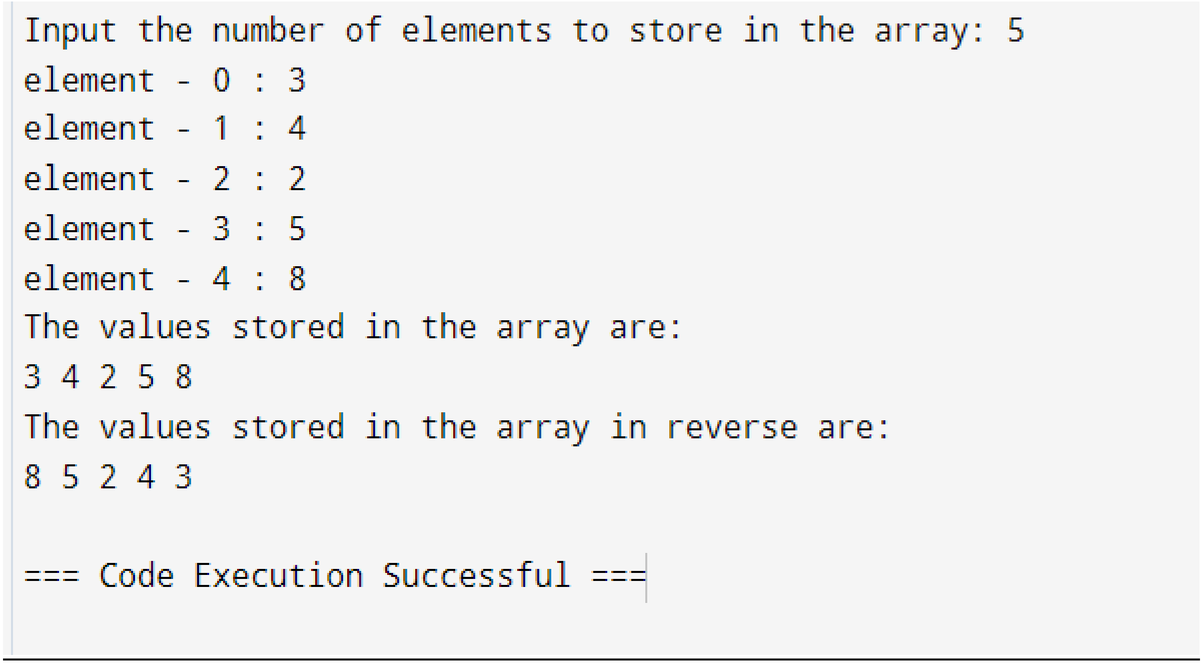
<u>28/8/24</u>

<u>1.</u>

```
#include <stdio.h>

int main() {
   int n, i;
   printf("Input the number of elements to store in the array: ");
   scanf("%d", &n);
   int arr[n];

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

   printf("The values stored in the array are:\n");
   for(i = 0; i < n; i++)
      printf("%d ", arr[i]);

   printf("\nThe values stored in the array in reverse are:\n");
   for(i = n - 1; i >= 0; i--)
      printf("%d ", arr[i]);

   return 0;
}
```

<u>OUTPUT:</u>



```
Input the number of elements to store in the array: 5
element - 0 : 3
element - 1 : 4
element - 2 : 2
element - 3 : 5
element - 4 : 8
The values stored in the array are:
3 4 2 5 8
The values stored in the array in reverse are:
8 5 2 4 3

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

2.

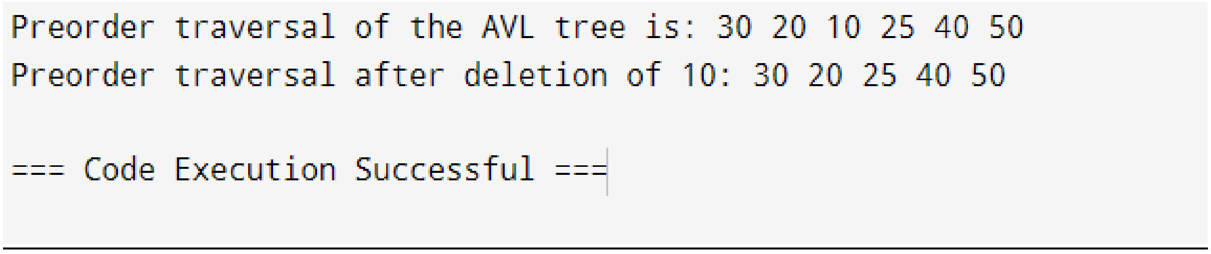
```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int key, height;
    struct Node *left, *right;
} Node;

int height(Node *N) {
    return N ? N->height : 0;
}

int max(int a, int b) {
    return (a > b) ? a : b;
}

Node* newNode(int key) {
    Node* node = (Node*)malloc(sizeof(Node));
    node->key = key;
    node->left = node->right = NULL;
    node->height = 1;
    return node;
}

Node *rightRotate(Node *y) {
    Node *x = y->left;
    Node *T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;
    return x;
}

Node *leftRotate(Node *x) {
    Node *y = x->right;
    Node *T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
    return y;
}

int getBalance(Node *N) {
    return N ? height(N->left) - height(N->right) : 0;
}
```

```
Node* insert(Node* node, int key) {
    if (!node) return newNode(key);
    if (key < node->key) node->left = insert(node->left, key);
    else if (key > node->key) node->right = insert(node->right, key);
    else return node;

    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);

    if (balance > 1 && key < node->left->key) return rightRotate(node);
    if (balance < -1 && key > node->right->key) return leftRotate(node);
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}

Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left) current = current->left;
    return current;
}

Node* deleteNode(Node* root, int key) {
```

```
if (!root) return root;
if (key < root->key) root->left = deleteNode(root->left, key);
else if (key > root->key) root->right = deleteNode(root->right, key);
else {
    if (!root->left || !root->right) {
        Node *temp = root->left ? root->left : root->right;
        if (!temp) {
            free(root);
            return NULL;
        } else {
            *root = *temp;
            free(temp);
        }
    } else {
        Node* temp = minValueNode(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }
}
if (!root) return root;
root->height = 1 + max(height(root->left), height(root->right));
int balance = getBalance(root);
if (balance > 1 && getBalance(root->left) >= 0) return rightRotate(root);
if (balance > 1 && getBalance(root->left) < 0) {
    root->left = leftRotate(root->left);
    return rightRotate(root);
}
if (balance < -1 && getBalance(root->right) <= 0) return leftRotate(root);
if (balance < -1 && getBalance(root->right) > 0) {
    root->right = rightRotate(root->right);
```

```
        return leftRotate(root);
    }
    return root;
}

void preOrder(Node *root) {
    if (root) {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}

int main() {
    Node *root = NULL;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);
    printf("Preorder traversal of the AVL tree is: ");
    preOrder(root);
    root = deleteNode(root, 10);
    printf("\nPreorder traversal after deletion of 10: ");
    preOrder(root);
    return 0;
}
```

OUTPUT:

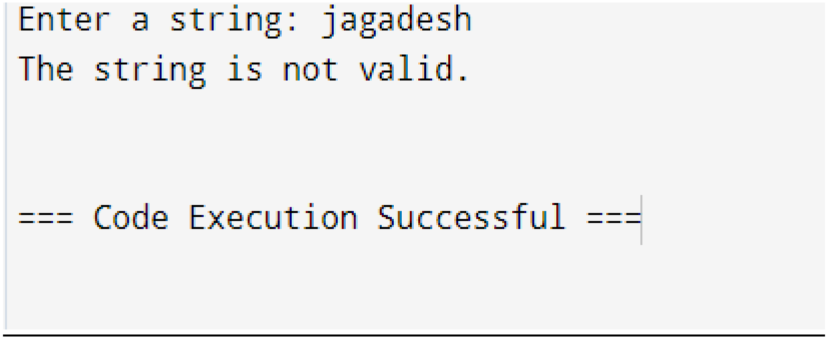


```
Preorder traversal of the AVL tree is: 30 20 10 25 40 50
Preorder traversal after deletion of 10: 30 20 25 40 50

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

3.

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

int isValidString(const char *str) {
    if (*str == '\0') return 0;
    while (*str) {
        if (!isalpha(*str++)) return 0;
    }
    return 1; // Valid string
}

int main() {
    char str[100];
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    printf("The string is %svalid.\n", isValidString(str) ? "" : "not ");
    return 0;
}
```

OUTPUT:

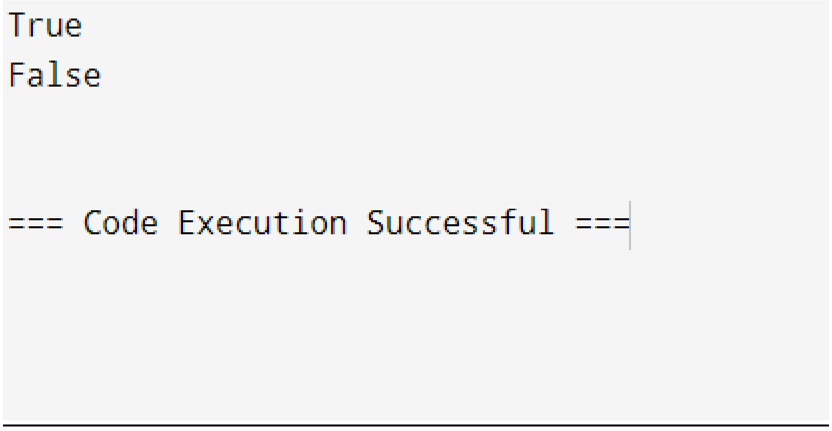


```
Enter a string: jagadesh
The string is not valid.


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

4.

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

bool validateStackSequences(int* pushed, int pushedSize, int* popped, int poppedSize) {
    int stack[pushedSize], top = -1, j = 0;
    for (int i = 0; i < pushedSize; i++) {
        stack[++top] = pushed[i];
        while (top >= 0 && stack[top] == popped[j]) {
            top--;
            j++;
        }
    }
    return top == -1;
}

int main() {
    int pushed[] = {1, 2, 3, 4, 5};
    int popped1[] = {4, 5, 3, 2, 1};
    int popped2[] = {4, 3, 5, 1, 2};

    printf("%s\n", validateStackSequences(pushed, 5, popped1, 5) ? "True" : "False");
    printf("%s\n", validateStackSequences(pushed, 5, popped2, 5) ? "True" : "False");
    return 0;
}
```

OUTPUT:

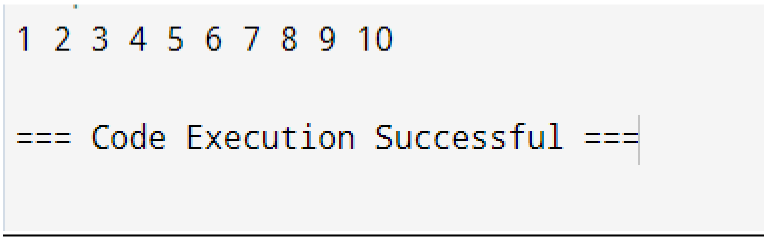


```
True
False


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

5.

```
#include <stdio.h>

int main() {
    int arr1[] = {1, 2, 3, 4, 5};
    int arr2[] = {6, 7, 8, 9, 10};
    int arr3[10];

    for (int i = 0; i < 5; i++) arr3[i] = arr1[i];
    for (int i = 0; i < 5; i++) arr3[i + 5] = arr2[i];

    for (int i = 0; i < 10; i++) printf("%d ", arr3[i]);
    return 0;
}
```

OUTPUT:



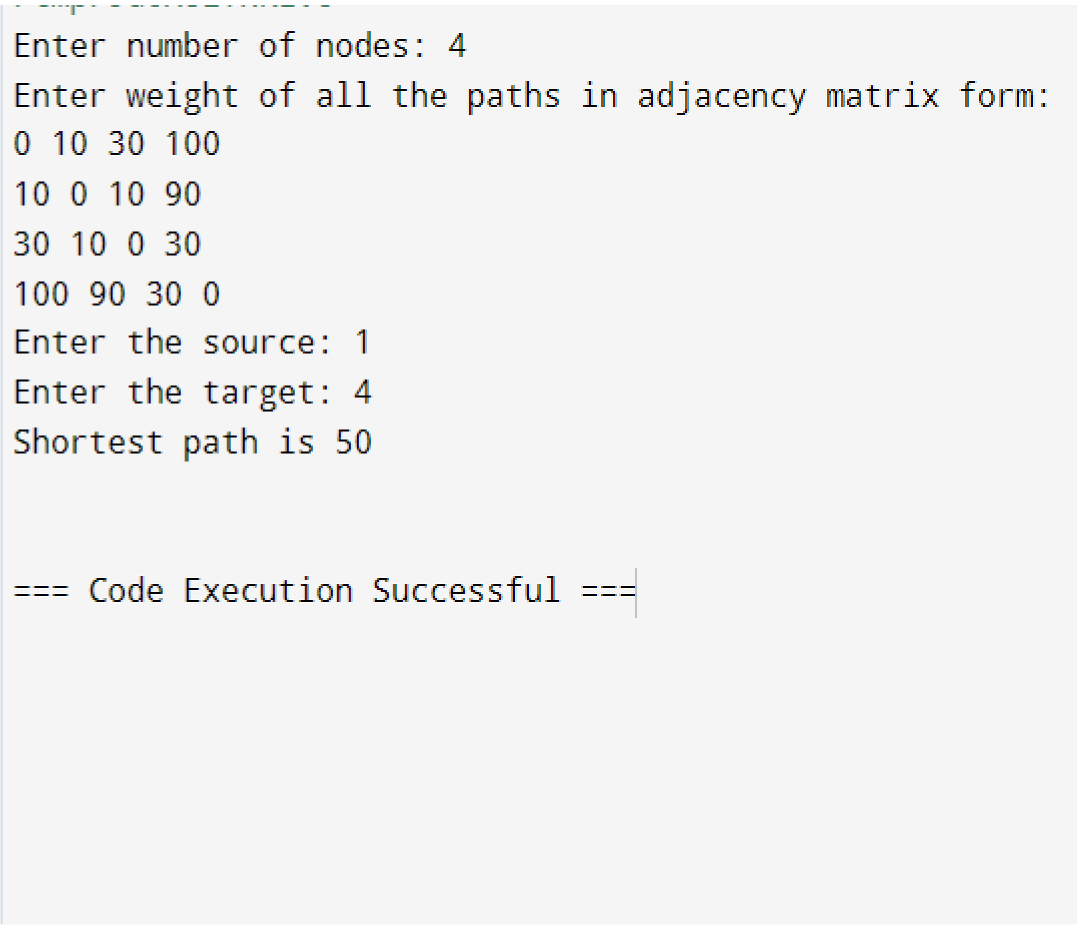
```
1 2 3 4 5 6 7 8 9 10

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

6.

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

```
#define MAX 10

int main() {
    int n, i, j, src, dest, dist[MAX], visited[MAX] = {0}, graph[MAX][MAX];
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter weight of all the paths in adjacency matrix form:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);
    printf("Enter the source: ");
    scanf("%d", &src);
    printf("Enter the target: ");
    scanf("%d", &dest);

    for (i = 0; i < n; i++) {
        dist[i] = INT_MAX;
        visited[i] = 0;
    }
    dist[src - 1] = 0;

    for (i = 0; i < n - 1; i++) {
        int min = INT_MAX, u;
        for (j = 0; j < n; j++)
            if (!visited[j] && dist[j] < min) {
                min = dist[j];
                u = j;
            }
        visited[u] = 1;
        for (j = 0; j < n; j++)
        if (!visited[j] && graph[u][j] && dist[u] + graph[u][j] < dist[j])
            dist[j] = dist[u] + graph[u][j];
    }

    printf("Shortest path is %d\n", dist[dest - 1]);
    return 0;
}
```

OUTPUT:

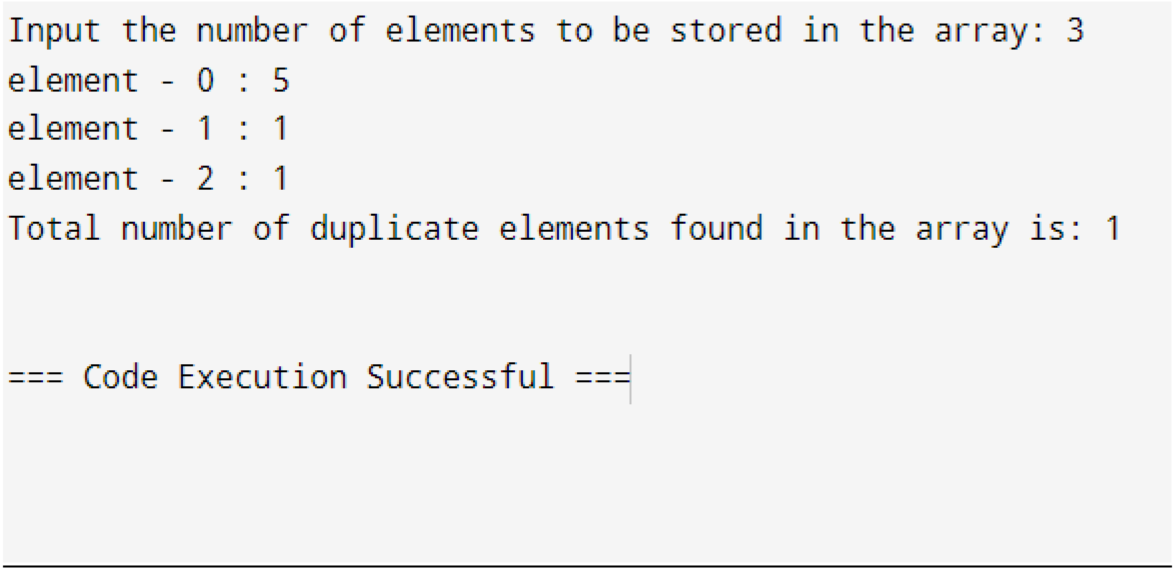


```
Enter number of nodes: 4
Enter weight of all the paths in adjacency matrix form:
0 10 30 100
10 0 10 90
30 10 0 30
100 90 30 0
Enter the source: 1
Enter the target: 4
Shortest path is 50


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

7.

```
#include <stdio.h>

int main() {
    int n, count = 0;
    printf("Input the number of elements to be stored in the array: ");
    scanf("%d", &n);
    int arr[n], freq[n];

    for (int i = 0; i < n; i++) {
        printf("element - %d : ", i);
        scanf("%d", &arr[i]);
        freq[i] = 0;
    }

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] == arr[j]) {
                freq[i]++;
            }
        }
    }

    for (int i = 0; i < n; i++) {
        if (freq[i] > 0) count++;
    }

    printf("Total number of duplicate elements found in the array is: %d\n", count);
    return 0;
}
```

OUTPUT:



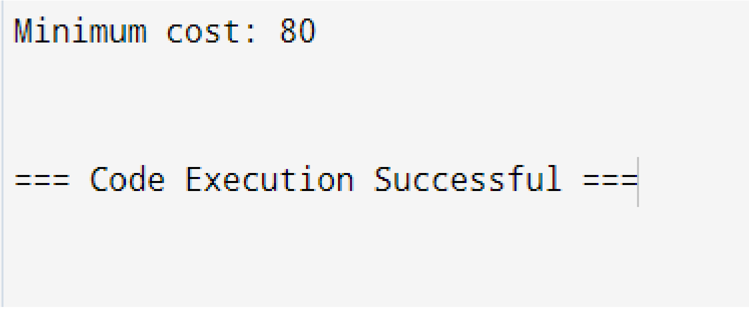
```
Input the number of elements to be stored in the array: 3
element - 0 : 5
element - 1 : 1
element - 2 : 1
Total number of duplicate elements found in the array is: 1


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

8.

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

#define CITIES 4

int tsp(int graph[CITIES][CITIES], int path[], int pos, int n, int visited[], int count, int cost, int ans) {
  if (count == n && graph[pos][0]) {
    return (cost + graph[pos][0] < ans) ? cost + graph[pos][0] : ans;
  }
  for (int i = 0; i < n; i++) {
    if (!visited[i] && graph[pos][i]) {
      visited[i] = 1;
      ans = tsp(graph, path, i, n, visited, count + 1, cost + graph[pos][i], ans);
      visited[i] = 0;
    }
  }
  return ans;
}
```

```
int main() {
    int graph[CITIES][CITIES] = { {0, 10, 15, 20}, {10, 0, 35, 25}, {15, 35, 0, 30}, {20, 25, 30, 0} };
    int visited[CITIES] = {0};
    visited[0] = 1;
    int result = tsp(graph, NULL, 0, CITIES, visited, 1, 0, INT_MAX);
    printf("Minimum cost: %d\n", result);
    return 0;
}
```

OUTPUT:

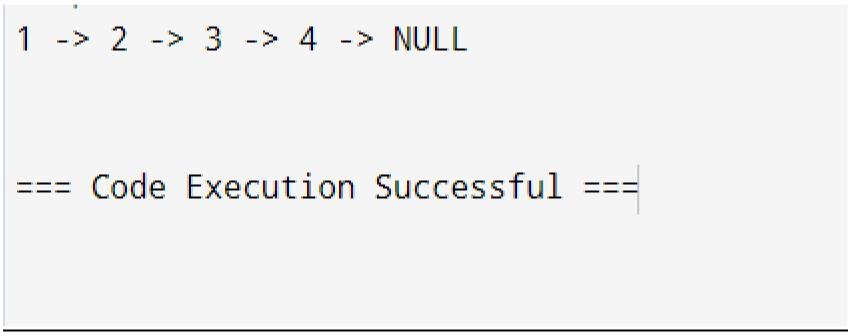


```
Minimum cost: 80


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

9.

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

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

struct Node* mergeLists(struct Node* l1, struct Node* l2) {
    if (!l1) return l2;
    if (!l2) return l1;

    struct Node* merged = NULL;
    if (l1->data < l2->data) {
        merged = l1;
        merged->next = mergeLists(l1->next, l2);
    } else {
        merged = l2;
        merged->next = mergeLists(l1, l2->next);
    }
    return merged;
}

void printList(struct Node* node) {
    while (node) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* l1 = (struct Node*)malloc(sizeof(struct Node));
    struct Node* l2 = (struct Node*)malloc(sizeof(struct Node));
    l1->data = 1; l1->next = (struct Node*)malloc(sizeof(struct Node)); l1->next->data = 3; l1->next->next = NULL;
    l2->data = 2; l2->next = (struct Node*)malloc(sizeof(struct Node)); l2->next->data = 4; l2->next->next = NULL;

    struct Node* mergedList = mergeLists(l1, l2);
    printList(mergedList);

    return 0;
}
```

<u>OUTPUT:</u>



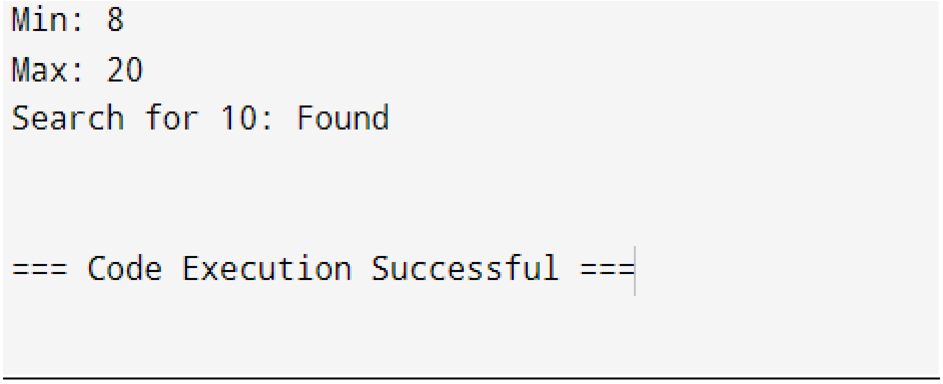
```
1 -> 2 -> 3 -> 4 -> NULL


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

10.

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

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

struct Node* insert(struct Node* node, int data) {
    if (!node) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->left = newNode->right = NULL;
        return newNode;
    }
    if (data < node->data)
        node->left = insert(node->left, data);
    else
        node->right = insert(node->right, data);
    return node;
}
```

```
struct Node* search(struct Node* root, int key) {
    if (!root || root->data == key)
        return root;
    return key < root->data ? search(root->left, key) : search(root->right, key);
}

int minValue(struct Node* node) {
    struct Node* current = node;
    while (current && current->left)
        current = current->left;
    return current->data;
}

int maxValue(struct Node* node) {
    struct Node* current = node;
    while (current && current->right)
        current = current->right;
    return current->data;
}

int main() {
    struct Node* root = NULL;
    root = insert(root, 15);
    insert(root, 10);
    insert(root, 20);
    insert(root, 8);
    insert(root, 12);

    printf("Min: %d\n", minValue(root));
    printf("Max: %d\n", maxValue(root));
    struct Node* found = search(root, 10);
    printf("Search for 10: %s\n", found ? "Found" : "Not Found");


    return 0;
}
```

OUTPUT:



```
Min: 8
Max: 20
Search for 10: Found


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