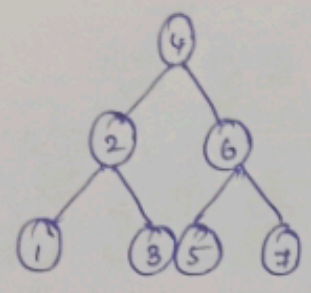


29/07/24

# 1. Binary Tree Traversal:-

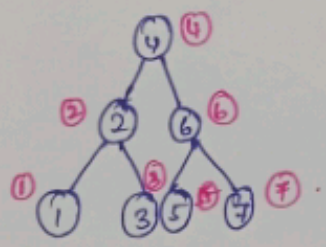
Ex-1:



## Inorder Traversal:-

Left → Root → Right

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

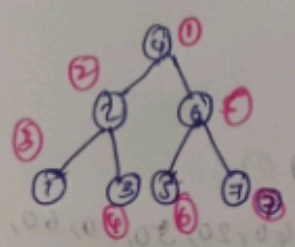


Inorder Traversal : 1, 2, 3, 4, 5, 6, 7.

## Preorder Traversal:-

Root → Left → Right

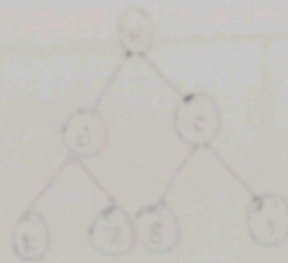
- Visit the root node.
- Visit the left subtree.
- Visit the right subtree.



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

## Postorder Traversal:-

Left → Right → Root



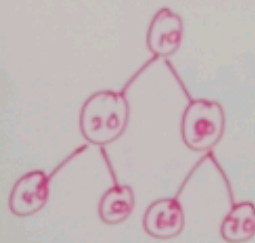
• Visit the left subtree

• Visit the right subtree

• Visit the root node

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

Ex: 2:



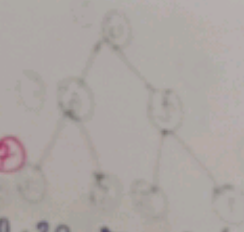
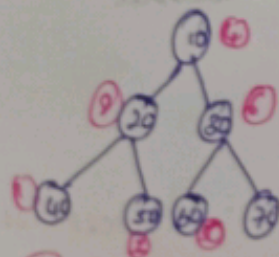
## Preorder Traversal:-

Left → Root → Right

• Visit the left subtree

• Visit the right subtree

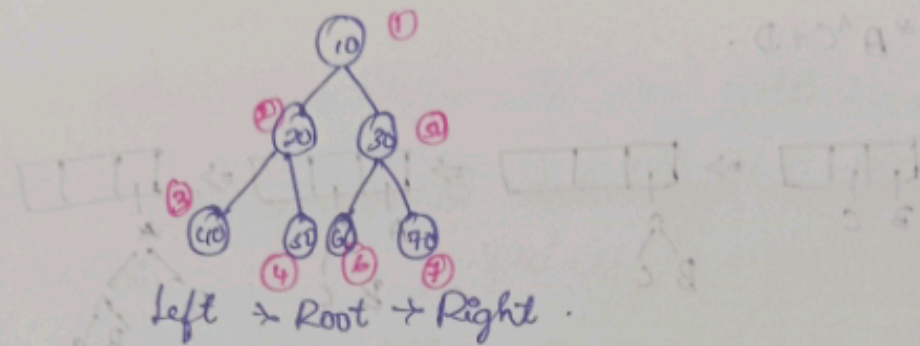
• Visit the root node



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



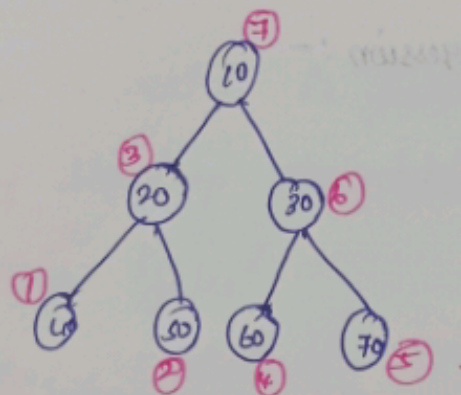
**Preorder Traversal:-**



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

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

**Postorder Traversal:-**

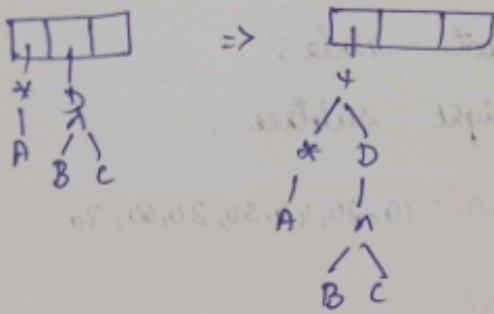
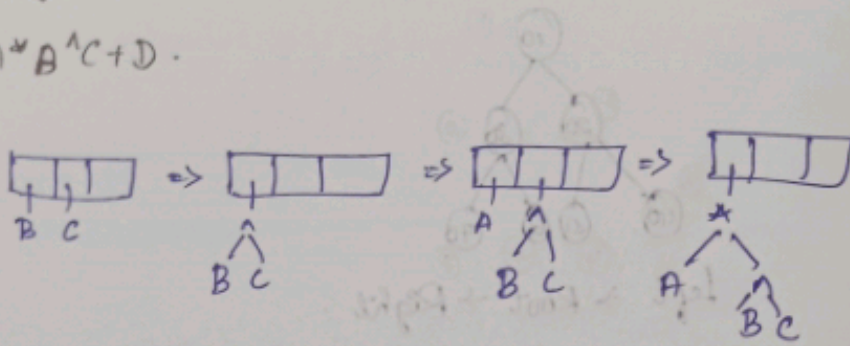


- Visit the left subtree .
- Visit the right subtree .
- Visit the root node .

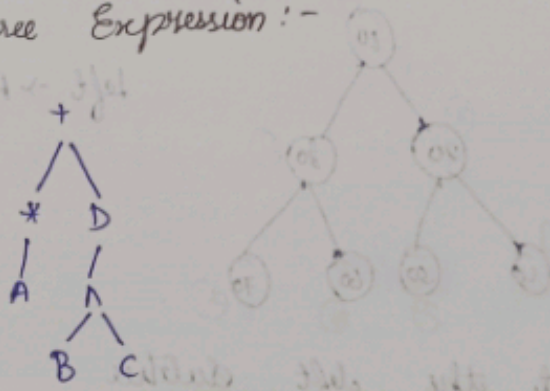
**Postorder Traversal:-** 40, 50, 20, 60, 70, 30, 10 .

## Binary Tree Expression:-

i)  $A * B^A C + D$ .



## Binary Tree Expression:-



(ii)  $(a + b * c) + (c * d * e + f) * g$



Binary Expression Tree.

$\wedge \rightarrow R-L$   
 $*, / \rightarrow L-R$   
 $-, + \rightarrow L-R$

main.c

Share

Run

```
1 //Binary search tree operation.
2 #include <stdio.h>
3 #include <stdlib.h>
4 typedef struct Node {
5     int data;
6     struct Node* left;
7     struct Node* right;
8 } Node;
9 Node* createNode(int data) {
10     Node* newNode = (Node*)malloc(sizeof(Node));
11     newNode->data = data;
12     newNode->left = NULL;
13     newNode->right = NULL;
14     return newNode;
15 }
16 Node* insertNode(Node* root, int data) {
17     if (root == NULL) {
18         return createNode(data);
19     }
20     if (data < root->data) {
21         root->left = insertNode(root->left, data);
22     } else {
23         root->right = insertNode(root->right, data);
24     }
25     return root;
26 }
27 Node* findMin(Node* root) {
28     while (root->left != NULL) {
29         root = root->left;
30     }
31     return root;
32 }
33 Node* deleteNode(Node* root, int data) {
34     if (root == NULL) {
35         return root;
36     }
```

Output

Clear

```
/tmp/2kumme4orm.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 ===
```

main.c

Share

Run

35

return root;

36

}

37

if (data < root->data) {

38

root->left = deleteNode(root->left, data);

39

} else if (data > root->data) {

40

root->right = deleteNode(root->right, data);

41

} else {

42

if (root->left == NULL) {

43

Node\* temp = root->right;

44

free(root);

45

return temp;

46

} else if (root->right == NULL) {

47

Node\* temp = root->left;

48

free(root);

49

return temp;

50

}

51

Node\* temp = findMin(root->right);

52

root->data = temp->data;

53

root->right = deleteNode(root->right, temp->data);

54

}

55

return root;

56

}

57

Node\* searchNode(Node\* root, int data) {

58

if (root == NULL || root->data == data) {

59

return root;

60

}

61

if (data < root->data) {

62

return searchNode(root->left, data);

63

}

64

return searchNode(root->right, data);

65

}

66

void inorderTraversal(Node\* root) {

67

if (root != NULL) {

68

inorderTraversal(root->left);

69

printf("%d ", root->data);

Output

Clear

/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 ===



main.c

Share

Run

```
74 * if (root != NULL) {
75     printf("%d ", root->data);
76     preorderTraversal(root->left);
77     preorderTraversal(root->right);
78 }
79 }
80 void postorderTraversal(Node* root) {
81 * if (root != NULL) {
82     postorderTraversal(root->left);
83     postorderTraversal(root->right);
84     printf("%d ", root->data);
85 }
86 }
87 int main() {
88     Node* root = NULL;
89     root = insertNode(root, 10);
90     root = insertNode(root, 5);
91     root = insertNode(root, 15);
92     root = insertNode(root, 3);
93     root = insertNode(root, 8);
94     root = insertNode(root, 12);
95     root = insertNode(root, 7);
96     printf("In-order Traversal: ");
97     inorderTraversal(root);
98     printf("\n");
99     printf("Pre-order Traversal: ");
100    preorderTraversal(root);
101    printf("\n");
102    printf("Post-order Traversal: ");
103    postorderTraversal(root);
104    printf("\n");
105    int searchValue = 8;
106    Node* searchResult = searchNode(root, searchValue);
107 * if (searchResult != NULL) {
108     printf("Value %d found in the BST.\n", searchValue);
```

Output

Clear

/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 ===

main.c

Share

Run

86 }  
87 - int main() {  
88 Node\* root = NULL;  
89 root = insertNode(root, 10);  
90 root = insertNode(root, 5);  
91 root = insertNode(root, 15);  
92 root = insertNode(root, 3);  
93 root = insertNode(root, 8);  
94 root = insertNode(root, 12);  
95 root = insertNode(root, 7);  
96 printf("In-order Traversal: ");  
97 inorderTraversal(root);  
98 printf("\n");  
99 printf("Pre-order Traversal: ");  
100 preorderTraversal(root);  
101 printf("\n");  
102 printf("Post-order Traversal: ");  
103 postorderTraversal(root);  
104 printf("\n");  
105 int searchValue = 8;  
106 Node\* searchResult = searchNode(root, searchValue);  
107 - if (searchResult != NULL) {  
108 printf("Value %d found in the BST.\n", searchValue);  
109 + } else {  
110 printf("Value %d not found in the BST.\n", searchValue);  
111 }  
112 int deleteValue = 5;  
113 root = deleteNode(root, deleteValue);  
114 printf("In-order Traversal after deleting %d: ", deleteValue);  
115 inorderTraversal(root);  
116 printf("\n");  
117 return 0;  
118 }  
119 }

Output

Clear

/tmp/2kumme4orm.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 ===