

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:1

Enter the data : 25

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:2

Enter a data to insert : 16

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:2

Enter a data to insert : 36

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:2

Enter your option:2
Enter a data to insert : 18

- 1.Create Root
- 2.Insert
- 3.Delete
- 4.Inorder
- 5.Preorder
- 6.Postorder
- 7.Exit

Enter your option:2
Enter a data to insert : 30

- 1.Create Root
- 2.Insert
- 3.Delete
- 4.Inorder
- 5.Preorder
- 6.Postorder
- 7.Exit

Enter your option:2
Enter a data to insert : 27

- 1.Create Root
- 2.Insert
- 3.Delete
- 4.Inorder
- 5.Preorder
- 6.Postorder
- 7.Exit

Enter your option:4

Inorder traversal:
16 18 25 27 30 36

- 1.Create Root
- 2.Insert
- 3.Delete
- 4.Inorder
- 5.Preorder
- 6.Postorder

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:5

Preorder traversal:

25 16 18 36 30 27

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:6

Postorder traversal:

18 16 27 30 36 25

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:3

Enter a data to delete : 27

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:4

Inorder traversal:

16 18 25 30 36

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:5

Preorder traversal:

25 16 18 36 30

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

Enter your option:6

Postorder traversal:

18 16 30 36 25

1.Create Root
2.Insert
3.Delete
4.Inorder
5.Preorder
6.Postorder
7.Exit

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 struct node {
4     int data;
5     struct node *left, *right;
6 };
7 struct node *root = NULL;
8 struct node *create(int item)
9 {
10     struct node *temp = (struct node *)malloc(sizeof(struct node));
11     temp->data = item;
12     temp->left = temp->right = NULL;
13     return temp;
14 }
15 struct node *insert(struct node *node, int val)
16 {
17     if (node == NULL)
18         return create(val);
19     if (val < node->data)
20         node->left = insert(node->left, val);
21     else
22         node->right = insert(node->right, val);
23     return node;
24 }
25 struct node *minimum(struct node *node)
26 {
27     struct node *current = node;
28     while (current && current->left != NULL)
29         current = current->left;
30     return current;
31 }
32 struct node *deletenode(struct node *root, int val) {
33     if (root == NULL)
34         return root;
35     if (val < root->data)
36         root->left = deletenode(root->left, val);
37     else if (val > root->data)
38         root->right = deletenode(root->right, val);
39     else {
40         if (root->left == NULL)
41         {
42             struct node *temp = root->right;

```

```

42     struct node *temp = root->right;
43     free(root);
44     return temp;
45 }
46 else if (root->right == NULL)
47 {
48     struct node *temp = root->left;
49     free(root);
50     return temp;
51 }
52 struct node *temp = minimum(root->right);
53 root->data = temp->data;
54 root->right = deletenode(root->right, temp->data);
55 }
56 return root;
57 }
58 void inorder(struct node *root) {
59     if(root!=NULL){
60         inorder(root->left);
61         printf("%d \t ", root->data);
62         inorder(root->right);
63     }
64 }
65 void preorder(struct node *root) {
66     if(root!=NULL){
67         printf("%d \t ", root->data);
68         preorder(root->left);
69         preorder(root->right);
70     }
71 }
72 void postorder(struct node *root) {
73     if(root!=NULL){
74         postorder(root->left);
75         postorder(root->right);
76         printf("%d \t ", root->data);
77     }
78 }
79 int main() {
80     int ch,val;
81     do
82     {

```

```

83 {
84     printf("\n1.Create Root \n2.Insert\n3.Delete\n4.Inorder\n5.Preorder\n6.Postorder\n7.Exit \n");
85     printf("\nEnter your option:");
86     scanf("%d",&ch);
87     switch(ch)
88     {
89         case 1:
90             printf("Enter the data :\t");
91             scanf("%d",&val);
92             root = create(val);
93             break;
94         case 2:
95             printf("Enter a data to insert :\t ");
96             scanf("%d",&val);
97             root=insert(root,val);
98             break;
99         case 3:
100             printf("Enter a data to delete\t : ");
101             scanf("%d",&val);
102             root=deletenode(root,val);
103             break;
104         case 4:
105             printf("\nInorder traversal:\n ");
106             if(root==NULL)
107                 printf("tree is empty");
108             else
109                 inorder(root);
110             break;
111         case 5:
112             printf("\nPreorder traversal:\n ");
113             if(root==NULL)
114                 printf("tree is empty");
115             else
116                 preorder(root);
117             break;
118         case 6:
119             printf("\nPostorder traversal:\n ");
120             if(root==NULL)
121                 printf("tree is empty");
122             else
123                 postorder(root);
124             break;

```

```

92     scanf("%d",&val);
93     root = create(val);
94     break;
95 case 2:
96     printf("Enter a data to insert :\t ");
97     scanf("%d",&val);
98     root=insert(root,val);
99     break;
100 case 3:
101     printf("Enter a data to delete\t : ");
102     scanf("%d",&val);
103     root=deletenode(root,val);
104     break;
105 case 4:
106     printf("\nInorder traversal:\n ");
107     if(root==NULL)
108         printf("tree is empty");
109     else
110         inorder(root);
111     break;
112 case 5:
113     printf("\nPreorder traversal:\n ");
114     if(root==NULL)
115         printf("tree is empty");
116     else
117         preorder(root);
118     break;
119 case 6:
120     printf("\nPostorder traversal:\n ");
121     if(root==NULL)
122         printf("tree is empty");
123     else
124         postorder(root);
125     break;
126 case 7: exit(0);
127 default :
128     printf("Error");
129 }
130 }while(ch!=7);
131 return 0;
132 }

```