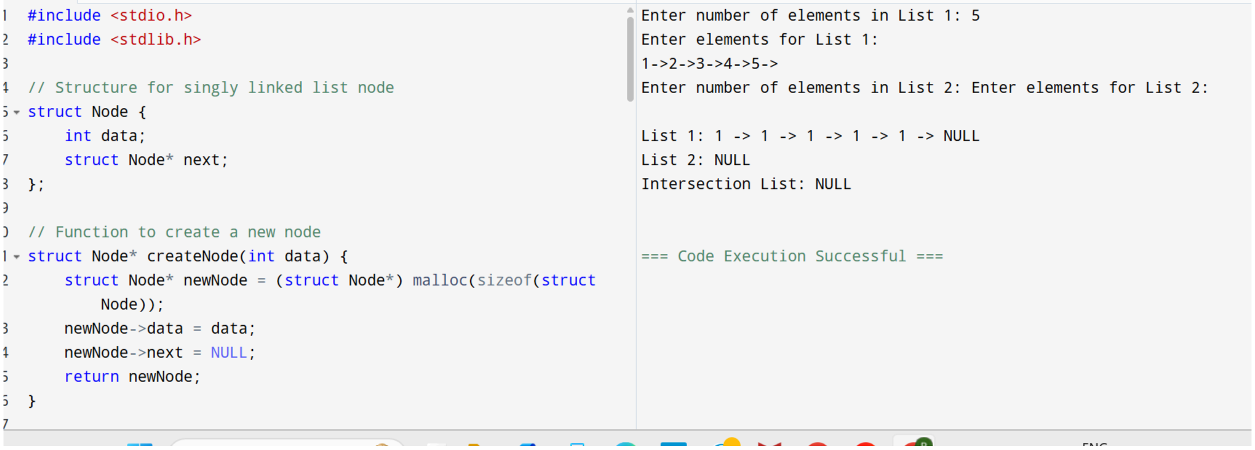
```c
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct
        Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
```

```
Enter number of nodes: 6
Enter 6 values:
20 30 40 50 60 70
Enter value of k to find kth minimum: 3
The 3-th minimum value in the BST is: 40


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

33. Intersect SLL by using c programming

Aim: To implement a C program to find the **intersection of two singly linked lists (SLLs)**, where intersection means elements **common** to both lists (based on data values).



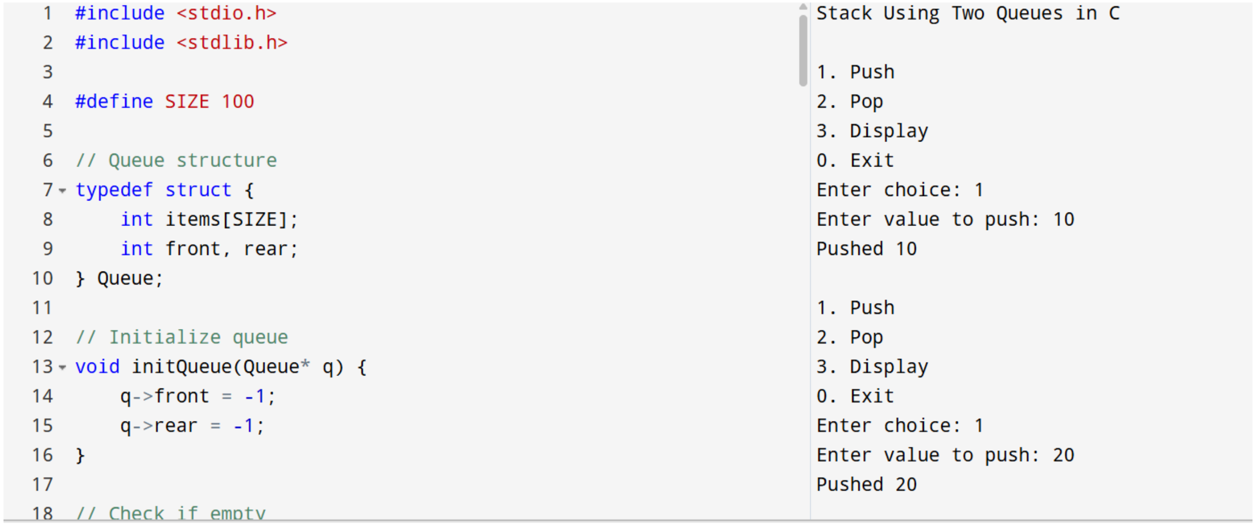
```c
#include <stdio.h>
#include <stdlib.h>

// Structure for singly linked list node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct
        Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

```
Enter number of elements in List 1: 5
Enter elements for List 1:
1->2->3->4->5->
Enter number of elements in List 2: Enter elements for List 2:

List 1: 1 -> 1 -> 1 -> 1 -> 1 -> NULL
List 2: NULL
Intersection List: NULL


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified

34, stack using two queues by using c programming language

Aim: To implement a **Stack using two Queues** in C, supporting `push`, `pop`, and `display` operations.The objective is to simulate **Last In First Out (LIFO)** behavior of a stack using two **First In First Out (FIFO)** queues.



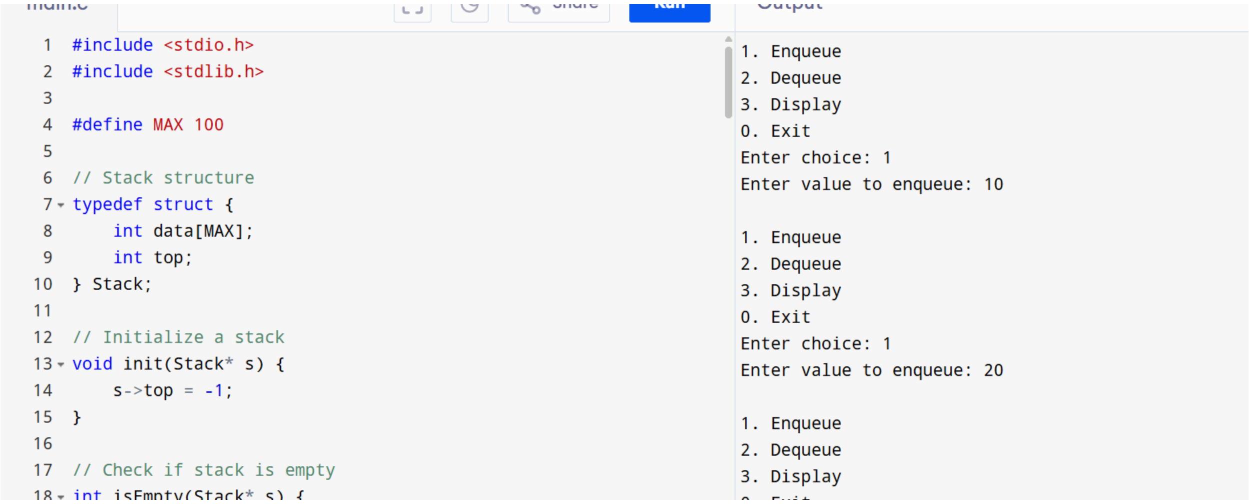
```c
#include <stdio.h>
#include <stdlib.h>

#define SIZE 100

// Queue structure
typedef struct {
    int items[SIZE];
    int front, rear;
} Queue;

// Initialize queue
void initQueue(Queue* q) {
    q->front = -1;
    q->rear = -1;
}

// Check if empty
```

```
Stack Using Two Queues in C

1. Push
2. Pop
3. Display
0. Exit
Enter choice: 1
Enter value to push: 10
Pushed 10

1. Push
2. Pop
3. Display
0. Exit
Enter choice: 1
Enter value to push: 20
Pushed 20
```

Result:code executed successfully and output is verified.

35, queue using two stacks by using c programming

Aim:To implement a **queue using two stacks** in C, supporting enqueue and dequeue operations, and demonstrating how queue operations (FIFO) can be implemented using stack operations (LIFO).

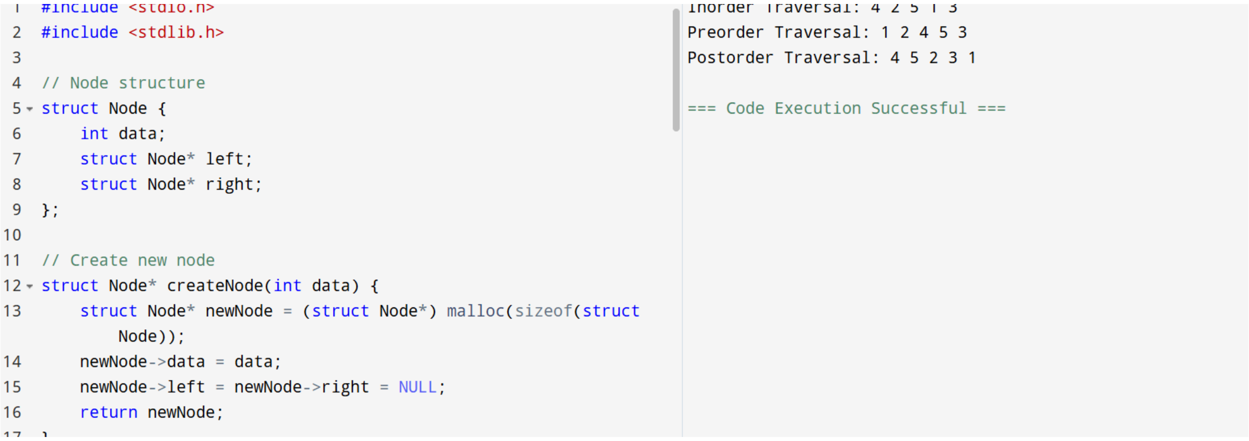


```c
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

// Stack structure
typedef struct {
    int data[MAX];
    int top;
} Stack;

// Initialize a stack
void init(Stack* s) {
    s->top = -1;
}

// Check if stack is empty
int isEmpty(Stack* s) {
```

```
1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter choice: 1
Enter value to enqueue: 10

1. Enqueue
2. Dequeue
3. Display
0. Exit
Enter choice: 1
Enter value to enqueue: 20

1. Enqueue
2. Dequeue
3. Display
0. Exit
```

36. Tree traverse by using c programming language

Aim: To implement tree traversal methods in C, namelyInorder Traversal (Left, Root, Right), Preorder Traversal (Root, Left, Right),Postorder Traversal (Left, Right, Root)



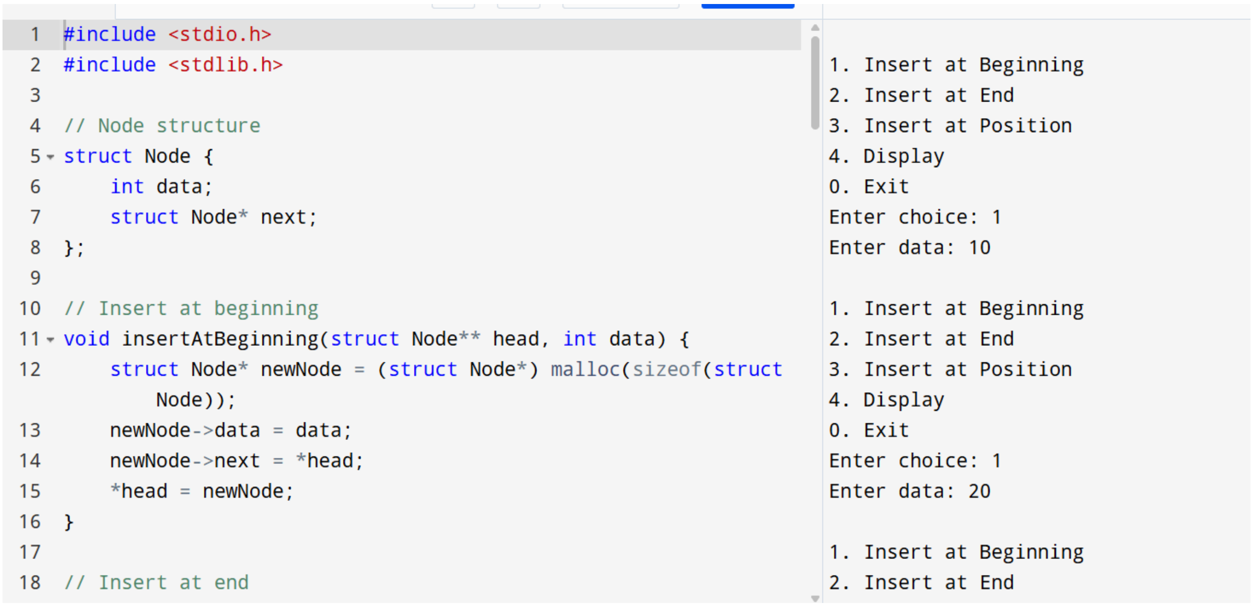
```c
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Create new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct
        Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
```

```
Inorder Traversal: 4 2 5 1 3
Preorder Traversal: 1 2 4 5 3
Postorder Traversal: 4 5 2 3 1

=== Code Execution Successful ===
```

Result:code executed successfully and output is verified.

37 linked list - Insertion by using c programm

Aim:To implement insertion operations in a **singly linked list** in C, including Insertion at the **beginning** Insertion at the **end** Insertion at a **given position**



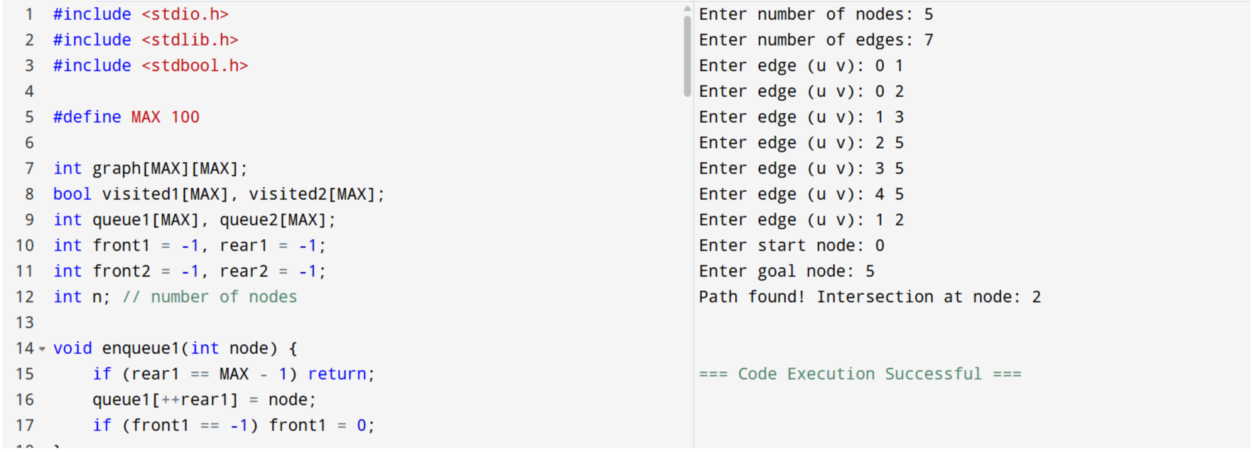
```c
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node* next;
};

// Insert at beginning
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct
        Node));
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
}

// Insert at end
```

```
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
0. Exit
Enter choice: 1
Enter data: 10

1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
0. Exit
Enter choice: 1
Enter data: 20

1. Insert at Beginning
2. Insert at End
```

Result:code executed successfully and output is verified.

38.Bidirectional by using c program

Aim:To implement **Bidirectional Search** using C language for traversing or searching a graph to find the shortest path between a source and destination node



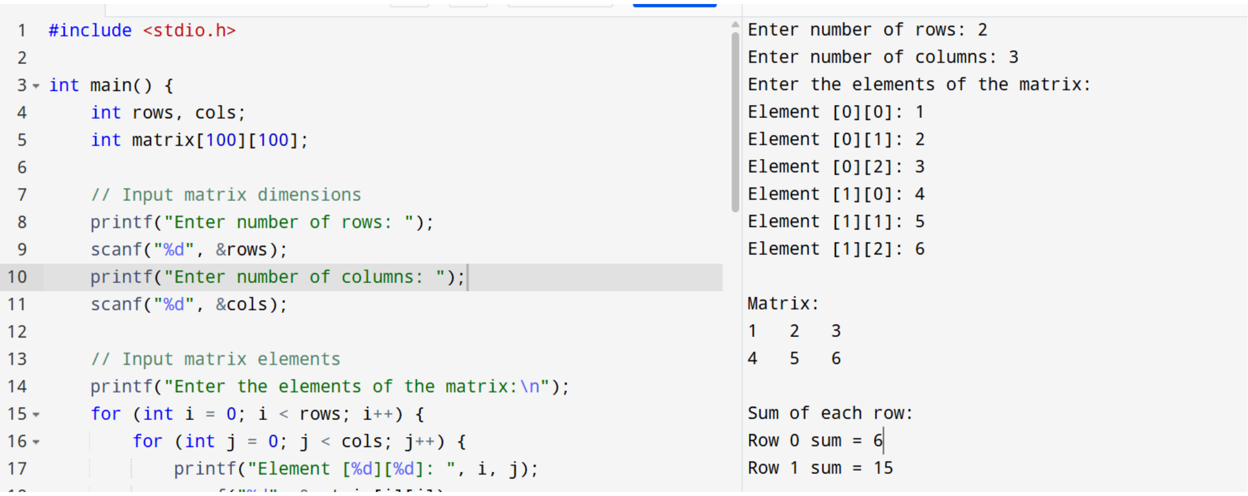
```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX 100

int graph[MAX][MAX];
bool visited1[MAX], visited2[MAX];
int queue1[MAX], queue2[MAX];
int front1 = -1, rear1 = -1;
int front2 = -1, rear2 = -1;
int n; // number of nodes

void enqueue1(int node) {
    if (rear1 == MAX - 1) return;
    queue1[++rear1] = node;
    if (front1 == -1) front1 = 0;
```

```
Enter number of nodes: 5
Enter number of edges: 7
Enter edge (u v): 0 1
Enter edge (u v): 0 2
Enter edge (u v): 1 3
Enter edge (u v): 2 5
Enter edge (u v): 3 5
Enter edge (u v): 4 5
Enter edge (u v): 1 2
Enter start node: 0
Enter goal node: 5
Path found! Intersection at node: 2


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified.

39. Sum of row and column - Array in c program

Aim:To write a C program that accepts a **2D array (matrix)** as input and calculates the **sum of each row and each column** separately.



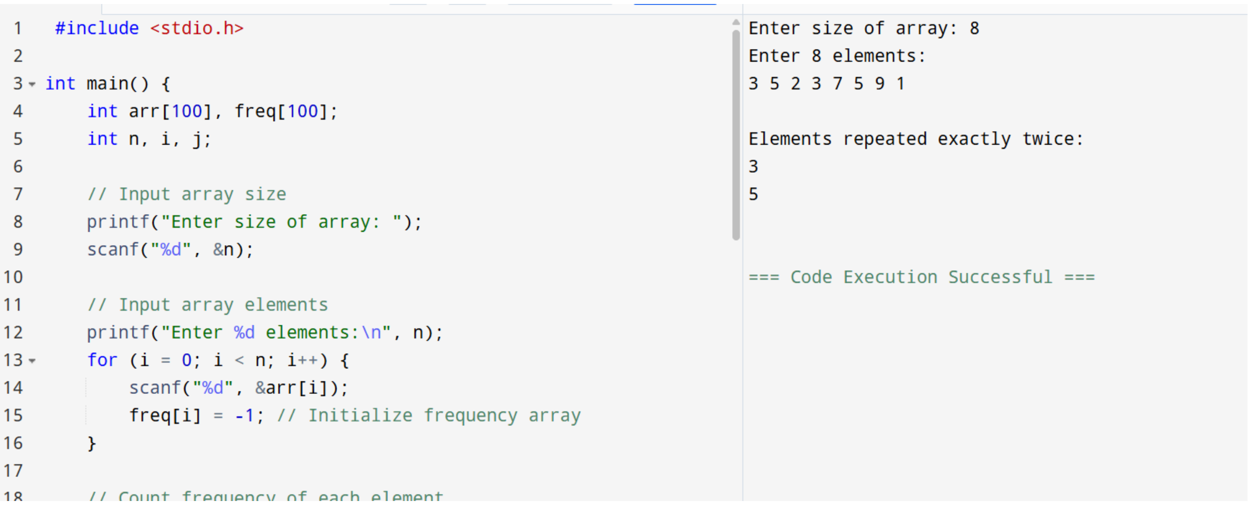
```c
#include <stdio.h>

int main() {
    int rows, cols;
    int matrix[100][100];

    // Input matrix dimensions
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    printf("Enter number of columns: ");
    scanf("%d", &cols);

    // Input matrix elements
    printf("Enter the elements of the matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("Element [%d][%d]: ", i, j);
```

```
Enter number of rows: 2
Enter number of columns: 3
Enter the elements of the matrix:
Element [0][0]: 1
Element [0][1]: 2
Element [0][2]: 3
Element [1][0]: 4
Element [1][1]: 5
Element [1][2]: 6

Matrix:
1   2   3
4   5   6

Sum of each row:
Row 0 sum = 6
Row 1 sum = 15
```

Result:code executed successfully and output is verified.

40. Elements repeated twice - Array in c programming

Aim:To write a C program that identifies and displays elements in a **1D array** that are **repeated exactly twice**.



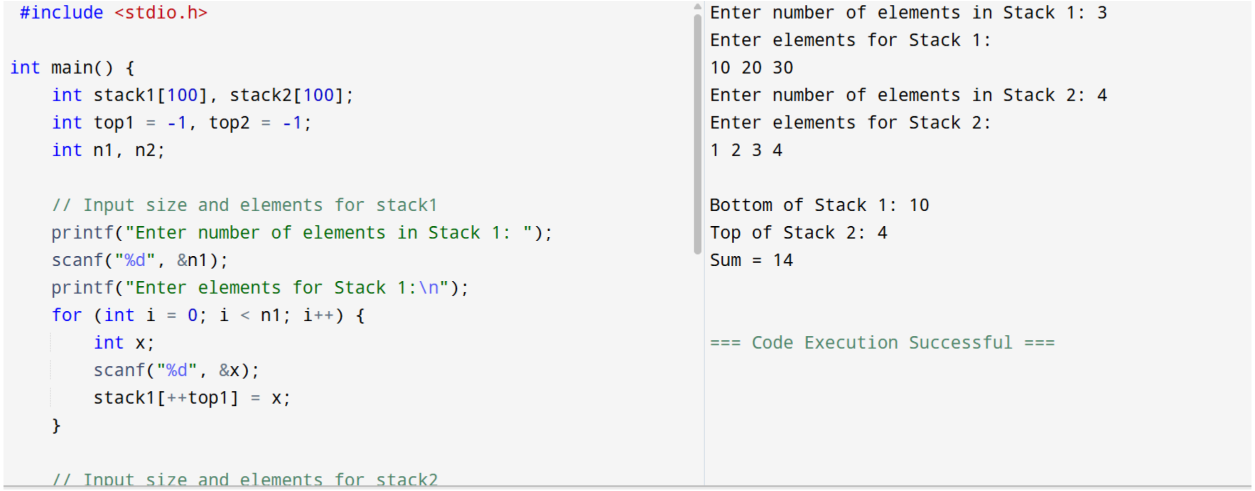
```c
 #include <stdio.h>

int main() {
    int arr[100], freq[100];
    int n, i, j;

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

    // Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
        freq[i] = -1; // Initialize frequency array
    }

    // Count frequency of each element
```

```
Enter size of array: 8
Enter 8 elements:
3 5 2 3 7 5 9 1

Elements repeated exactly twice:
3
5


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified.

41. Consider 2 stacks, add bottom most element and top most element print the
value

To write a C program that takes input for **two stacks,**Finds and adds the **bottom-most** element of the first stack and the **top-most** element of the second stack and Prints the result



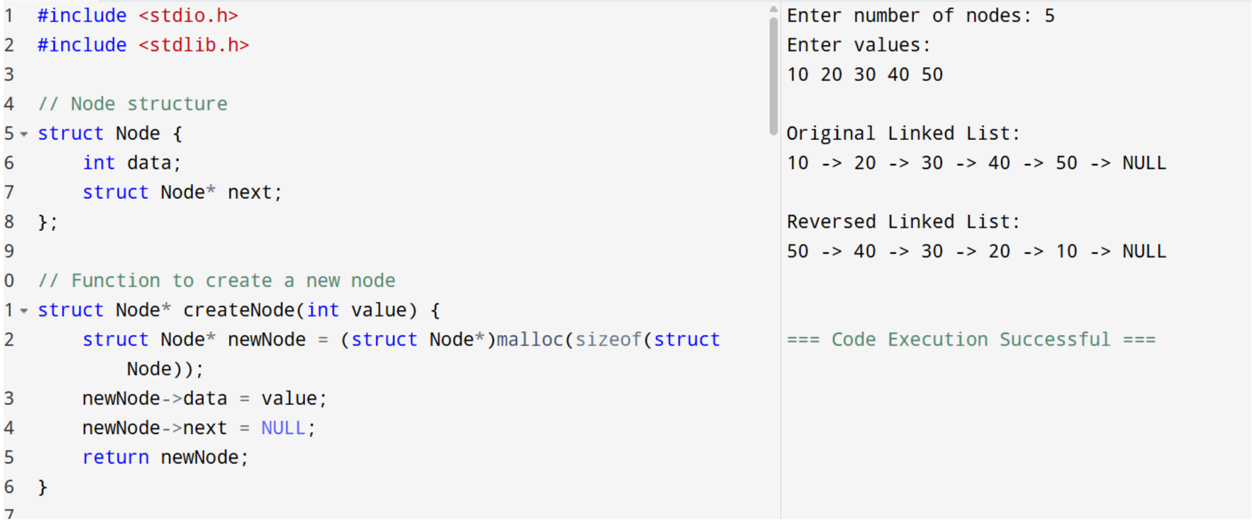
```c
#include <stdio.h>

int main() {
    int stack1[100], stack2[100];
    int top1 = -1, top2 = -1;
    int n1, n2;

    // Input size and elements for stack1
    printf("Enter number of elements in Stack 1: ");
    scanf("%d", &n1);
    printf("Enter elements for Stack 1:\n");
    for (int i = 0; i < n1; i++) {
        int x;
        scanf("%d", &x);
        stack1[++top1] = x;
    }

    // Input size and elements for stack2
```

```
Enter number of elements in Stack 1: 3
Enter elements for Stack 1:
10 20 30
Enter number of elements in Stack 2: 4
Enter elements for Stack 2:
1 2 3 4

Bottom of Stack 1: 10
Top of Stack 2: 4
Sum = 14


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified.

42. Reverse - SLL using c programming language

To write a **C program** to **reverse a singly linked list** (SLL) and display the reversed list.



```c
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
        Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}
```

```
Enter number of nodes: 5
Enter values:
10 20 30 40 50

Original Linked List:
10 -> 20 -> 30 -> 40 -> 50 -> NULL

Reversed Linked List:
50 -> 40 -> 30 -> 20 -> 10 -> NULL


=== Code Execution Successful ===
```

Result:code executed successfully and output is verified.