

11. Write a C program to implement Stack operations such as PUSH, POP and PEEK.

Aim

To implement a stack using an array and perform basic operations like PUSH, POP, and PEEK.

Algorithm

1. Initialize an array to represent the stack and a variable to keep track of the top element.

2. PUSH:

Check if the stack is full.

If not, increment the top index and store the element at that index.

3. POP:

Check if the stack is empty.

If not, return the element at the top index and decrement the top index.

4. PEEK:

Check if the stack is empty.

If not, return the element at the top index without modifying the stack.

Input:

1. PUSH

2. POP

3. PEEK

4. DISPLAY

5. EXIT

Enter your choice: 1

Enter value to push: 10

Enter your choice: 1

Enter value to push: 20

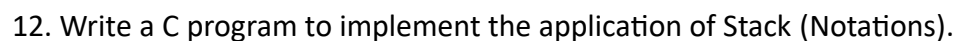
Enter your choice: 3

Output:

10 pushed onto stack.

20 pushed onto stack.

Code:



To implement a program that converts infix notation to postfix notation using a stack.

1. Initialize an empty stack.

2. Scan the infix expression from left to right.

3. If the scanned character is an operand, add it to the output string.

4. If the scanned character is an operator:

If the stack is empty or the top of the stack has lower precedence, push the operator onto the stack.

If the top of the stack has higher or equal precedence, pop operators from the stack and add them to the output string until a lower precedence operator is found or the stack is empty. Then, push the current operator onto the stack.

5. If the scanned character is a left parenthesis, push it onto the stack.

6. If the scanned character is a right parenthesis, pop operators from the stack and add them to the output string until a left parenthesis is found. Discard the left parenthesis.

7. After scanning the entire infix expression, pop any remaining operators from the stack and add them to the output string.

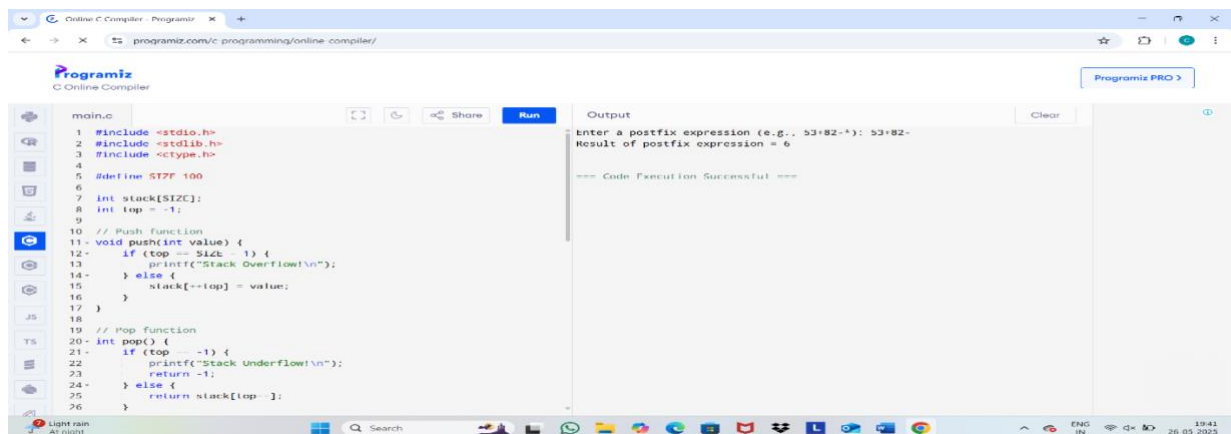
Input:

Enter a postfix expression (e.g., 53+82-): 53+82-

Output:

Result of postfix expression = 6

Code:



The screenshot shows a web browser window with the URL 'programiz.com/c-programming/online-compiler/'. The page title is 'Programiz C Online Compiler'. The code editor on the left contains the following C code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <ctype.h>
4 #define SIZE 100
5
6 int stack[SIZE];
7 int top = -1;
8
9 // Push function
10 void push(int value) {
11     if (top == SIZE - 1) {
12         printf("Stack Overflow\n");
13     } else {
14         stack[++top] = value;
15     }
16 }
17
18 // Pop function
19 int pop() {
20     if (top == -1) {
21         printf("Stack Underflow\n");
22         return -1;
23     } else {
24         return stack[top--];
25     }
26 }
```

The output window on the right shows the input 'Enter a postfix expression (e.g., 53+82-): 53+82-' and the result 'Result of postfix expression = 6'. Below the output, it says '=== Code Execution Successful ==='. The browser's taskbar at the bottom shows the time as 19:41 on 26.05.2023.

13. Write a C program to implement Queue operations such as ENQUEUE, DEQUEUE and Display.

Aim:

To implement a queue data structure and perform basic operations like ENQUEUE, DEQUEUE, and DISPLAY.

Algorithm:

ENQUEUE Operation

1. Check if the queue is full (i.e., rear == SIZE - 1).
2. If the queue is not full, increment the rear index.
3. If the queue is initially empty (i.e., front == -1), set front = 0.
4. Add the new element to the queue at the rear index.

DEQUEUE Operation

1. Check if the queue is empty (i.e., front == -1 or front > rear).
2. If the queue is not empty, remove the element at the front index.
3. Increment the front index.

4. If the queue becomes empty after dequeuing (i.e.,  $\text{front} > \text{rear}$ ), reset front and rear to -1.

#### DISPLAY Operation

1. Check if the queue is empty (i.e.,  $\text{rear} == -1$ ).
2. If the queue is not empty, iterate through the elements from front to rear index.
3. Print each element in the queue.

#### Queue Operations:

- ENQUEUE: Adds an element to the end of the queue.
- DEQUEUE: Removes an element from the front of the queue.
- DISPLAY: Prints all elements in the queue.

#### Input:

1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT

Enter your choice: 1

Enter value to enqueue: 10

Enter your choice: 1

Enter value to enqueue: 20

Enter your choice: 3

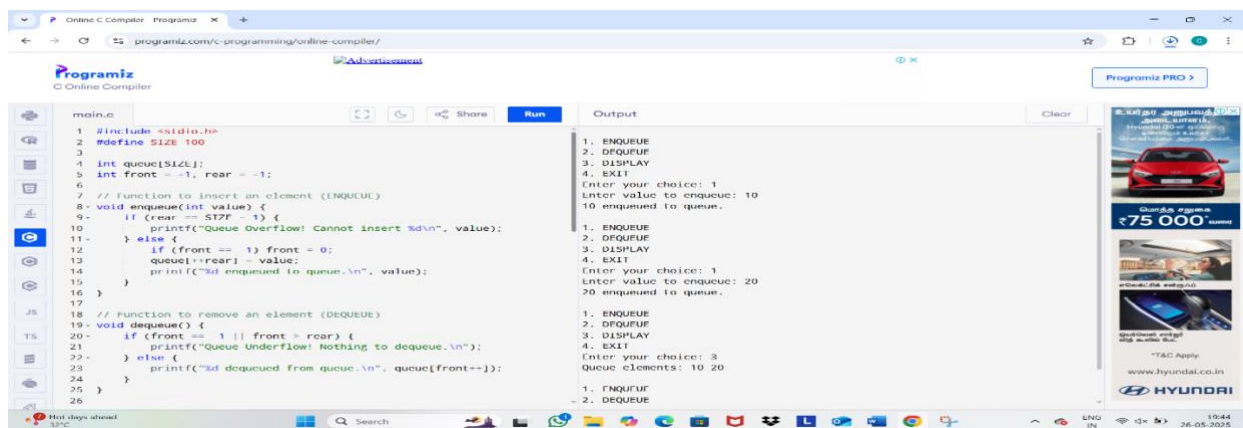
#### Output:

10 enqueued to queue.

20 enqueued to queue.

Queue elements: 10 20

Code:



The screenshot shows a web browser window with the URL `programiz.com/c-programming/online-compiler/`. The page displays a C program for implementing a queue. The code is as follows:

```
1 #include <stdio.h>
2 #define SIZE 100
3
4 int queue[SIZE];
5 int front = -1, rear = -1;
6
7 // function to insert an element (ENQUEUE)
8 void enqueue(int value) {
9     if (rear == SIZE - 1) {
10         printf("Queue Overflow! Cannot insert %d\n", value);
11     } else {
12         if (front == 1) front = 0;
13         queue[++rear] = value;
14         printf("%d enqueued to queue.\n", value);
15     }
16 }
17
18 // function to remove an element (DEQUEUE)
19 void dequeue() {
20     if (front == 1 || front > rear) {
21         printf("Queue Underflow! Nothing to dequeue.\n");
22     } else {
23         printf("%d dequeued from queue.\n", queue[front++]);
24     }
25 }
26
```

The output of the program is shown on the right side of the compiler interface:

```
1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter value to enqueue: 10
10 enqueued to queue.

1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 1
Enter value to enqueue: 20
20 enqueued to queue.

1. ENQUEUE
2. DEQUEUE
3. DISPLAY
4. EXIT
Enter your choice: 3
Queue elements: 10 20

1. ENQUEUE
2. DEQUEUE
```

14. Write a C program to implement the Tree Traversals (Inorder, Preorder, Postorder).

Aim

To implement a binary tree and perform Inorder, Preorder, and Postorder traversals.

Algorithm

Node Structure

- Define a node structure with an integer data field and pointers to the left and right child nodes.

Tree Traversals

- Inorder Traversal: Left subtree -> Root node -> Right subtree

1. Recursively traverse the left subtree.
2. Visit the root node.
3. Recursively traverse the right subtree.

- Preorder Traversal: Root node -> Left subtree -> Right subtree

1. Visit the root node.
2. Recursively traverse the left subtree.
3. Recursively traverse the right subtree.

- Postorder Traversal: Left subtree -> Right subtree -> Root node

1. Recursively traverse the left subtree.
2. Recursively traverse the right subtree.

Input:

Enter number of nodes to insert: 5

Enter 5 values:

40 20 60 10 30

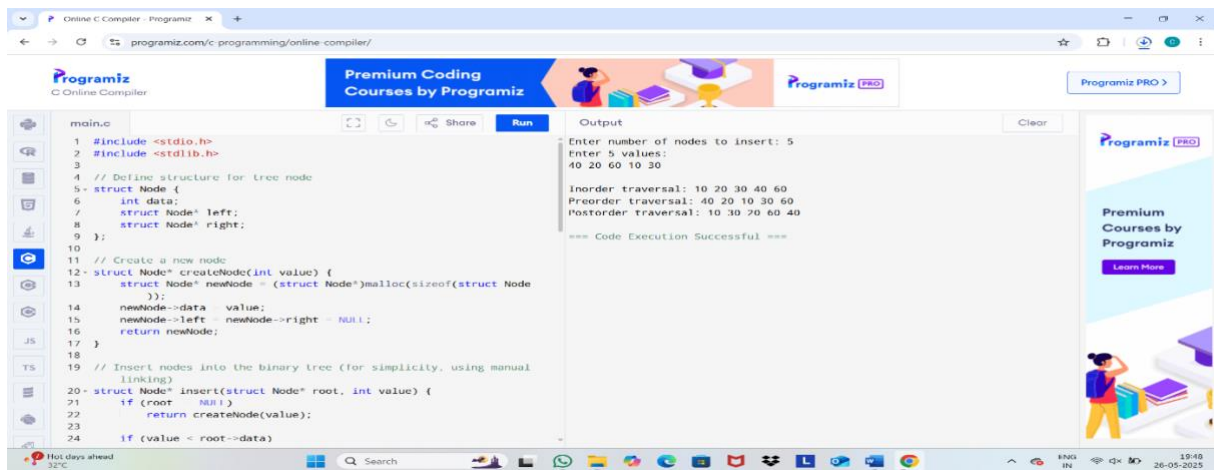
Output:

Inorder traversal: 10 20 30 40 60

Preorder traversal: 40 20 10 30 60

Postorder traversal: 10 30 20 60 40

Code:



The screenshot shows a web browser window with the URL `programiz.com/c-programming/online-compiler/`. The page features the Programiz logo and a banner for "Premium Coding Courses by Programiz". The main content area displays a C program for inserting nodes into a binary tree and performing inorder, preorder, and postorder traversals. The code is as follows:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // Define structure for tree node
5 struct Node {
6     int data;
7     struct Node* left;
8     struct Node* right;
9 };
10
11 // Create a new node
12 struct Node* createNode(int value) {
13     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
14     newNode->data = value;
15     newNode->left = newNode->right = NULL;
16     return newNode;
17 }
18
19 // Insert nodes into the binary tree (for simplicity, using manual linking)
20 struct Node* insert(struct Node* root, int value) {
21     if (root == NULL) {
22         return createNode(value);
23     }
24     if (value < root->data) {
25         root->left = insert(root->left, value);
26     } else {
27         root->right = insert(root->right, value);
28     }
29     return root;
30 }
31
32 // Inorder traversal
33 void inorder(struct Node* root) {
34     if (root == NULL) return;
35     inorder(root->left);
36     printf("%d ", root->data);
37     inorder(root->right);
38 }
39
40 // Preorder traversal
41 void preorder(struct Node* root) {
42     if (root == NULL) return;
43     printf("%d ", root->data);
44     preorder(root->left);
45     preorder(root->right);
46 }
47
48 // Postorder traversal
49 void postorder(struct Node* root) {
50     if (root == NULL) return;
51     postorder(root->left);
52     postorder(root->right);
53     printf("%d ", root->data);
54 }
55
56 int main() {
57     struct Node* root = NULL;
58     int n, value;
59     printf("Enter number of nodes to insert: ");
60     scanf("%d", &n);
61     printf("Enter %d values:\n", n);
62     for (int i = 0; i < n; i++) {
63         scanf("%d", &value);
64         root = insert(root, value);
65     }
66     printf("\nInorder traversal: ");
67     inorder(root);
68     printf("\nPreorder traversal: ");
69     preorder(root);
70     printf("\nPostorder traversal: ");
71     postorder(root);
72     return 0;
73 }
```

The output window on the right shows the following results:

```
Enter number of nodes to insert: 5
Enter 5 values:
40 20 60 10 30

Inorder traversal: 10 20 30 40 60
Preorder traversal: 40 20 10 30 60
Postorder traversal: 10 30 20 60 40

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

15. Write a C program to implement hashing using Linear Probing method.

Aim

To implement a hash table using Linear Probing method for collision resolution.

Algorithm

1. Initialize a hash table with a fixed size.
2. Define a hash function to map keys to indices of the hash table.
3. When inserting a key-value pair:

Calculate the index using the hash function.

If the index is empty, insert the key-value pair.

If the index is occupied (collision), probe linearly to find the next empty slot.

#### 4. When searching for a key:

- Calculate the index using the hash function.
- If the key is found at the index, return the value.
- If the key is not found, probe linearly until the key is found or an empty slot is encountered.

Input:

Enter number of elements: 4

Enter 4 values:

23 33 43 13

Output:

Index 0: -1

Index 1: -1

Index 2: -1

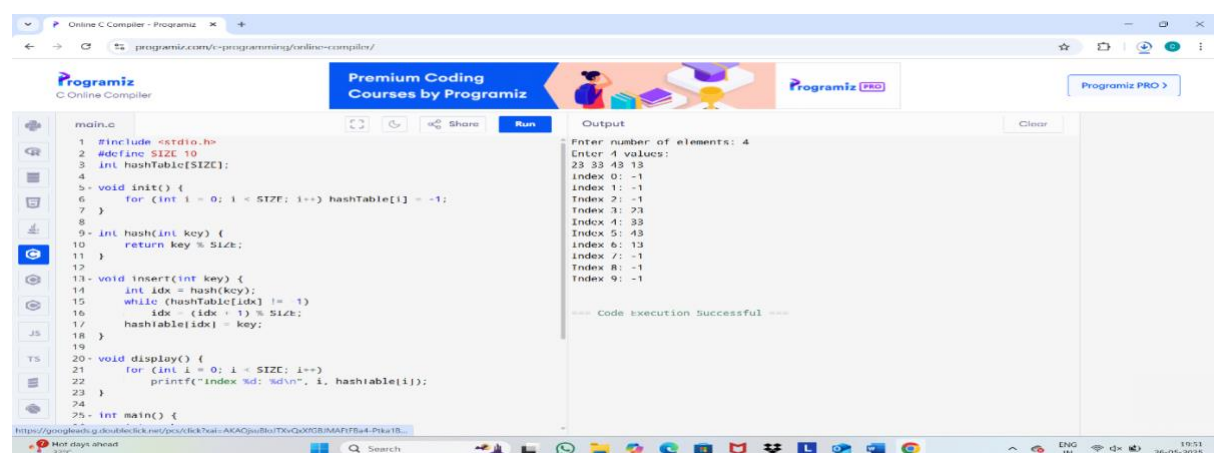
Index 3: 23

Index 4: 33

Index 5: 43

Index 6: 13

Code:



The screenshot shows a web browser window with the URL `programiz.com/c-programming/online-compiler/`. The page features the Programiz logo and a banner for "Premium Coding Courses by Programiz". The main content area is divided into two panels: a code editor on the left and an output window on the right.

The code editor contains the following C code:

```
1 #include <stdio.h>
2 #define SIZE 10
3 int hashTable[SIZE];
4
5 void init() {
6     for (int i = 0; i < SIZE; i++) hashTable[i] = -1;
7 }
8
9 int hash(int key) {
10     return key % SIZE;
11 }
12
13 void insert(int key) {
14     int idx = hash(key);
15     while (hashTable[idx] != -1)
16         idx = (idx + 1) % SIZE;
17     hashTable[idx] = key;
18 }
19
20 void display() {
21     for (int i = 0; i < SIZE; i++)
22         printf("Index %d: %d\n", i, hashTable[i]);
23 }
24
25 int main() {
```

The output window displays the following text:

```
Enter number of elements: 4
Enter 4 values:
23 33 43 13
Index 0: -1
Index 1: -1
Index 2: -1
Index 3: 23
Index 4: 33
Index 5: 43
Index 6: 13
Index 7: -1
Index 8: -1
Index 9: -1
==== Code execution Successful =====
```

The bottom of the browser window shows the Windows taskbar with the date and time as 10:11 on 26-05-2023.

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

Aim

To implement the Insertion Sort algorithm to arrange a series of numbers in ascending order.

Algorithm

1. Start with the second element of the array (index 1).
2. Compare the current element with the previous elements.
3. Shift the previous elements that are greater than the current element one position to the right.
4. Insert the current element at the correct position.
5. Repeat steps 2-4 for the remaining elements in the array.

Input:

Enter number of elements: 5

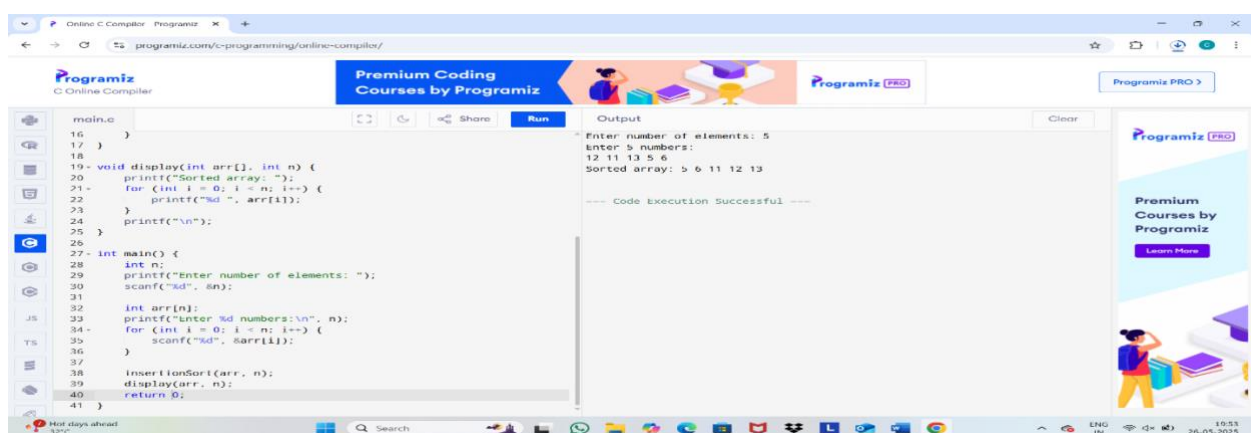
Enter 5 numbers:

12 11 13 5 6

Output:

Sorted array: 5 6 11 12 13

Code:



The screenshot shows a web browser window with the URL `programiz.com/c-programming/online-compiler/`. The page features the Programiz logo and a navigation bar. The main content area is divided into two panels. The left panel, titled 'main.c', contains the C code for the Insertion Sort algorithm. The right panel, titled 'Output', shows the program's execution results. The code in the left panel is as follows:

```
16 }
17 }
18
19 void display(int arr[], int n) {
20     printf("Sorted array: ");
21     for (int i = 0; i < n; i++) {
22         printf("%d ", arr[i]);
23     }
24     printf("\n");
25 }
26
27 int main() {
28     int n;
29     printf("Enter number of elements: ");
30     scanf("%d", &n);
31
32     int arr[n];
33     printf("Enter %d numbers:\n", n);
34     for (int i = 0; i < n; i++) {
35         scanf("%d", &arr[i]);
36     }
37
38     insertionSort(arr, n);
39     display(arr, n);
40     return 0;
41 }
```

The output panel displays the following text:

```
Enter number of elements: 5
Enter 5 numbers:
12 11 13 5 6
Sorted array: 5 6 11 12 13
```

Below the output, it states 'Code Execution Successful'.



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

Aim

To implement the Merge Sort algorithm to arrange a series of numbers in ascending order.

Algorithm

1. Divide the array into two halves until each half has one element (base case).
2. Merge the divided halves in a sorted manner.
3. Repeat step 2 until the entire array is merged and sorted.

Merge Function

1. Compare elements from the two halves.
2. Place the smaller element in the merged array.
3. Repeat step 1 until all elements from both halves are merged.

Input:

Enter number of elements: 5

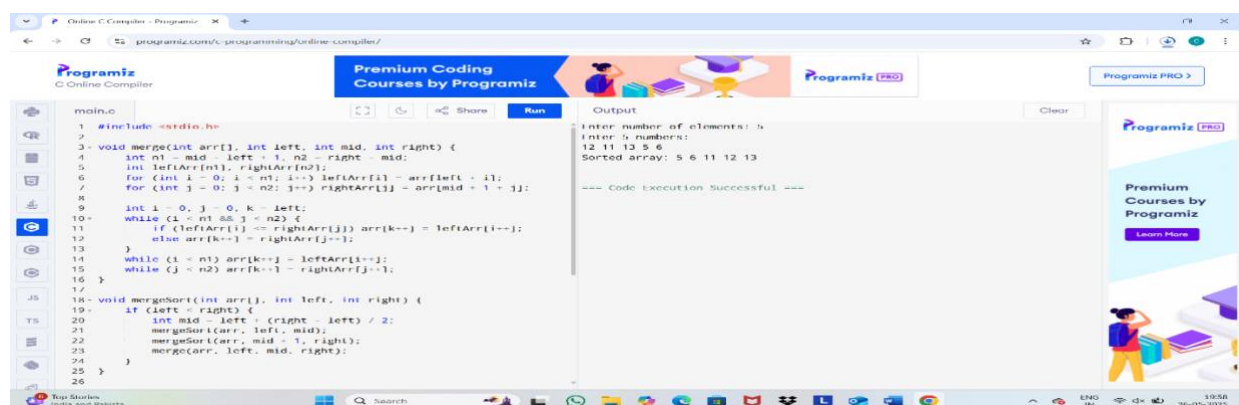
Enter 5 numbers:

12 11 13 5 6

Output:

Sorted array: 5 6 11 12 13

Code:



The screenshot shows a web browser window with the URL `programiz.com/c-programming/online-compiler/`. The page features the Programiz logo and a navigation bar with "Premium Coding Courses by Programiz" and a "Programiz PRO" button. The main content area is divided into two panels. The left panel, titled "main.c", contains the following C code:

```
1 #include <stdio.h>
2
3 void merge(int arr[], int left, int mid, int right) {
4     int n1 = mid - left + 1, n2 = right - mid;
5     int leftArr[n1], rightArr[n2];
6     for (int i = 0; i < n1; i++) leftArr[i] = arr[left + i];
7     for (int j = 0; j < n2; j++) rightArr[j] = arr[mid + 1 + j];
8
9     int i = 0, j = 0, k = left;
10    while (i < n1 && j < n2) {
11        if (leftArr[i] < rightArr[j]) arr[k++] = leftArr[i++];
12        else arr[k++] = rightArr[j++];
13    }
14    while (i < n1) arr[k++] = leftArr[i++];
15    while (j < n2) arr[k++] = rightArr[j++];
16 }
17
18 void mergeSort(int arr[], int left, int right) {
19     if (left < right) {
20         int mid = left + (right - left) / 2;
21         mergeSort(arr, left, mid);
22         mergeSort(arr, mid + 1, right);
23         merge(arr, left, mid, right);
24     }
25 }
26
```

The right panel, titled "Output", displays the program's execution results:

```
Enter number of elements: 5
Enter 5 numbers:
12 11 13 5 6
Sorted array: 5 6 11 12 13
--- Code Execution Successful ---
```

At the bottom of the browser window, a taskbar shows various system icons and the date/time: 10:58 AM, 26-05-2025.

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

Aim

To implement the Quick Sort algorithm to arrange a series of numbers in ascending order.

Algorithm

1. Choose a pivot element from the array.
2. Partition the array around the pivot element.
3. Recursively apply the above steps to the sub-arrays of elements less than and greater than the pivot.

Partition Function

1. Select the last element as the pivot.
2. Initialize the partition index to the start of the array.
3. Iterate through the array and swap elements smaller than the pivot with the element at the partition index.
4. Swap the pivot element with the element at the partition index.

Input:

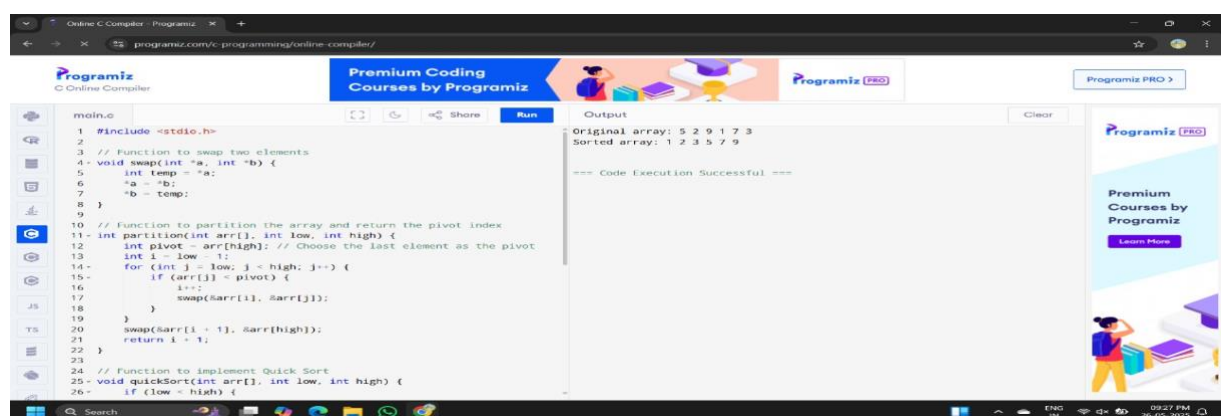
Enter no of Elements: 5

Enter the Elements: 12 3 5 7 19

Output:

3 5 7 12 19

Code:



The screenshot shows a web browser window with the URL `programiz.com/c-programming/online-compiler/`. The page features the Programiz logo and a navigation bar with links to 'Premium Coding Courses by Programiz' and 'Programiz PRO'. The main content area is divided into three sections: a code editor, an output window, and a sidebar with course recommendations.

The code editor displays the following C program:

```
1 #include <stdio.h>
2
3 // Function to swap two elements
4 void swap(int *a, int *b) {
5     int temp = *a;
6     *a = *b;
7     *b = temp;
8 }
9
10 // Function to partition the array and return the pivot index
11 int partition(int arr[], int low, int high) {
12     int pivot = arr[high]; // Choose the last element as the pivot
13     int i = low - 1;
14     for (int j = low; j < high; j++) {
15         if (arr[j] < pivot) {
16             i++;
17             swap(&arr[i], &arr[j]);
18         }
19     }
20     swap(&arr[i + 1], &arr[high]);
21     return i + 1;
22 }
23
24 // Function to implement Quick Sort
25 void quickSort(int arr[], int low, int high) {
26     if (low < high) {
```

The output window displays the following text:

```
Original array: 5 2 9 1 7 3
Sorted array: 1 2 3 5 7 9
=== Code Execution Successful ===
```

The sidebar on the right contains a section titled 'Premium Courses by Programiz' with a 'Learn More' button and an illustration of a person climbing stairs.

19. Write a C program to implement Heap sort.

Aim

To implement the Heap Sort algorithm to arrange a series of numbers in ascending order.

Algorithm

1. Build a max heap from the unsorted array.
2. Swap the root node (maximum element) with the last node in the heap.
3. Reduce the heap size by 1 and heapify the root node.
4. Repeat steps 2-3 until the heap is empty.

Heapify Function

1. Assume the current node is the largest.
2. Compare the current node with its left and right children.
3. If a child is larger, swap the current node with the child.
4. Recursively heapify the affected sub-tree.

Input:

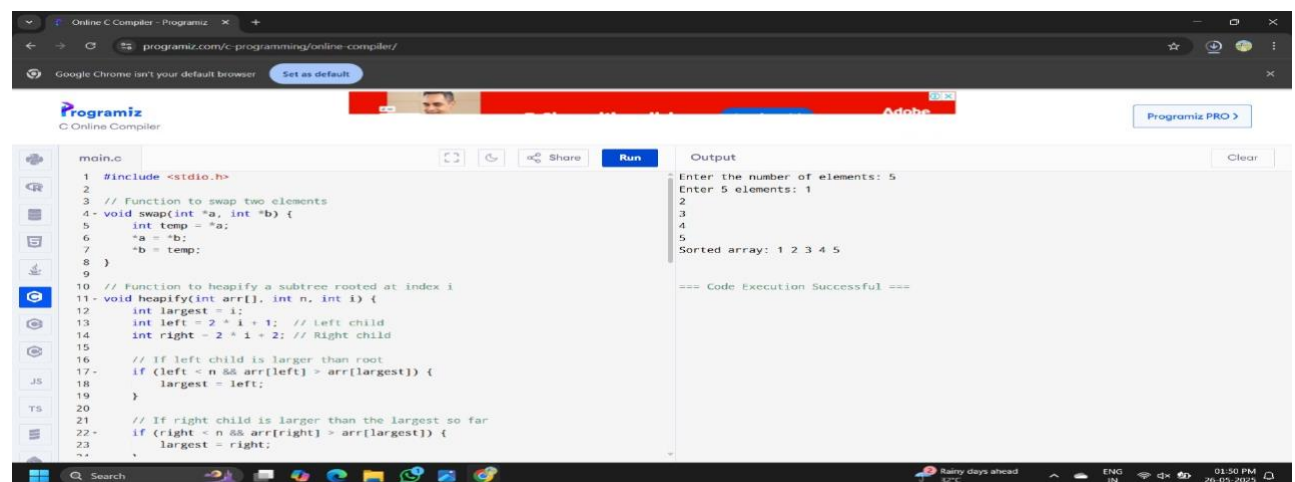
Enter the number of elements: 6

Enter 6 elements: 12 11 13 5 6 7

Output:

Sorted array: 5 6 7 11 12 13

Code:



```
main.c
1 #include <stdio.h>
2
3 // Function to swap two elements
4 void swap(int *a, int *b) {
5     int temp = *a;
6     *a = *b;
7     *b = temp;
8 }
9
10 // Function to heapify a subtree rooted at index i
11 void heapify(int arr[], int n, int i) {
12     int largest = i;
13     int left = 2 * i + 1; // Left child
14     int right = 2 * i + 2; // Right child
15
16     // If left child is larger than root
17     if (left < n && arr[left] > arr[largest]) {
18         largest = left;
19     }
20
21     // If right child is larger than the largest so far
22     if (right < n && arr[right] > arr[largest]) {
23         largest = right;
24     }
25
26     // Swap the root with the largest element
27     if (largest != i) {
28         swap(&arr[i], &arr[largest]);
29     }
30
31     // Recursively heapify the affected sub-tree
32     heapify(arr, n, largest);
33 }
34
35 int main() {
36     int n;
37     printf("Enter the number of elements: ");
38     scanf("%d", &n);
39
40     int arr[n];
41     printf("Enter %d elements: ", n);
42     for (int i = 0; i < n; i++) {
43         scanf("%d", &arr[i]);
44     }
45
46     // Build a max heap
47     for (int i = n/2 - 1; i >= 0; i--) {
48         heapify(arr, n, i);
49     }
50
51     // Sort the array
52     for (int i = n - 1; i > 0; i--) {
53         swap(&arr[0], &arr[i]);
54         heapify(arr, i, 0);
55     }
56
57     printf("Sorted array: ");
58     for (int i = 0; i < n; i++) {
59         printf("%d ", arr[i]);
60     }
61     printf("\n");
62     return 0;
63 }
```

Output

```
Enter the number of elements: 5
Enter 5 elements: 1
2
3
4
5
Sorted array: 1 2 3 4 5

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

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 implement an AVL tree and perform insertion, deletion, and search operations.

Algorithm

Insertion

1. Insert the new node as in a binary search tree.
2. Check if the tree is balanced. If not, perform rotations to balance the tree.

Deletion

1. Find the node to be deleted.
2. If the node has no children, simply remove it.
3. If the node has one child, replace it with its child.
4. If the node has two children, find its in-order successor and replace the node with it.
5. Check if the tree is balanced. If not, perform rotations to balance the tree.

Search

1. Start at the root node.
2. Compare the key with the node's key.
3. If the key is less than the node's key, move to the left child.
4. If the key is greater than the node's key, move to the right child.
5. Repeat steps 2-4 until the key is found or the tree is exhausted.

Input:

1. Insert
2. Delete
3. Search
4. Inorder

5. Exit

Enter your choice: 1

Enter element to insert: 10

1. Insert

2. Delete

3. Search

4. Inorder

5. Exit

Enter your choice: 1

Enter element to insert: 20

1. Insert

2. Delete

3. Search

4. Inorder

5. Exit

Enter your choice: 4

Inorder Traversal: 10 20

Output:

1. Insert

2. Delete

3. Search

4. Inorder

5. Exit

Enter your choice: 2

Enter element to delete: 20

1. Insert

2. Delete

3. Search

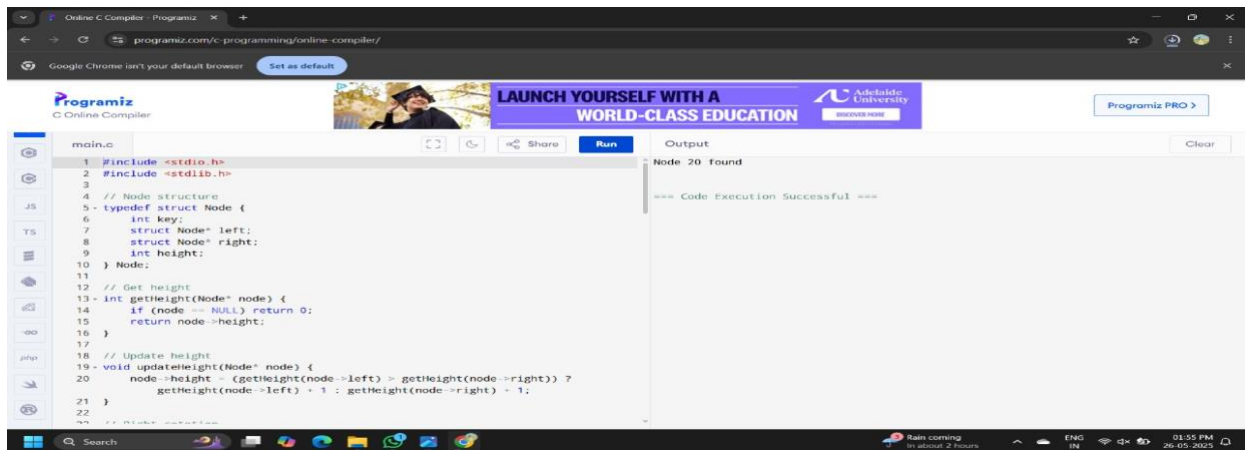
4. Inorder

5. Exit

Enter your choice: 4

Inorder Traversal: 10

Code:



The screenshot shows a web browser window with the URL `programiz.com/c-programming/online-compiler/`. The page features a header with the Programiz logo and a banner for Adelaide University. The main content area displays a C code editor with the following code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // Node structure
5 typedef struct Node {
6     int key;
7     struct Node* left;
8     struct Node* right;
9     int height;
10 } Node;
11
12 // Get height
13 int getHeight(Node* node) {
14     if (node == NULL) return 0;
15     return node->height;
16 }
17
18 // Update height
19 void updateHeight(Node* node) {
20     node->height = (getHeight(node->left) > getHeight(node->right)) ?
        getHeight(node->left) + 1 : getHeight(node->right) + 1;
21 }
22
23 // Print structure
```

The output window on the right shows the result of the code execution:

```
Node 20 found
=== Code Execution Successful ===
```

The bottom of the image shows a Windows taskbar with various icons and a system clock indicating 01:55 PM on 26-05-2025.