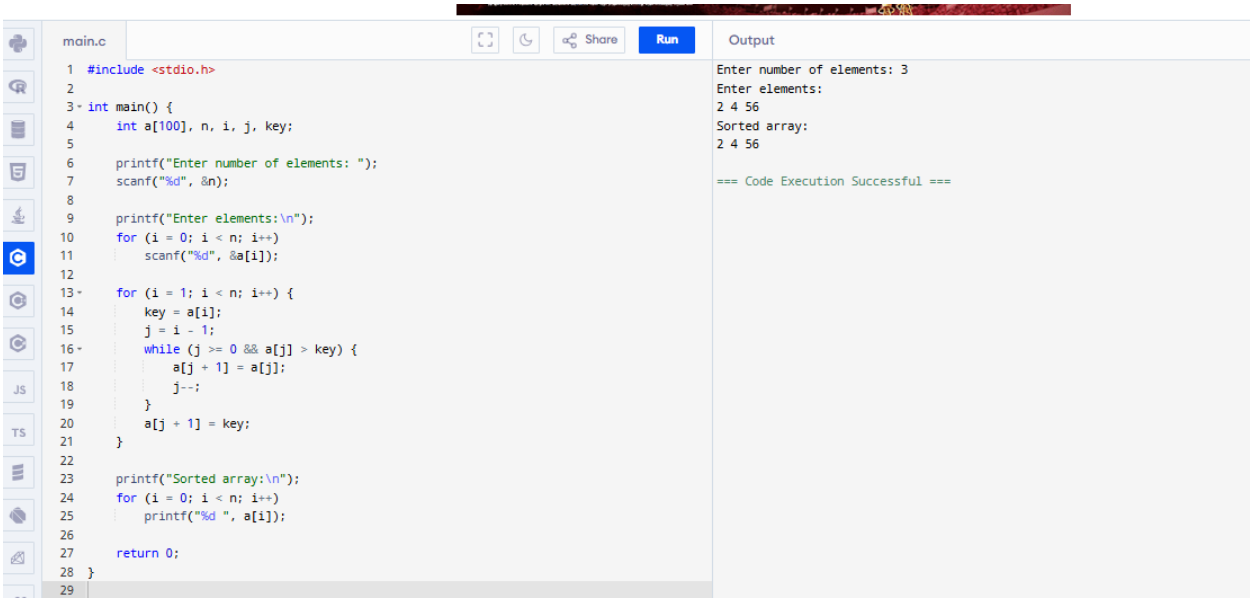
Lab exercise

16. Write a C program to arrange a series of numbers using Insertion Sort

Aim: To sort a list of numbers using the Insertion Sort algorithm.



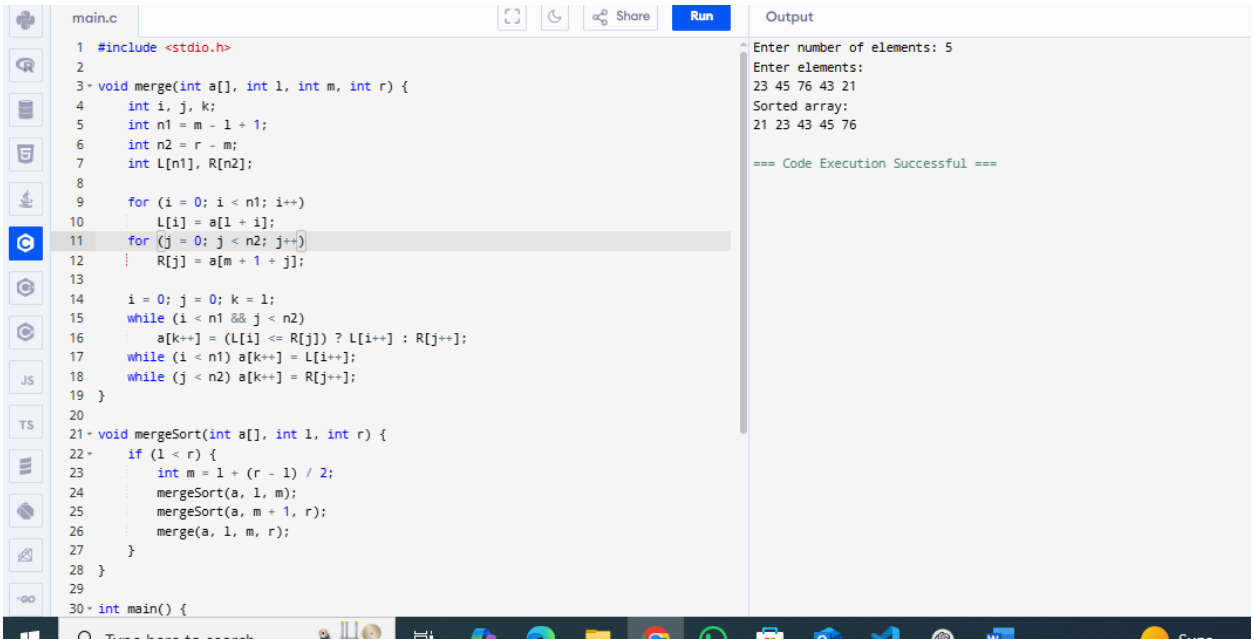
Result:

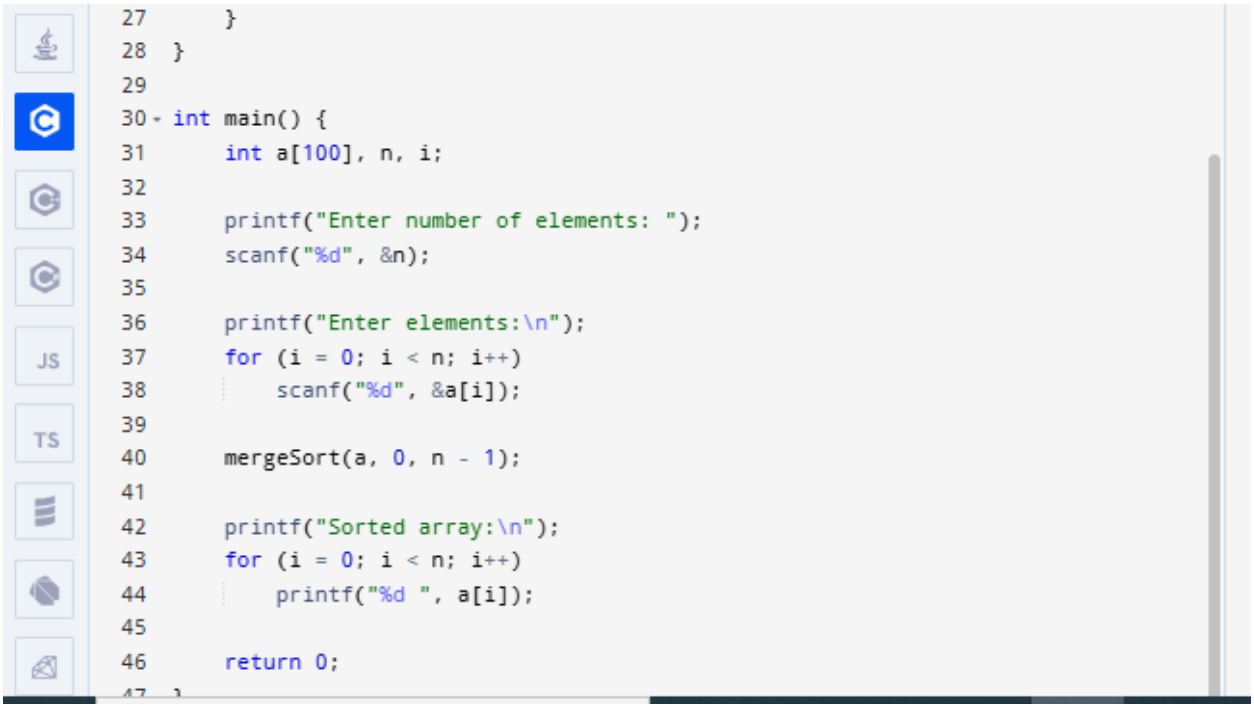
The program successfully sorted the numbers using Insertion Sort.

17. Write a C program to arrange a series of numbers using Merge Sort

Aim:

To sort an array using Merge Sort.





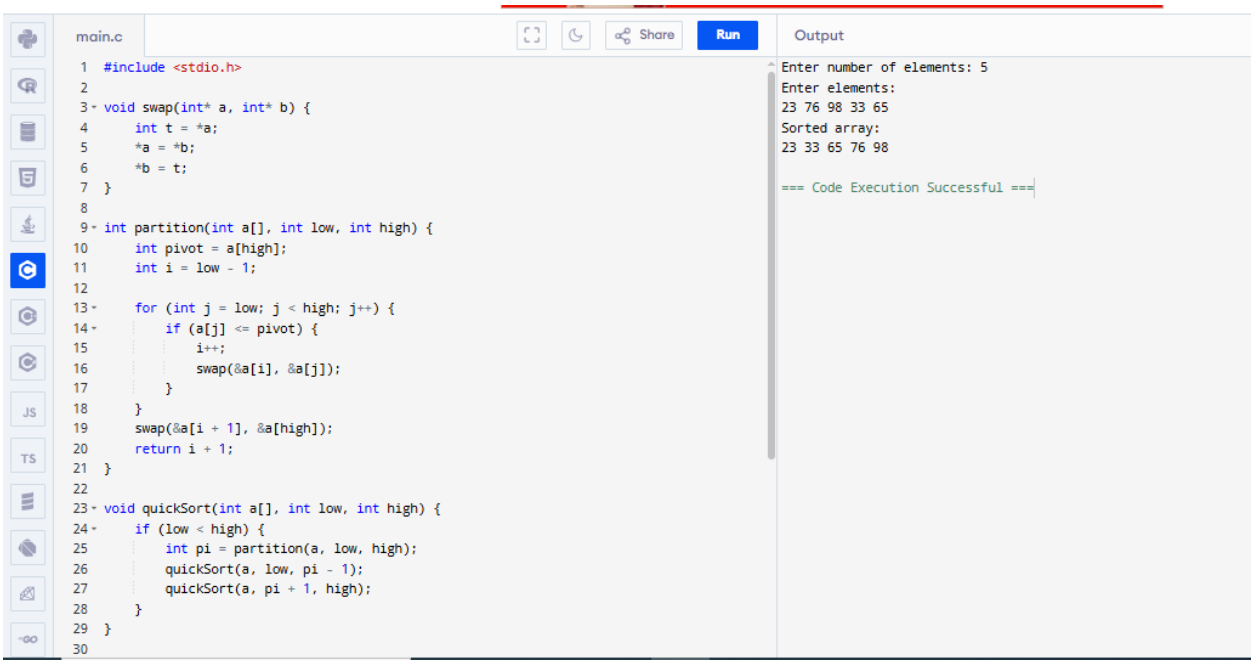
```
    }
}

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

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

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

    mergeSort(a, 0, n - 1);

    printf("Sorted array:\n");
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
```

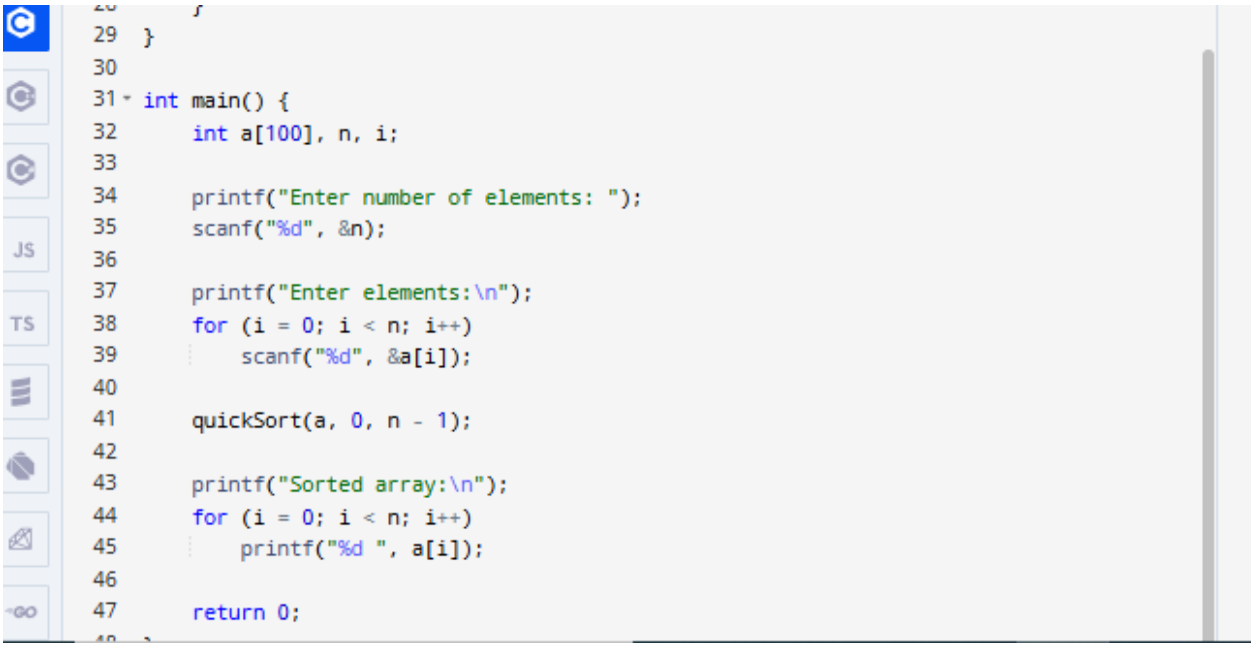
Result:

Successfully sorted using Merge Sort.

18. Write a C program to arrange a series of numbers using Quick Sort

Aim: To sort an array using the Quick Sort algorithm.





Result: Program correctly sorts elements using Quick Sort.

19. Write a C program to implement Heap sort

Aim:

To sort an array using Heap Sort

main.c

Share Run

Output

```c
#include <stdio.h>

void heapify(int a[], int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && a[l] > a[largest])
        largest = l;

    if (r < n && a[r] > a[largest])
        largest = r;

    if (largest != i) {
        int temp = a[i];
        a[i] = a[largest];
        a[largest] = temp;
        heapify(a, n, largest);
    }
}

void heapSort(int a[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(a, n, i);

    for (int i = n - 1; i > 0; i--) {
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;
        heapify(a, i, 0);
```

```
Enter number of elements: 5
Enter elements:
34 87 65 34 22
Sorted array:
22 34 34 65 87

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

```c
        heapify(a, i, 0);
    }
}

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

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

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

    heapSort(a, n);

    printf("Sorted array:\n");
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
```

Result: Heap Sort implemented and executed successfully.

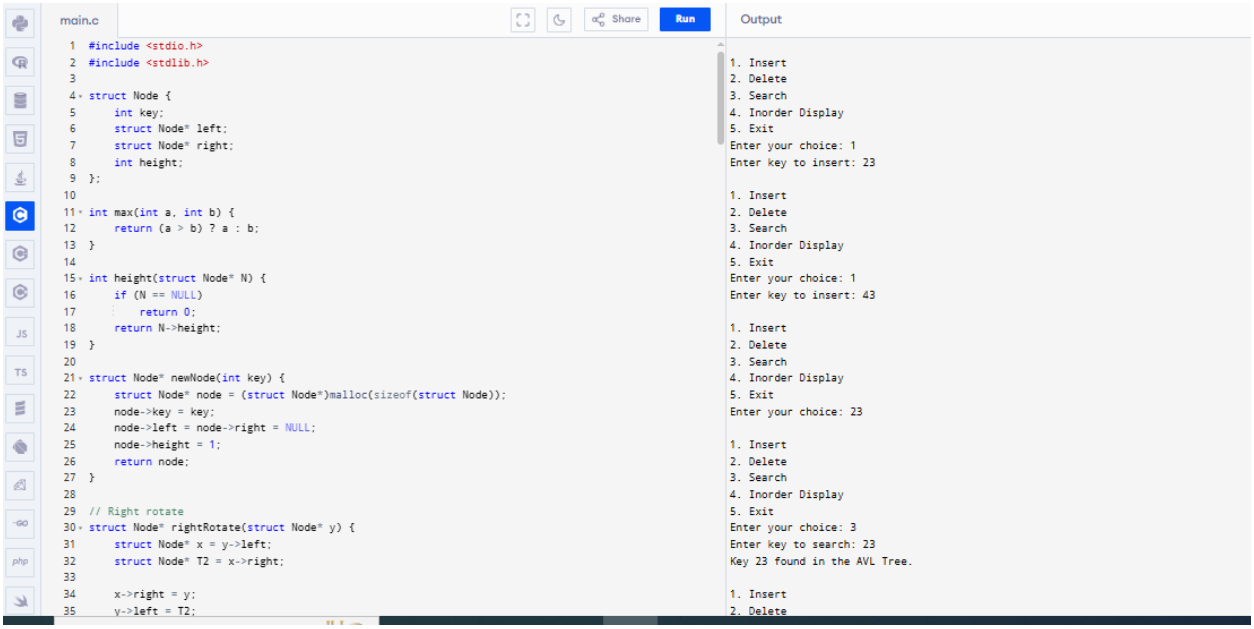
20. Write a program to perform the following operations:

a) Insert an element into a AVL tree

b) Delete an element from a AVL tree

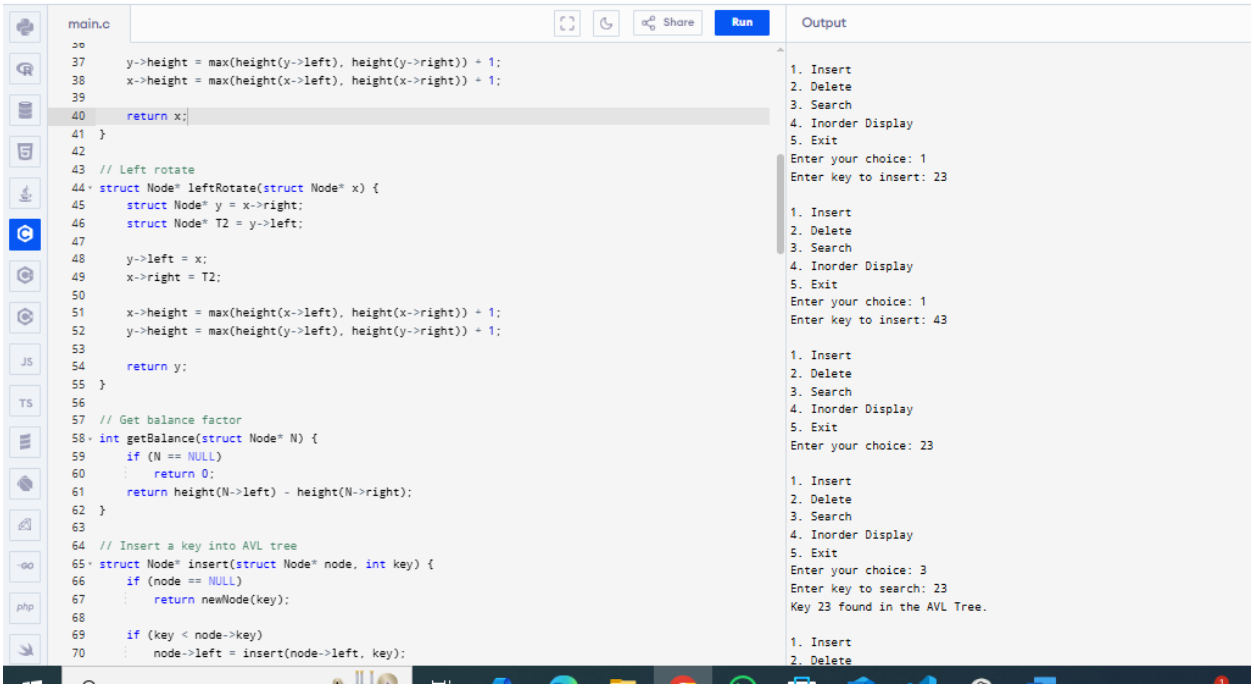
c) Search for a key element in a AVL tree

Aim: To write a C program to implement an AVL Tree with insertion, deletion, and search operations while maintaining balance.



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

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

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

int height(struct Node* N) {
    if (N == NULL)
        return 0;
    return N->height;
}

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

// Right rotate
struct Node* rightRotate(struct Node* y) {
    struct Node* x = y->left;
    struct Node* T2 = x->right;

    x->right = y;
    y->left = T2;
```

```
1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 1
Enter key to insert: 23

1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 1
Enter key to insert: 43

1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 23

1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 3
Enter key to search: 23
Key 23 found in the AVL Tree.

1. Insert
2. Delete
```



```c
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;

    return x;
}

// Left rotate
struct Node* leftRotate(struct Node* x) {
    struct Node* y = x->right;
    struct 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;
}

// Get balance factor
int getBalance(struct Node* N) {
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

// Insert a key into AVL tree
struct Node* insert(struct Node* node, int key) {
    if (node == NULL)
        return newNode(key);

    if (key < node->key)
        node->left = insert(node->left, key);
```

```
1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 1
Enter key to insert: 23

1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 1
Enter key to insert: 43

1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 23

1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 3
Enter key to search: 23
Key 23 found in the AVL Tree.

1. Insert
2. Delete
```

main.c

Share | Run | Output

```
79
80      // Left Left Case
81      if (balance > 1 && key < node->left->key)
82          return rightRotate(node);
83
84      // Right Right Case
85      if (balance < -1 && key > node->right->key)
86          return leftRotate(node);
87
88      // Left Right Case
89      if (balance > 1 && key > node->left->key) {
90          node->left = leftRotate(node->left);
91          return rightRotate(node);
92      }
93
94      // Right Left Case
95      if (balance < -1 && key < node->right->key) {
96          node->right = rightRotate(node->right);
97          return leftRotate(node);
98      }
99
100     return node;
101 }
102
103 // Get minimum value node
104 struct Node* minValueNode(struct Node* node) {
105     struct Node* current = node;
106     while (current->left != NULL)
107         current = current->left;
108     return current;
109 }
110
111 // Delete a node from AVL tree
112 struct Node* deleteNode(struct Node* root, int key) {
113     if (root == NULL)
```

```
1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 1
Enter key to insert: 23

1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 1
Enter key to insert: 43

1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 23

1. Insert
2. Delete
3. Search
4. Inorder Display
5. Exit
Enter your choice: 3
Enter key to search: 23
Key 23 found in the AVL Tree.

1. Insert
2. Delete
```

main.c

Share | Run | Output

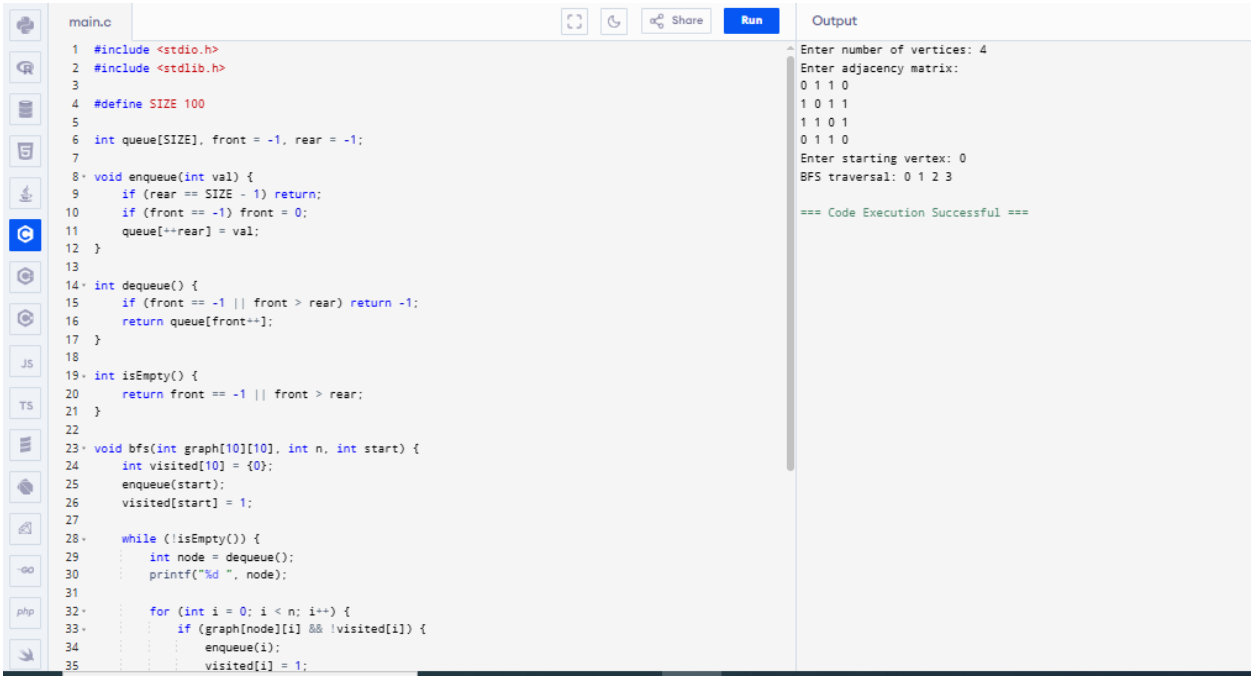
```
191
192     do {
193         printf("\n1. Insert\n2. Delete\n3. Search\n4. Inorder Display\n5. Exit\n");
194         printf("Enter your choice: ");
195         scanf("%d", &choice);
196
197         switch (choice) {
198             case 1:
199                 printf("Enter key to insert: ");
200                 scanf("%d", &key);
201                 root = insert(root, key);
202                 break;
203             case 2:
204                 printf("Enter key to delete: ");
205                 scanf("%d", &key);
206                 root = deleteNode(root, key);
207                 break;
208             case 3:
209                 printf("Enter key to search: ");
210                 scanf("%d", &key);
211                 if (search(root, key))
212                     printf("Key %d found in the AVL Tree.\n", key);
213                 else
214                     printf("Key %d not found.\n", key);
215                 break;
216             case 4:
217                 printf("Inorder Traversal: ");
218                 inorder(root);
219                 printf("\n");
220                 break;
221         }
222     } while (choice != 5);
223
224     return 0;
```

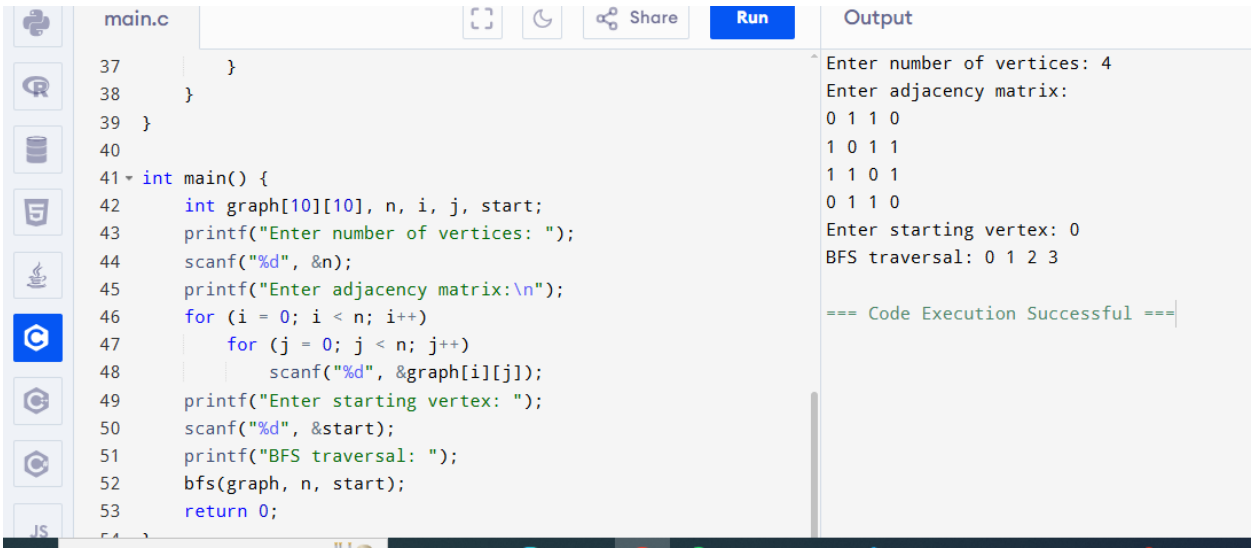
```
1. Insert
2. Delete
3. Search
4. Inorder Dis
5. Exit
Enter your cho
Enter key to i

1. Insert
2. Delete
3. Search
4. Inorder Dis
5. Exit
Enter your cho
Enter key to i

1. Insert
2. Delete
3. Search
4. Inorder Dis
5. Exit
Enter your cho

1. Insert
2. Delete
3. Search
4. Inorder Dis
5. Exit
Enter your cho
Enter key to s
Key 23 found i

1. Insert
2. Delete
```

Result: The AVL Tree operations of insertion, deletion, and search were performed successfully with height balancing maintained.

21. Write a C program to Graph traversal using Breadth First Search

Aim: To implement Breadth First Search (BFS) traversal on a graph using adjacency matrix.

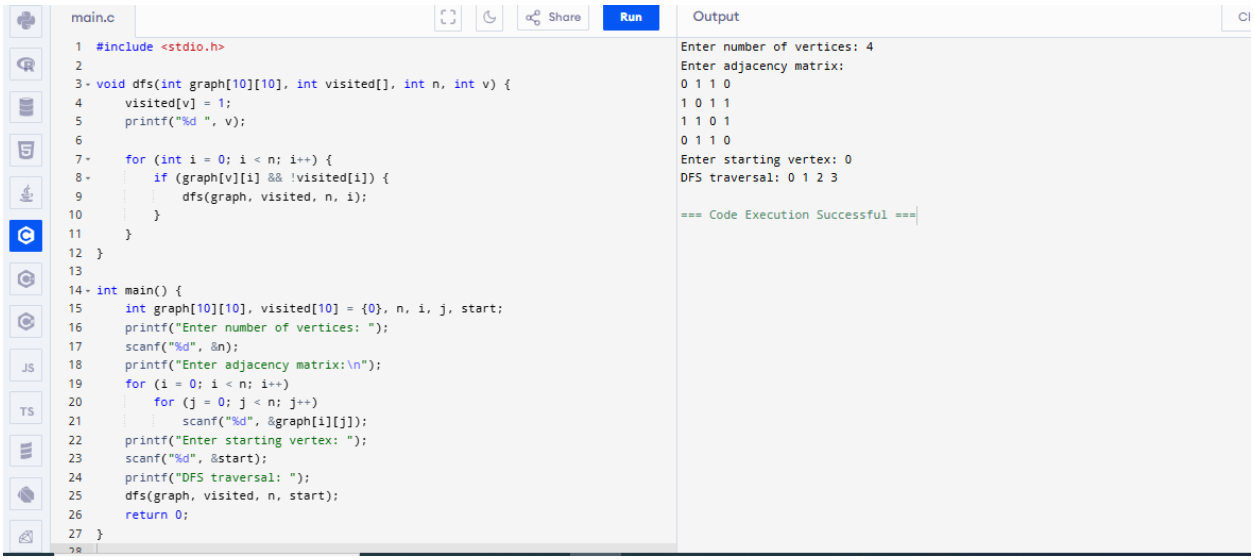




Result: Program successfully performs BFS traversal from a given start vertex.

22. Write a C program to Graph traversal using Depth First Search

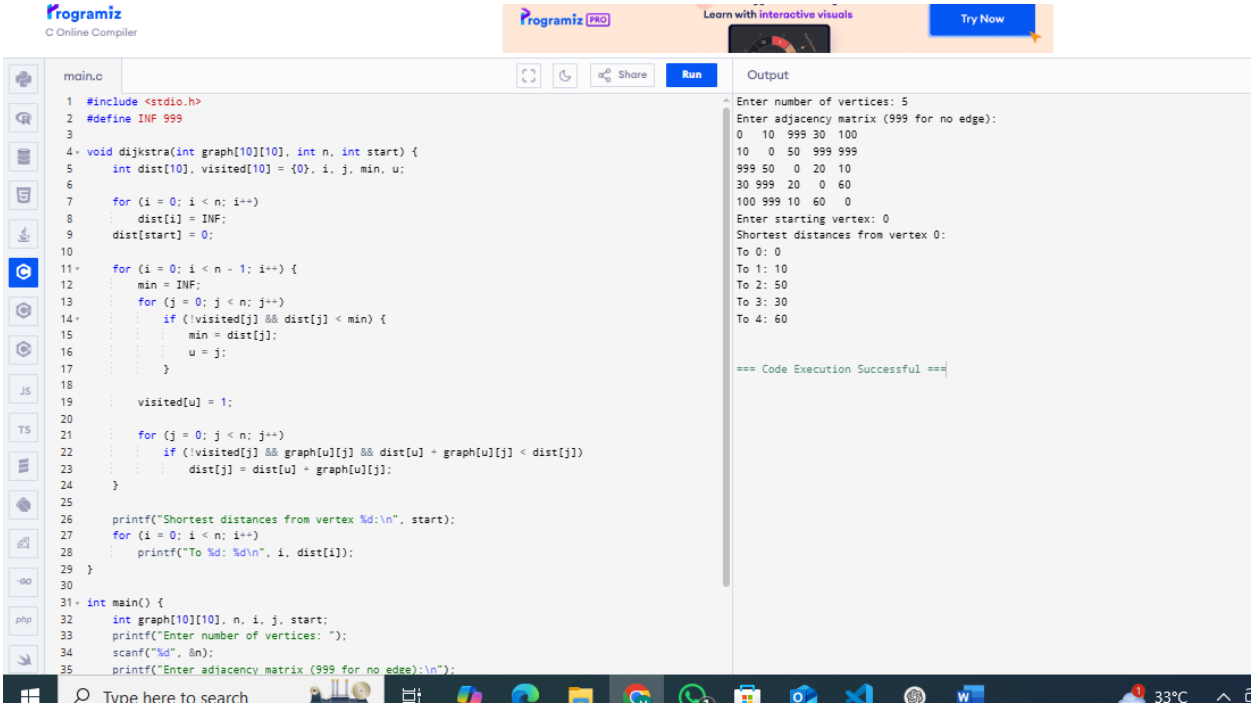
Aim: To implement Depth First Search (DFS) traversal on a graph using recursion.

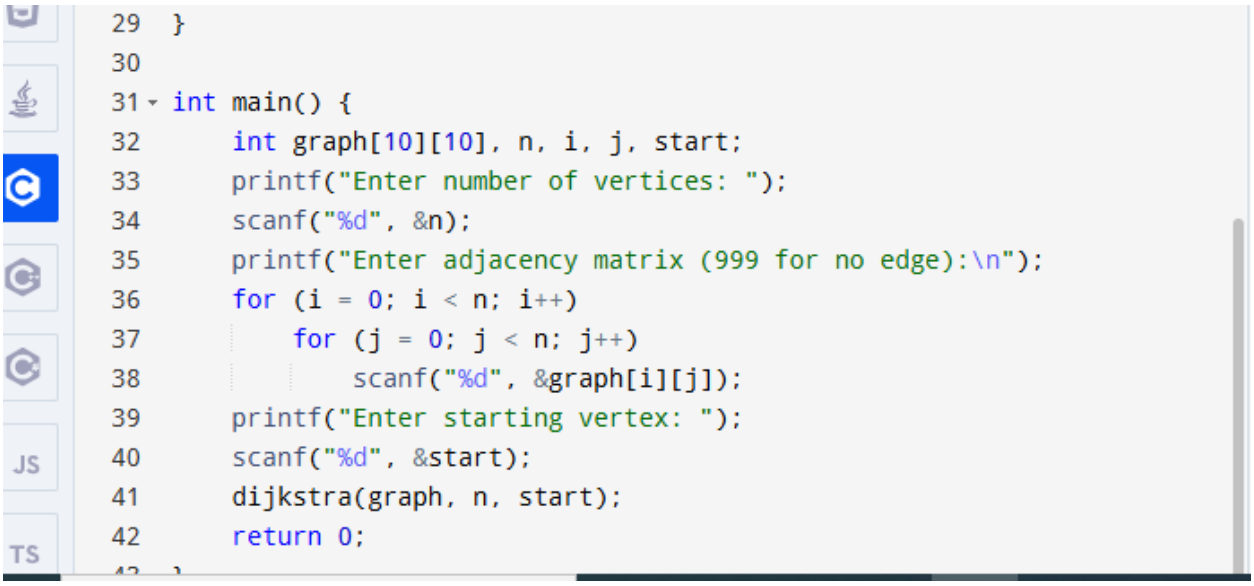


Result: DFS traversal correctly visits all connected vertices from a starting node.

23. Implementation of Shortest Path Algorithms using Dijkstra's Algorithm

Aim: To find the shortest path from a source node to all other nodes using Dijkstra's algorithm.

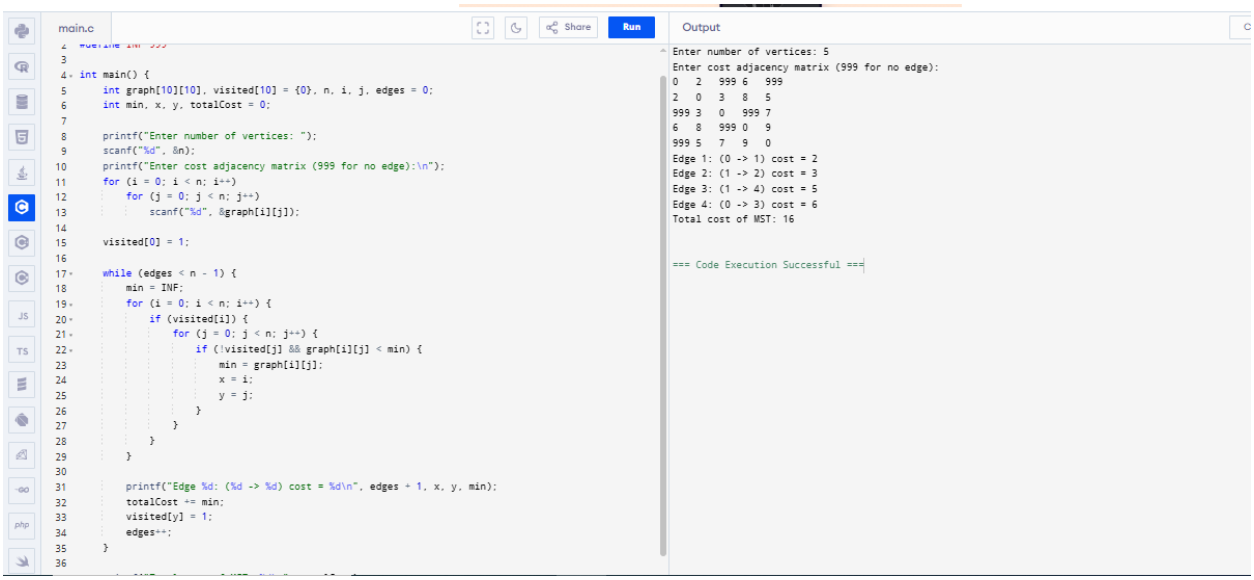




Result: The program computes the minimum distance from the start vertex to all other vertices correctly.

24. Implementation of Minimum Spanning Tree using Prim's Algorithm

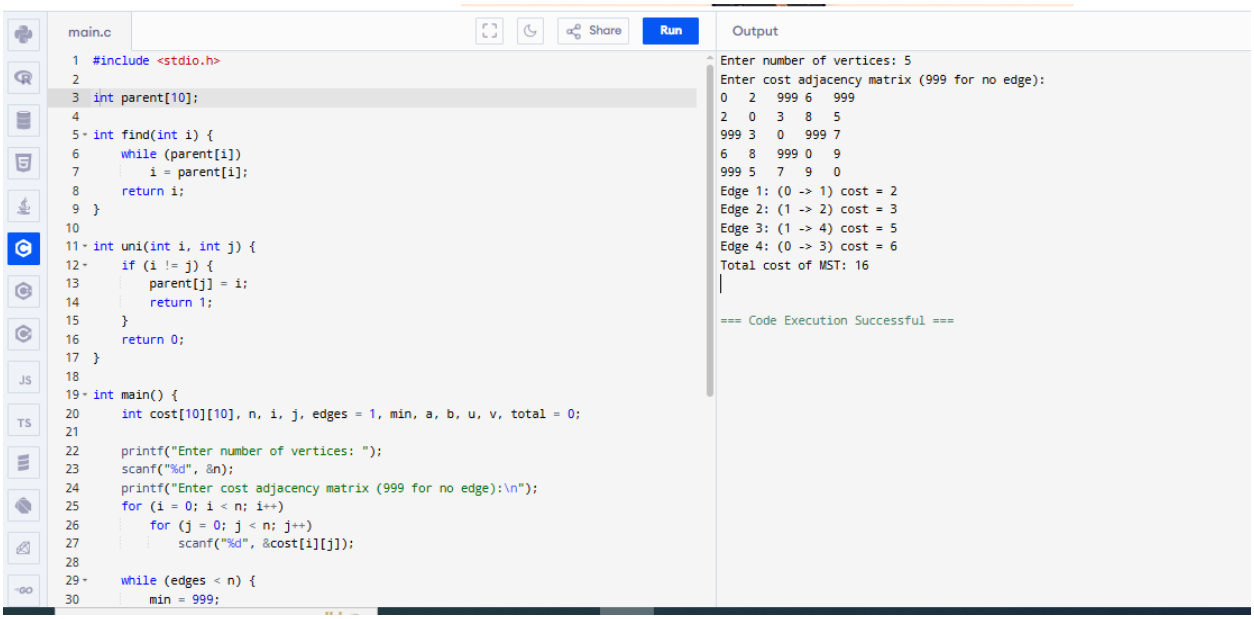
Aim: To find the Minimum Spanning Tree of a graph using Prim's algorithm.

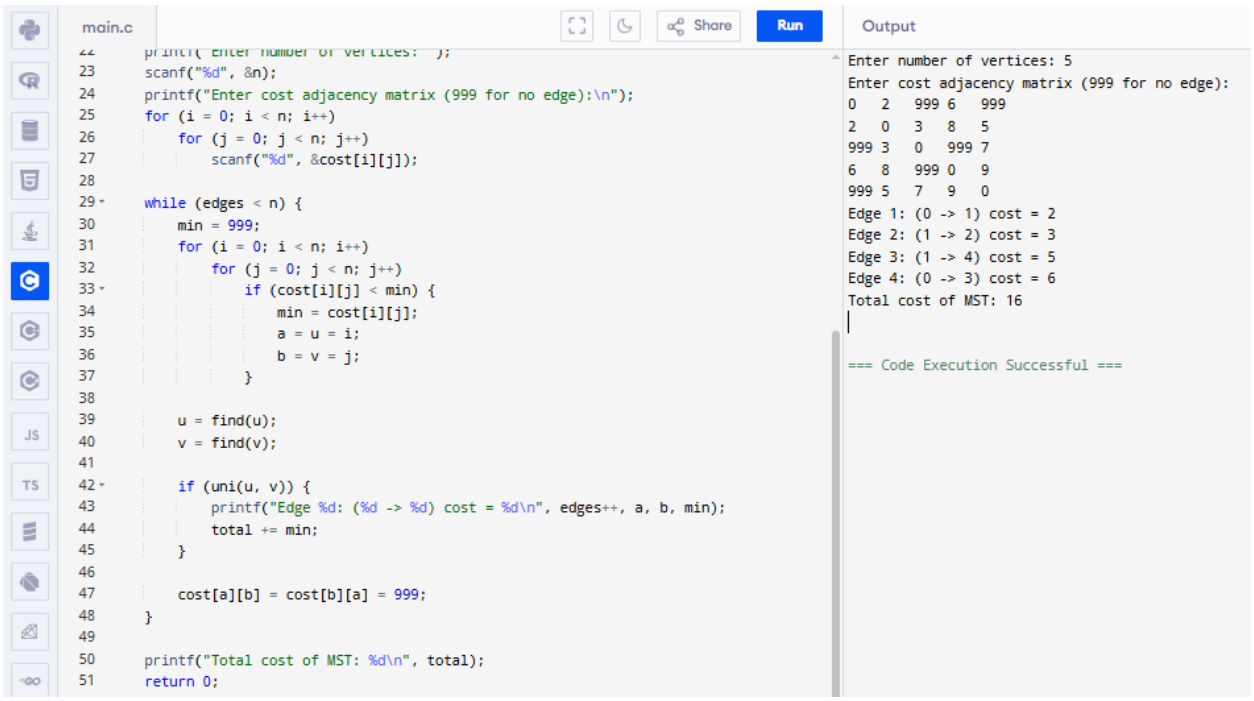


Result: The MST is generated with minimum cost using Prim's algorithm.

25. Implementation of Minimum Spanning Tree using Kruskal Algorithm.

Aim: To construct a Minimum Spanning Tree using Kruskal's algorithm.





Result: All edges of MST were correctly selected using Kruskal's algorithm, and minimum total cost was computed.