pression so for.

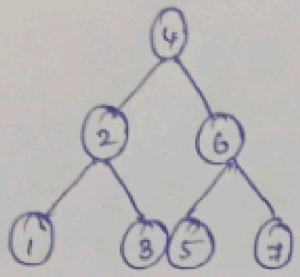
(
(a
(a+
(a+b
a+b)
(a+b)*
(a+b)* c
(a+b)* c
(a+b)*c-
a+b)*c-d
(a+b)*c-d.

mbol".

29/07/24

1. Binary Tree Traversal:-

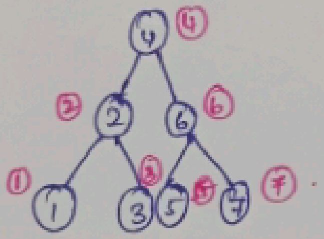
Ex-1:



Inorder Traversal:-

Left → Root → Right

- Visit the left subtree
- Visit the root node
- Visit the right subtree

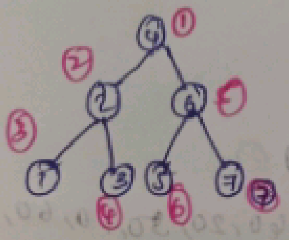


Inorder Traversal : 1, 2, 3, 4, 5, 6, 7.

Preorder Traversal:-

Root → Left → Right

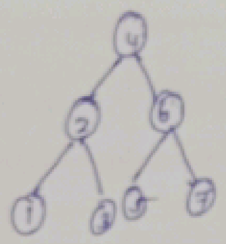
- Visit the root node.
- Visit the left subtree.
- Visit the right subtree.



Preorder Traversal:- 4, 2, 1, 3, 6, 5, 7.

Postorder Traversal:-

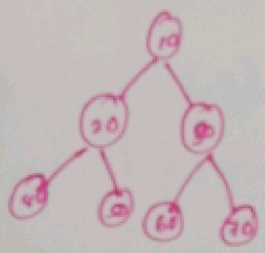
Left → Right - Root.



- Visit the left subtree.
- Visit the right subtree
- Visit the root node.

Postorder Traversal :- 1, 3, 2, 5, 7, 6, 4.

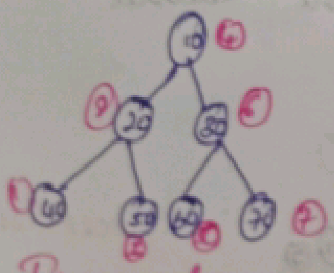
Ex: 2 :



Inorder Traversal:-

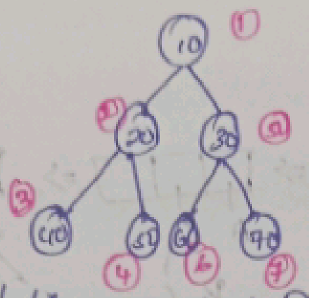
Left → Root → Right.

- Visit the left subtree.
- . Visit the right subtree.
- . Visit the root node.



Inorder Traversal : 40, 20, 50, 10, 60, 30, 70.

Preorder Traversal :-

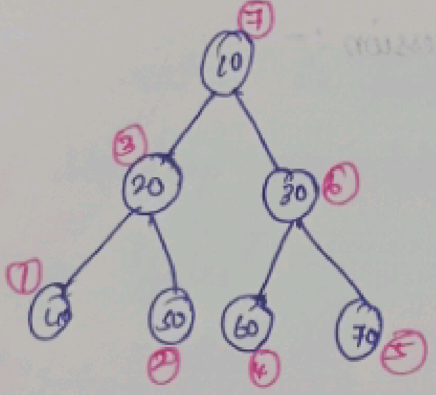


Left → Root → Right .

- Visit left subtree .
- Visit the root node .
- Visit the right subtree .

Preorder Traversal : 10, 20, 40, 50, 30, 60, 70 .

Postorder Traversal :-



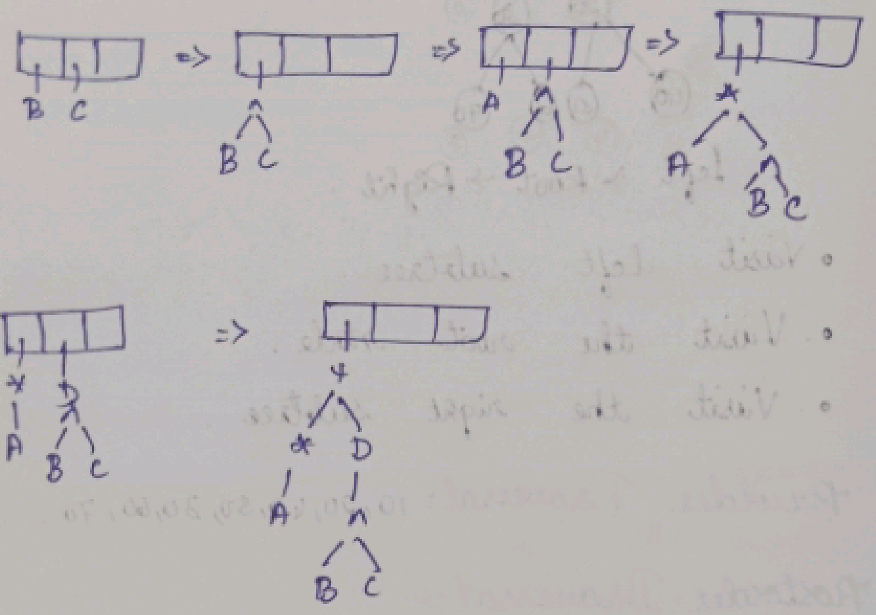
Left → Right → Root .

- Visit the left subtree
- Visit the right subtree .
- Visit the root node .

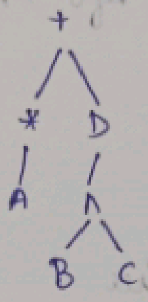
Postorder Traversal :- 40, 50, 20, 60, 70, 30, 10 .

Binary Tree Expression:-

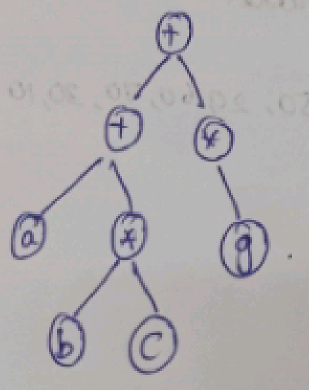
i) A*B^C+D.



Binary Tree Expression:-



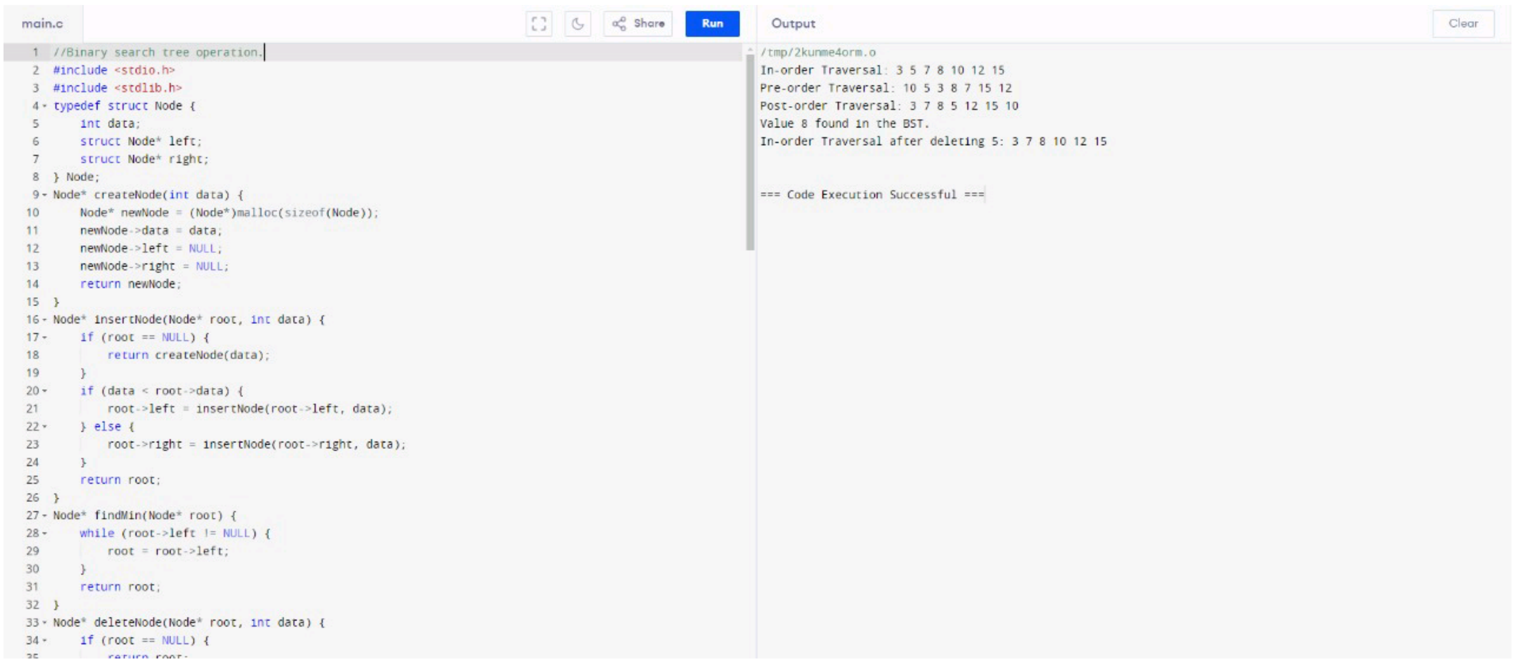
(ii) (a+b*c)+ (cd*e+f)*g)



^ → R-L
*,/ → L-R
-,+ → L-R.

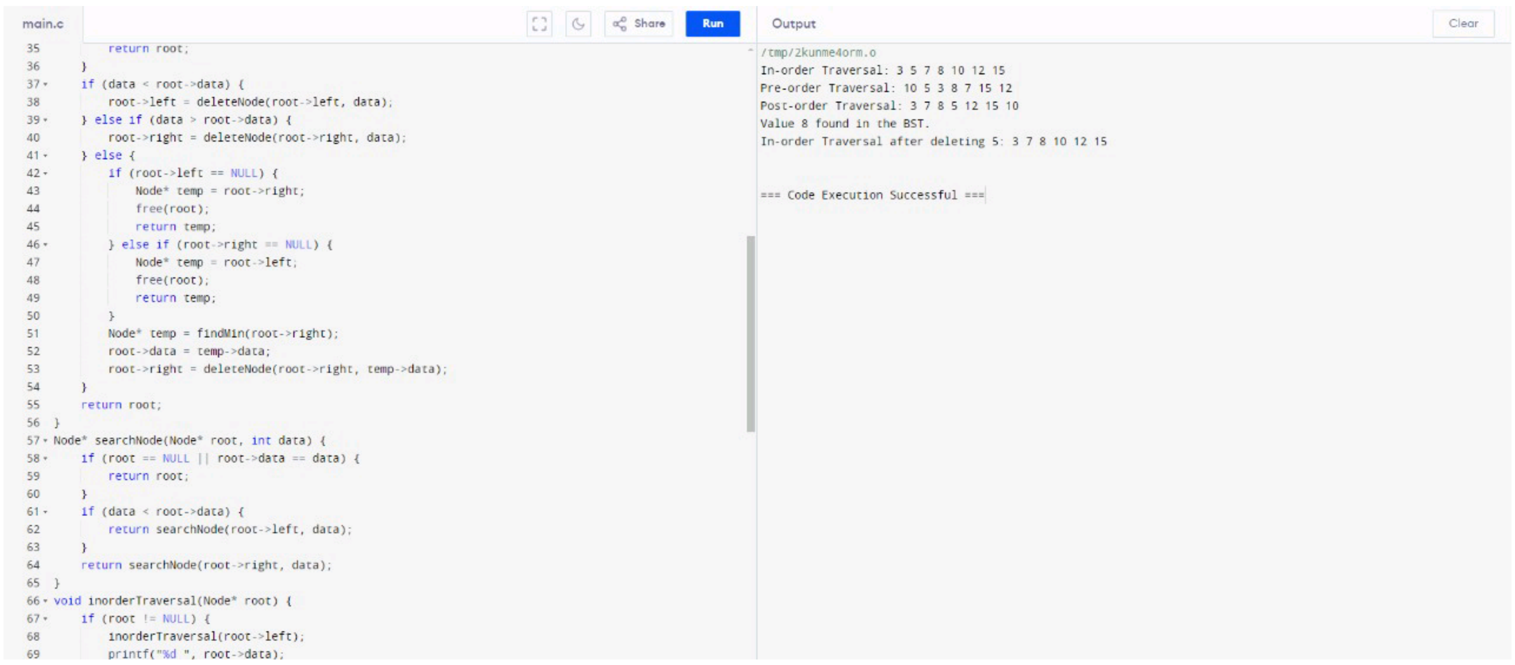
Binary Expression Tree.

1. Linear
Pseudo
1. 'data
where
element
2. 'tar
data.
3. Th
element
4. For
equals
5. If
6. If
it retu
Inputs
tar
Output



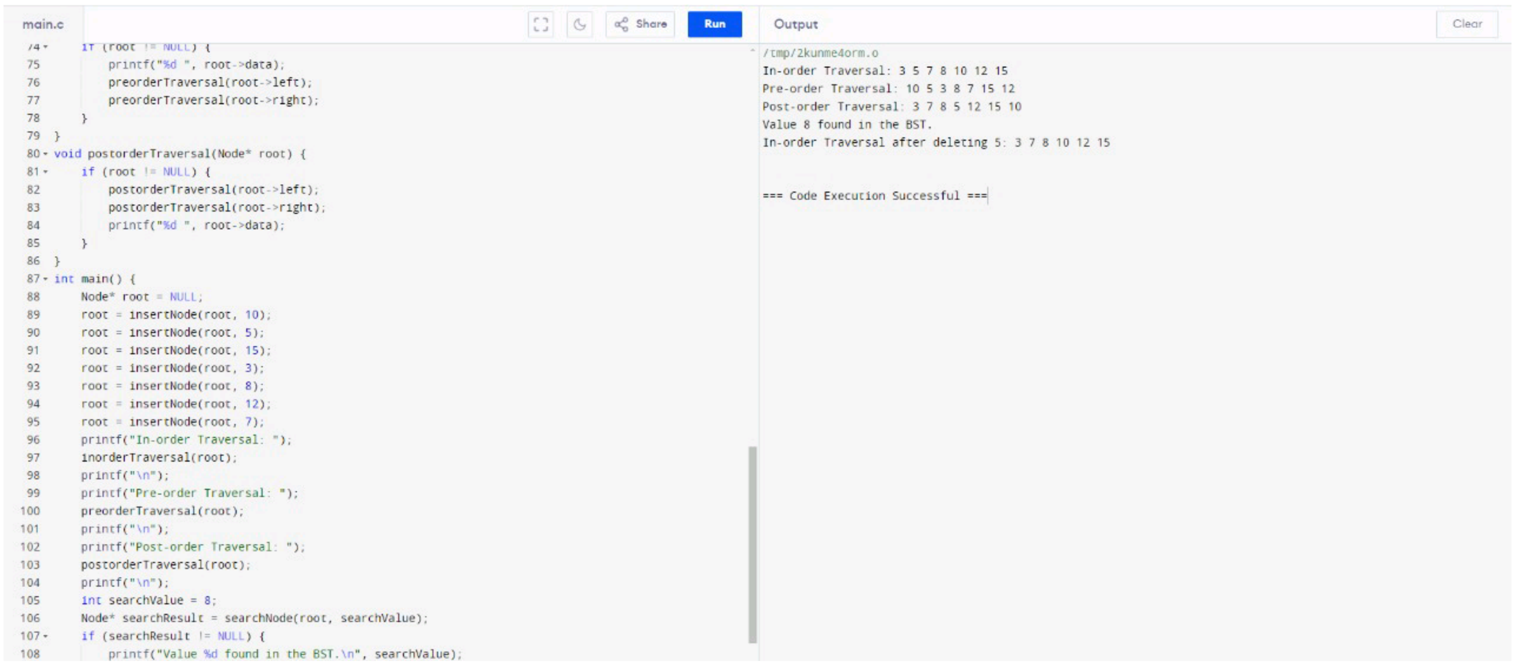
```c
//Binary search tree operation.
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
Node* insertNode(Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}
Node* findMin(Node* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}
Node* deleteNode(Node* root, int data) {
    if (root == NULL) {
```

```
/tmp/2kunme4orm.o
In-order Traversal: 3 5 7 8 10 12 15
Pre-order Traversal: 10 5 3 8 7 15 12
Post-order Traversal: 3 7 8 5 12 15 10
Value 8 found in the BST.
In-order Traversal after deleting 5: 3 7 8 10 12 15


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



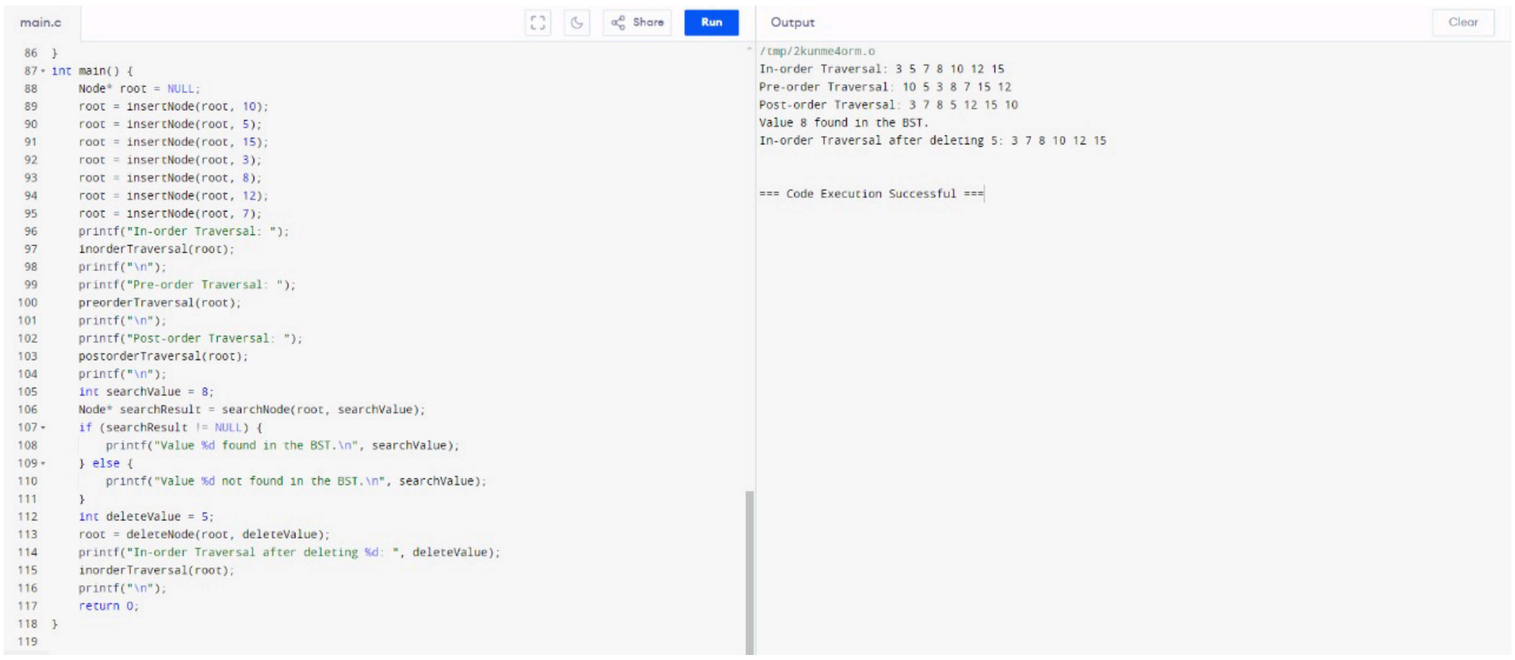
```c
        return root;
    }
    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        if (root->left == NULL) {
            Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            free(root);
            return temp;
        }
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
Node* searchNode(Node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }
    if (data < root->data) {
        return searchNode(root->left, data);
    }
    return searchNode(root->right, data);
}
void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
```

```
/tmp/2kunme4orm.o
In-order Traversal: 3 5 7 8 10 12 15
Pre-order Traversal: 10 5 3 8 7 15 12
Post-order Traversal: 3 7 8 5 12 15 10
Value 8 found in the BST.
In-order Traversal after deleting 5: 3 7 8 10 12 15


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



```c
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}
void postorderTraversal(Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}
int main() {
    Node* root = NULL;
    root = insertNode(root, 10);
    root = insertNode(root, 5);
    root = insertNode(root, 15);
    root = insertNode(root, 3);
    root = insertNode(root, 8);
    root = insertNode(root, 12);
    root = insertNode(root, 7);
    printf("In-order Traversal: ");
    inorderTraversal(root);
    printf("\n");
    printf("Pre-order Traversal: ");
    preorderTraversal(root);
    printf("\n");
    printf("Post-order Traversal: ");
    postorderTraversal(root);
    printf("\n");
    int searchValue = 8;
    Node* searchResult = searchNode(root, searchValue);
    if (searchResult != NULL) {
        printf("Value %d found in the BST.\n", searchValue);
```

```
/tmp/2kunme4orm.o
In-order Traversal: 3 5 7 8 10 12 15
Pre-order Traversal: 10 5 3 8 7 15 12
Post-order Traversal: 3 7 8 5 12 15 10
Value 8 found in the BST.
In-order Traversal after deleting 5: 3 7 8 10 12 15


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



```c
}
int main() {
    Node* root = NULL;
    root = insertNode(root, 10);
    root = insertNode(root, 5);
    root = insertNode(root, 15);
    root = insertNode(root, 3);
    root = insertNode(root, 8);
    root = insertNode(root, 12);
    root = insertNode(root, 7);
    printf("In-order Traversal: ");
    inorderTraversal(root);
    printf("\n");
    printf("Pre-order Traversal: ");
    preorderTraversal(root);
    printf("\n");
    printf("Post-order Traversal: ");
    postorderTraversal(root);
    printf("\n");
    int searchValue = 8;
    Node* searchResult = searchNode(root, searchValue);
    if (searchResult != NULL) {
        printf("Value %d found in the BST.\n", searchValue);
    } else {
        printf("Value %d not found in the BST.\n", searchValue);
    }
    int deleteValue = 5;
    root = deleteNode(root, deleteValue);
    printf("In-order Traversal after deleting %d: ", deleteValue);
    inorderTraversal(root);
    printf("\n");
    return 0;
}

```

```
/tmp/2kunme4orm.o
In-order Traversal: 3 5 7 8 10 12 15
Pre-order Traversal: 10 5 3 8 7 15 12
Post-order Traversal: 3 7 8 5 12 15 10
Value 8 found in the BST.
In-order Traversal after deleting 5: 3 7 8 10 12 15


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