NAME: ADWAIT S PURAO
UID: 2021300101
EXP NO. :5
AIM: To implement Insertion , Deletion and Inorder , Preorder , Postorder Traversal in a Binary Search Tree

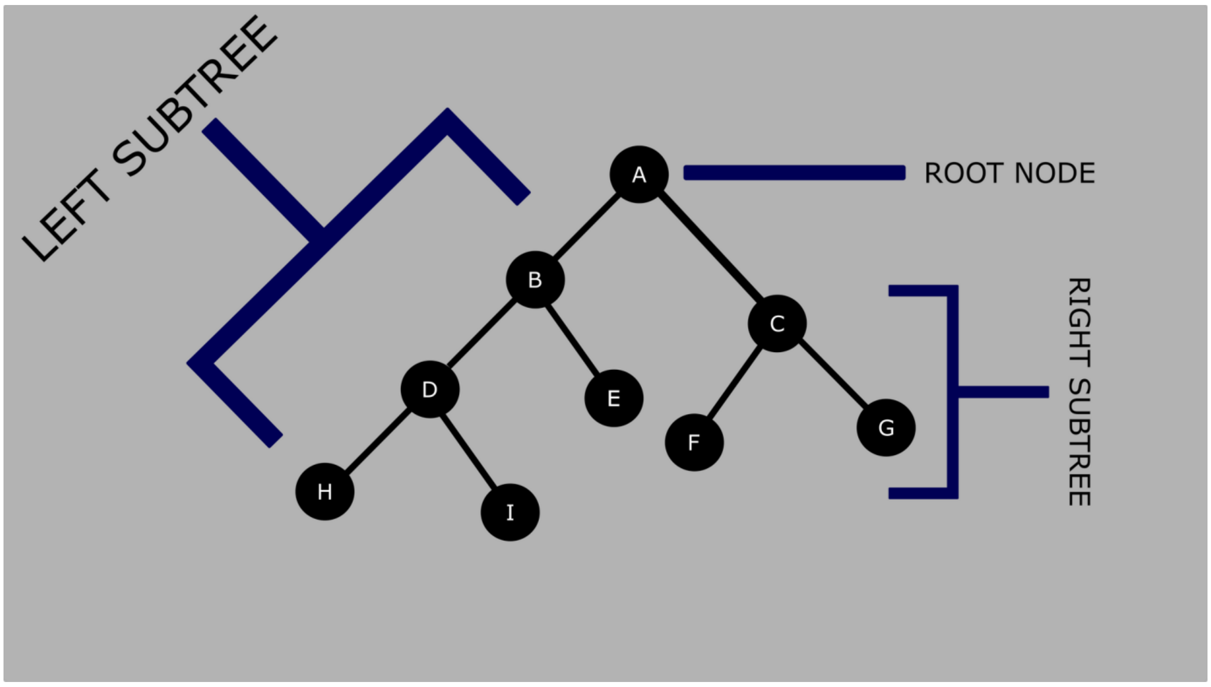
THEORY:

What is a tree?

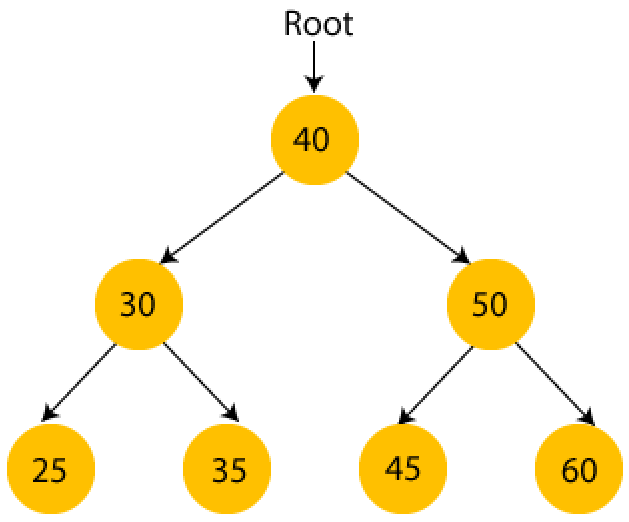
A tree is a kind of data structure that is used to represent the data in hierarchical form. It can be defined as a collection of objects or entities called as nodes that are linked together to simulate a hierarchy. Tree is a non-linear data structure as the data in a tree is not stored linearly or sequentially.

What is a Binary Search tree?

A binary search tree follows some order to arrange the elements. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.



Let's understand the concept of Binary search tree with an example.



In the above figure, we can observe that the root node is 40, and all the nodes of the left subtree are smaller than the root node, and all the nodes of the right subtree are greater than the root node.

Similarly, we can see the left child of root node is greater than its left child and smaller than its right child. So, it also satisfies the property of binary search tree. Therefore, we can say that the tree in the above image is a binary search tree.

Advantages of Binary search tree

Searching an element in the Binary search tree is easy as we always have a hint that which subtree has the desired element.

As compared to array and linked lists, insertion and deletion operations are faster in BST.

Creation of a binary search tree

Now, let's see the creation of binary search tree using an example.

Suppose the data elements are - 45, 15, 79, 90, 10, 55, 12, 20, 50

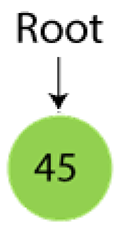
First, we have to insert 45 into the tree as the root of the tree.

Then, read the next element; if it is smaller than the root node, insert it as the root of the left subtree, and move to the next element.

Otherwise, if the element is larger than the root node, then insert it as the root of the right subtree.

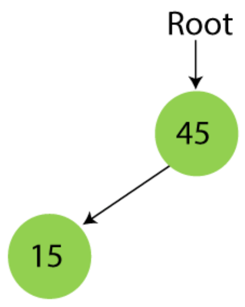
Now, let's see the process of creating the Binary search tree using the given data element. The process of creating the BST is shown below -

Step 1 - Insert 45.



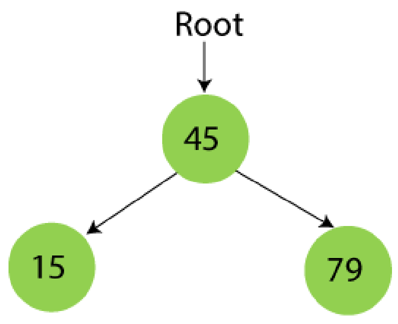
Step 2 - Insert 15.

As 15 is smaller than 45, so insert it as the root node of the left subtree.



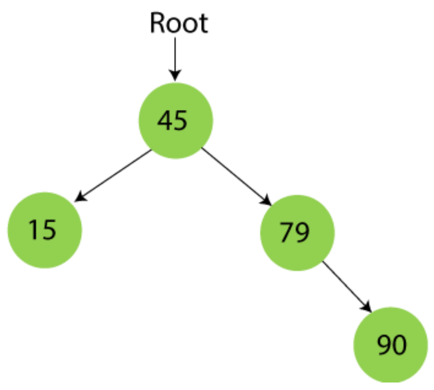
Step 3 - Insert 79.

As 79 is greater than 45, so insert it as the root node of the right subtree.



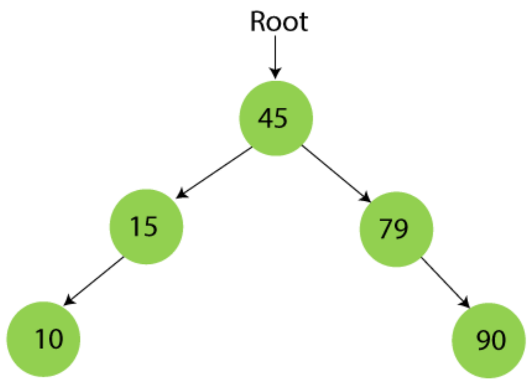
Step 4 - Insert 90.

90 is greater than 45 and 79, so it will be inserted as the right subtree of 79.



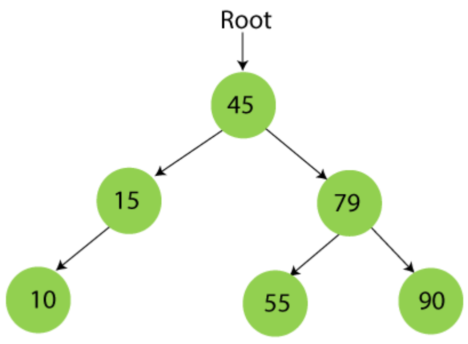
Step 5 - Insert 10.

10 is smaller than 45 and 15, so it will be inserted as a left subtree of 15.



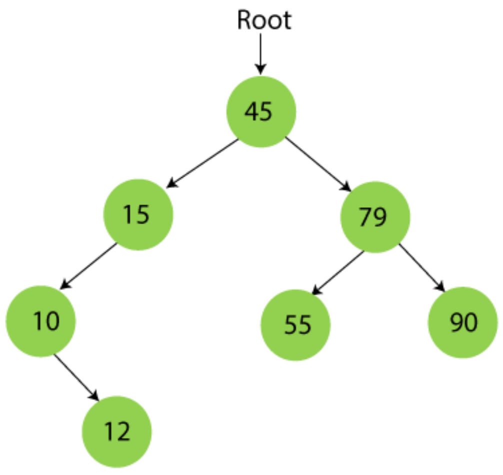
Step 6 - Insert 55.

55 is larger than 45 and smaller than 79, so it will be inserted as the left subtree of 79.



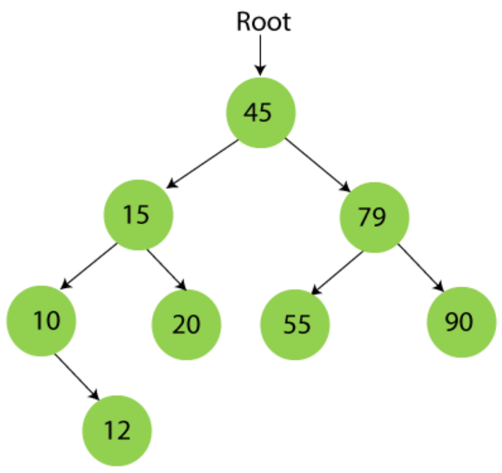
Step 7 - Insert 12.

12 is smaller than 45 and 15 but greater than 10, so it will be inserted as the right subtree of 10.



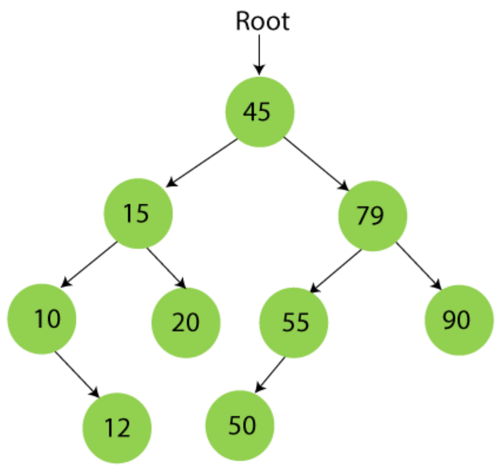
Step 8 - Insert 20.

20 is smaller than 45 but greater than 15, so it will be inserted as the right subtree of 15.



Step 9 - Insert 50.

50 is greater than 45 but smaller than 79 and 55. So, it will be inserted as a left subtree of 55.



Searching in Binary search tree

Searching means to find or locate a specific element or node in a data structure. In Binary search tree, searching a node is easy because elements in BST are stored in a specific order. The steps of searching a node in Binary Search tree are listed as follows -

First, compare the element to be searched with the root element of the tree.

If root is matched with the target element, then return the node's location.

If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.

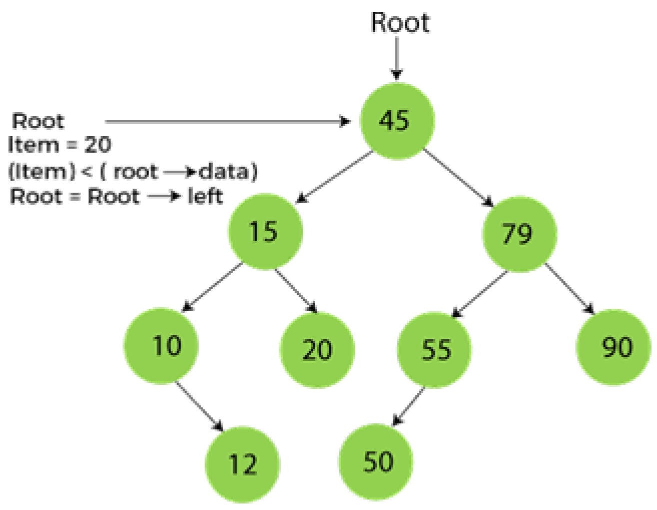
If it is larger than the root element, then move to the right subtree.

Repeat the above procedure recursively until the match is found.

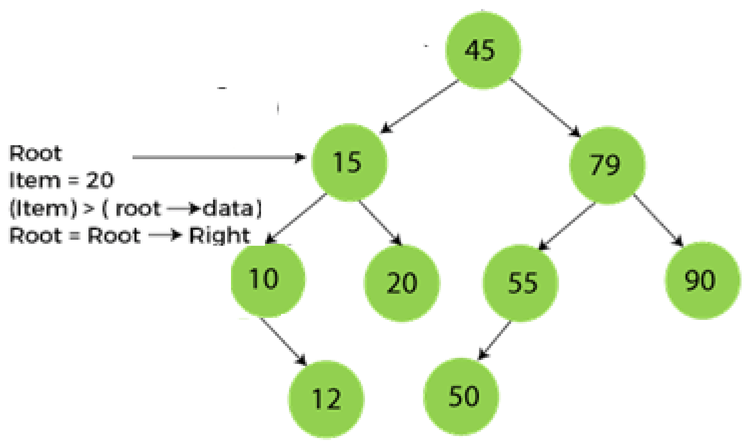
If the element is not found or not present in the tree, then return NULL.

Now, let's understand the searching in binary tree using an example. We are taking the binary search tree formed above. Suppose we have to find node 20 from the below tree.

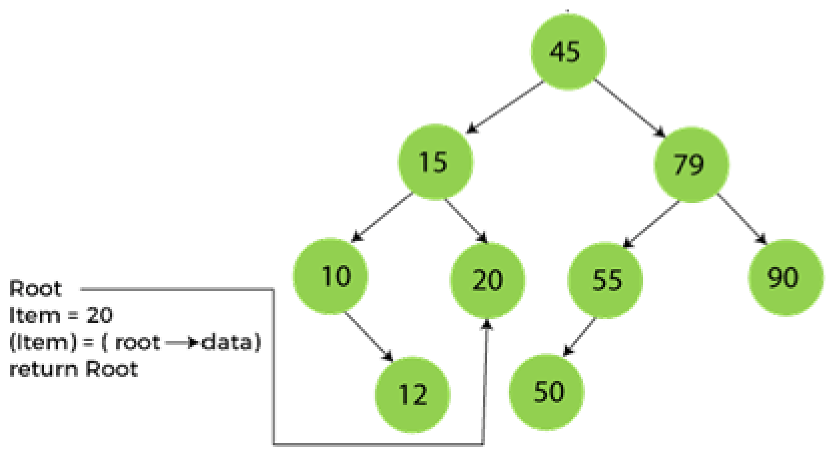
Step1:



Step2:



Step3:



Deletion in Binary Search tree

In a binary search tree, we must delete a node from the tree by keeping in mind that the property of BST is not violated. To delete a node from BST, there are three possible situations occur -

The node to be deleted is the leaf node, or,

The node to be deleted has only one child, and,

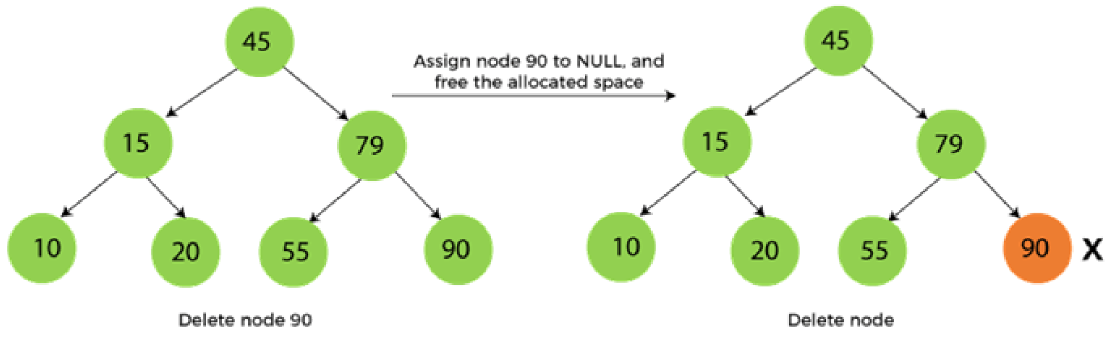
The node to be deleted has two children

We will understand the situations listed above in detail.

When the node to be deleted is the leaf node

It is the simplest case to delete a node in BST. Here, we have to replace the leaf node with NULL and simply free the allocated space.

We can see the process to delete a leaf node from BST in the below image. In below image, suppose we have to delete node 90, as the node to be deleted is a leaf node, so it will be replaced with NULL, and the allocated space will free.

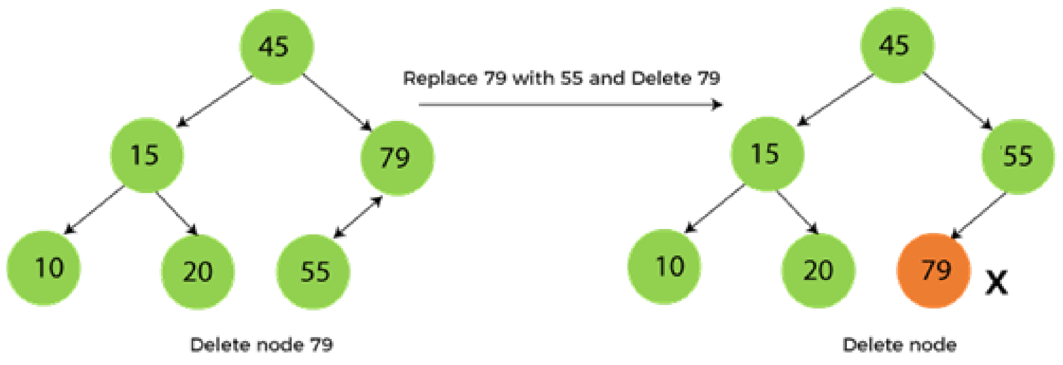


When the node to be deleted has only one child

In this case, we have to replace the target node with its child, and then delete the child node. It means that after replacing the target node with its child node, the child node will now contain the value to be deleted. So, we simply have to replace the child node with NULL and free up the allocated space.

We can see the process of deleting a node with one child from BST in the below image. In the below image, suppose we have to delete the node 79, as the node to be deleted has only one child, so it will be replaced with its child 55.

So, the replaced node 79 will now be a leaf node that can be easily deleted.



When the node to be deleted has two children

This case of deleting a node in BST is a bit complex among other two cases. In such a case, the steps to be followed are listed as follows -

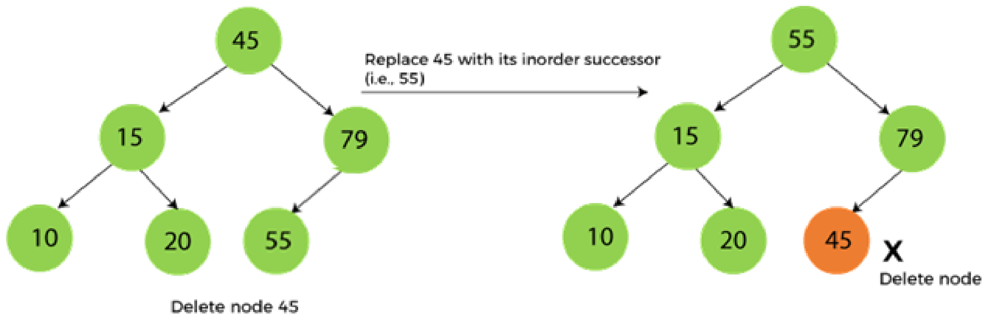
First, find the inorder successor of the node to be deleted.

After that, replace that node with the inorder successor until the target node is placed at the leaf of tree.

And at last, replace the node with NULL and free up the allocated space.

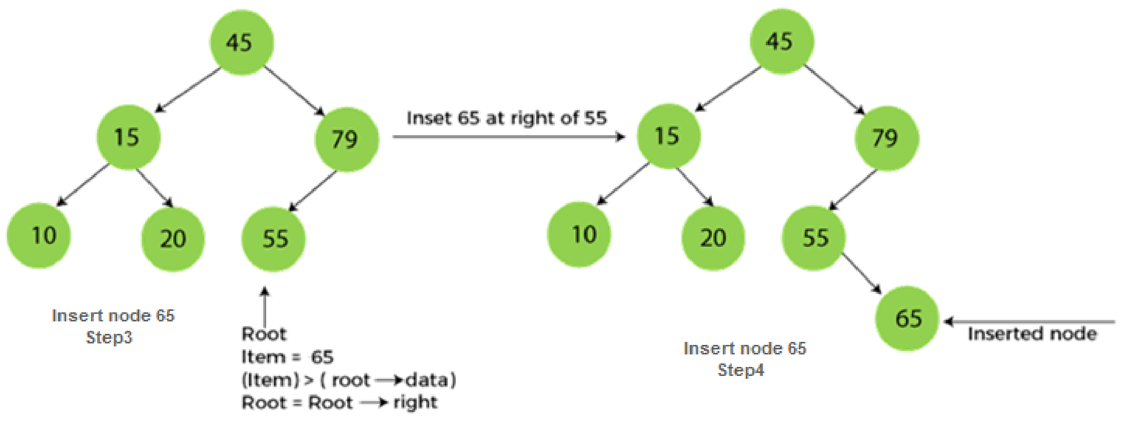
The inorder successor is required when the right child of the node is not empty. We can obtain the inorder successor by finding the minimum element in the right child of the node.

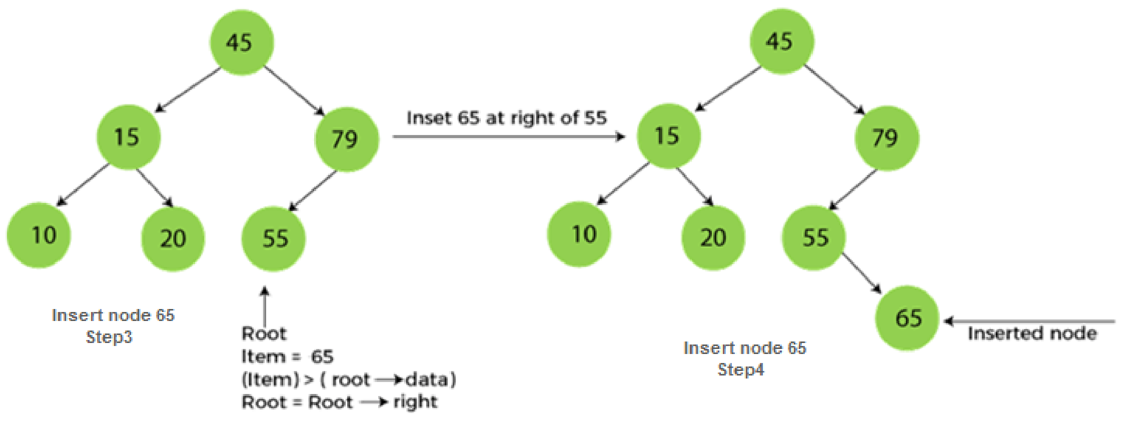
We can see the process of deleting a node with two children from BST in the below image. In the below image, suppose we have to delete node 45 that is the root node, as the node to be deleted has two children, so it will be replaced with its inorder successor. Now, node 45 will be at the leaf of the tree so that it can be deleted easily.



Insertion in Binary Search tree

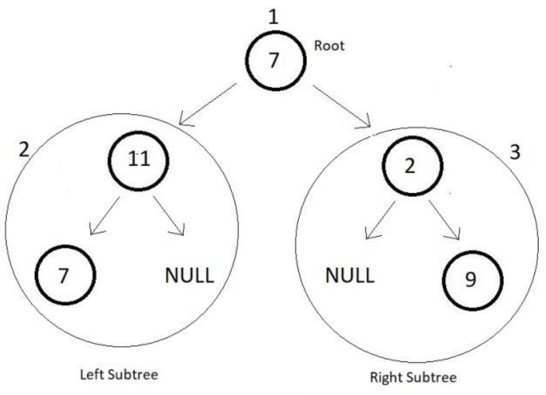
A new key in BST is always inserted at the leaf. To insert an element in BST, we have to start searching from the root node; if the node to be inserted is less than the root node, then search for an empty location in the left subtree. Else, search for the empty location in the right subtree and insert the data. Insert in BST is similar to searching, as we always have to maintain the rule that the left subtree is smaller than the root, and right subtree is larger than the root.





PreOrder Traversal in a Binary Tree:

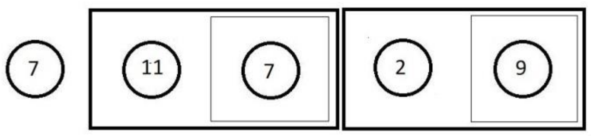
So in this traversal technique, the first node you start with is the root node. And thereafter you visit the left subtree and then the right subtree. Taking the above example, I'll mark your order of traversal as below. You first visit section 1, then 2, and then 3.



Now, this was to give you a general idea of the traverse in a binary tree using PreOrder Traversal. Each time you get a tree, you first visit its root node, and then move to its left subtree, and then to the right.

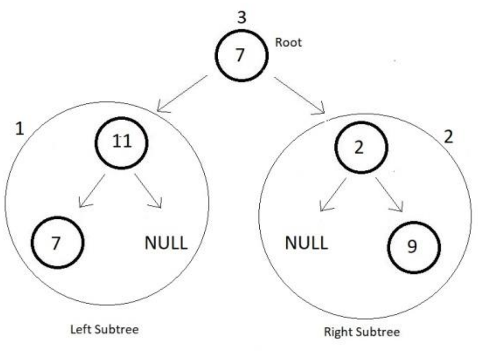
So, here you first visit the root node element 7 and then move to the left subtree. The left subtree in itself is a tree with root node 11. So, you visit that and move further to the left subtree of this subtree. There you see a single element 7, you visit that, and then move to the right subtree which is a NULL. So, you're finished with the left subtree of the main tree. Now, you move to the right subtree which has element 2 as its node. And then a NULL to its left and element 9 to its right.

So basically, you recursively visit each subtree in the same order. And your final traversal order is:



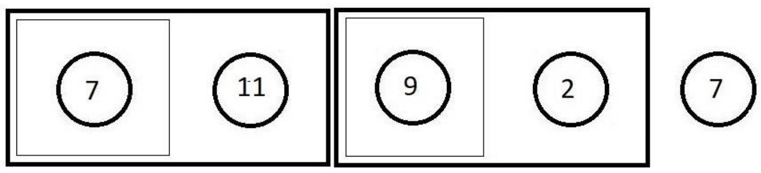
PostOrder Traversal in a Binary Tree:

In this traversal technique, things are quite opposite to the PreOrder traversal. Here, you first visit the left subtree, and then the right subtree. So, the last node you'll visit is the root node. Taking the same above example, I'll mark your order of traversal as below. You first visit section 1, then 2, and then 3.



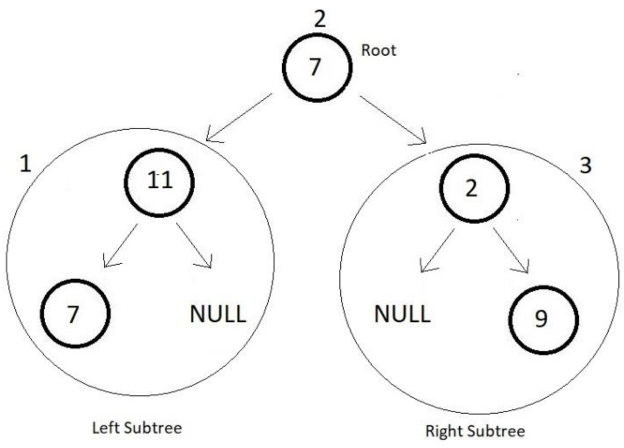
This was again a general idea of you traverse in a binary tree using PostOrder Traversal. Each time you get a tree, you first visit its left subtree, and then its right subtree, and then move to its root node.

I expect you to write the flow of the traversal in PostOrder yourself, and let me know if you could. We would anyway see them in detail.



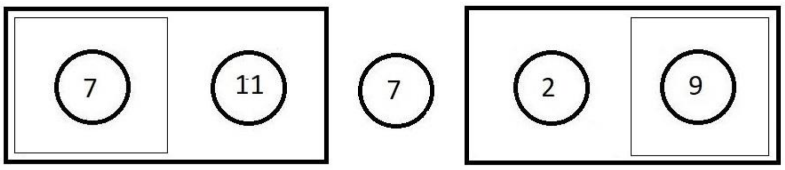
InOrder Traversal in a Binary Tree:

In this traversal technique, we simply start with the left subtree, that is you first visit the left subtree, and then go to the root node and then you'll visit the right subtree. Taking the same above example, I'll mark your order of traversal as below. You first visit section 1, then 2, and then 3.



This was a general idea of you traverse in a binary tree using InOrder Traversal. Each time you get a tree, you first visit its left subtree, and then its root node, and then finally its right subtree.

I expect you to write the flow of the traversal in InOrder yourself, and let me know if you could.



ALGORITHM:

```
1)Struct node
Data members
Int data
Pointer to right node
Pointer to left node
2) Function void Preorder(struct node*root)
        If root is not equal to null
                Print root->data
                Call Preorder recursively to left of the root
                Call Preorder recursively to right of the root
3) Function void Postorder(struct node*root)
        If root is not equal to null
                Call Postorder recursively to left of the root
                Call Postorder recursively to right of the root
                Print root->data
4) Function void Inorder(struct node*root)
        If root is not equal to null
        Call Postorder recursively to left of the root
        Print root->data
        Call Postorder recursively to right of the root

5) Function struct node*Creation(int val)
    Allocate memory for node ptr
    Set ptr->data =data
    Set ptr->left =NULL
    Set ptr->right=NULL
    Return ptr
6) Function void Insertion(int value,struct node*root)
    Set a temporary pointer prev to NULL
    Run a while loop until root is not equal to NULL
        Store root in prev
        If value is equal to root->data
            Print value you entered is already present
            Return
        Else if value is less than root->data
            Go to left of root
        Else
            Go to right of root
    Create a node new_node with the help of Creation function
    If value is less than prev->data
        Store new_node in prev->left
    Else
        Store new_node in prev->right

7)Function int InorderSuccessor(struct node*root)
```

```
    Set a temporary node temp to root
    Run a while loop until temp and temp->left is not equal to NULL
        Make temp equal to temp->left
    Return temp->data

8) Function int Search(struct node*root,int value)
    Run a while loop until root is not equal to NULL
        If value is equal to root value
            Return 1
        Else if value is less than root value
            Go to left of root
        Else
            Go to right of root
    Return 0
9) Function struct node*Deletion(struct node*root,int data)
    If data is less than root value
        Call the function Deletion to root->left
    Else if data is greater than root value
        Call the function Deletion to root->right
    Else
        If root->left==NULL
            Store root->right in node ptr
            Free root
            Return ptr
        Else if root->right ==NULL
            Store root->left in node ptr
            Free root
            Return ptr
        Else
            Call the function InorderSuccessor(root->right) and store the value in temp
            Set root->data=temp
            Call the function Deletion(root->right,temp) recursively and store value in root->right
    Return root
10) Main Function
    Data members
    Int ch,flag=0
    Set struct node*root to NULL
    Do while(flag!=1)
        Display the menu
        Take input of ch
        Switch(ch)
            Case 1:
                Take input of value you want to insert
                If root equal to NULL
                    Call function Creation and  store result in root
                Else
                    Call function Insertion(value,root)
                Call function Inorder(root)
                Break
            Case 2:
                 Take input the value you want to delete
                If Search function returns 1
                    Print Element found
                    Call function Deletion(root,value) and store result in root
                Else
                    Print element not found

                Call function Inorder(root)
                Break

            Case 3:
                Call function Inorder(root)
                Break

            Case 4:
                Call function Preorder(root)
                Break

            Case 5:
                Call function Postorder(root)
                Break

            Case 6:
                Set Flag to 1
                Break
            Default:
                Print Invalid input
                Break
```

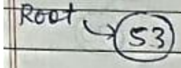
PROBLEM SOLVING ON CONCEPT:

Adwait. S. Purao
2021300101
B2

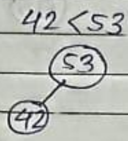
Date
Page

Binary Search Tree
Values to be inserted
53, 42, 18, 11, 36, 24, 10, 62, 85, 9, 90, 52
Insertion:

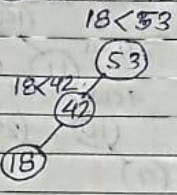
① Num = 53 (Root)



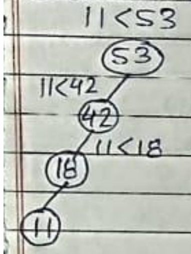
② Num: 42



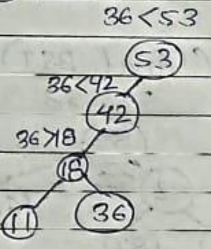
③ Num: 18



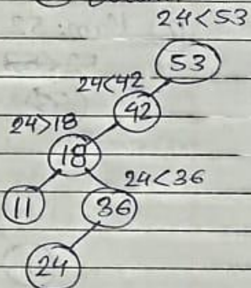
④ Num: 11



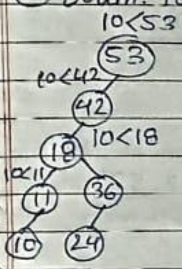
⑤ Num: 36



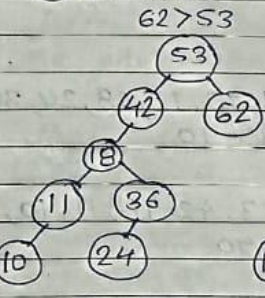
⑥ Num: 24



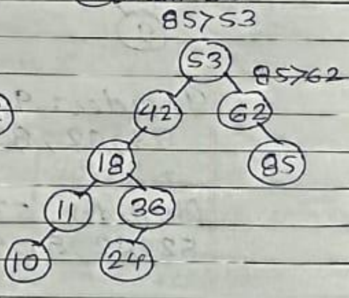
⑦ Num: 10



⑧ Num: 62



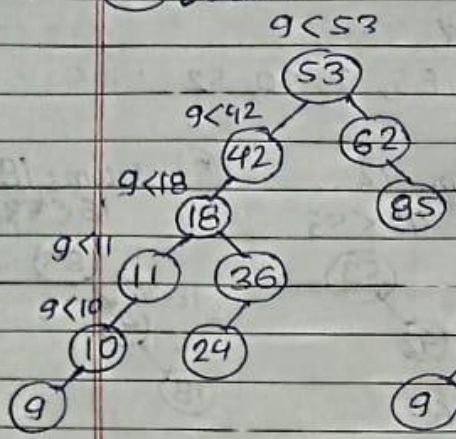
⑨ Num: 85



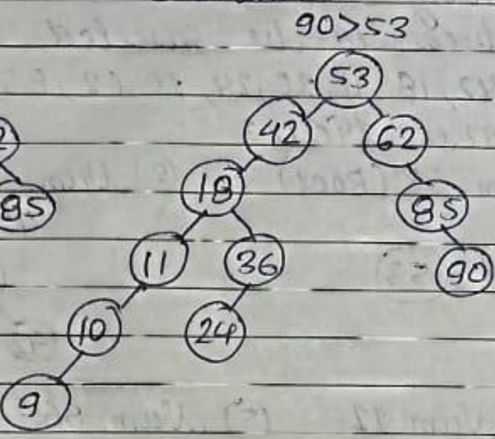
Adwait . S . Purao
2021300101
B2

Date
Page

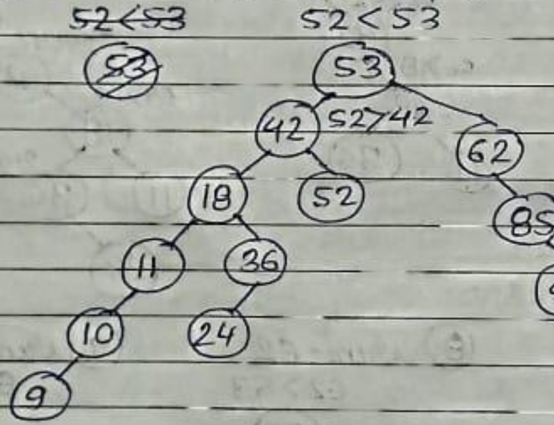
(10) Num: 9



(11) Num: 90



(12) Num: 52 (Final BST)



Inorder: 9, 10, 11, 18, 24, 36, 42, 52, 53, 62, 85, 90

Preorder: 53, 42, 18, 11, 10, 9, 36, 24, 52, 62, 85, 90

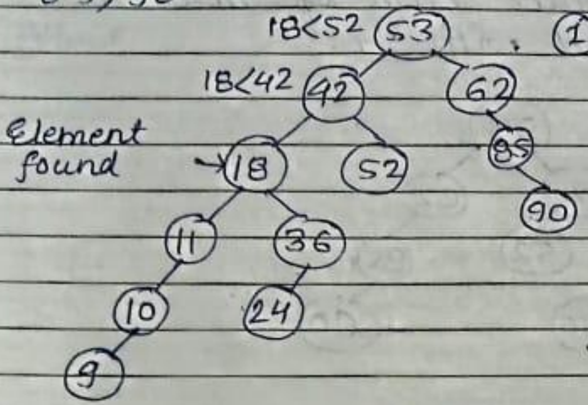
Postorder: 9, 10, 11, 24, 36, 18, 52, 42, 90, 85, 62, 53

Adwait S. Purao
2021300107
B2

Date
Page

①

Deletion: Element: 18
Inorder: 9, 10, 11, 18, 24, 36, 42, 52, 53, 62, 85, 90

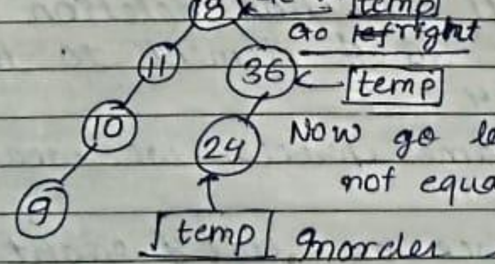


① First find the element

• As element is found, check the left & right subtrees of the element.
• We can see neither left nor right subtrees of given element are NULL
• Now find Inorder Successor

Inorder Successor: Element which succeeds a given element in Inorder Traversal.
→ It is the leftmost element of the right subtree of the given element

⇒ Consider the tree from 18:



Take a temporary pointer temp pointing to 18

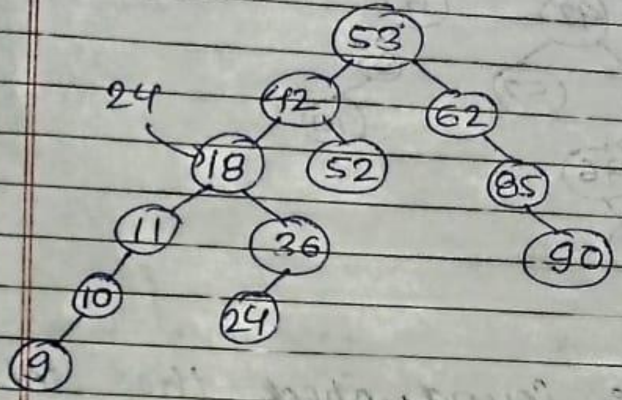
Now go left until temp→left not equal to null

Inorder successor found

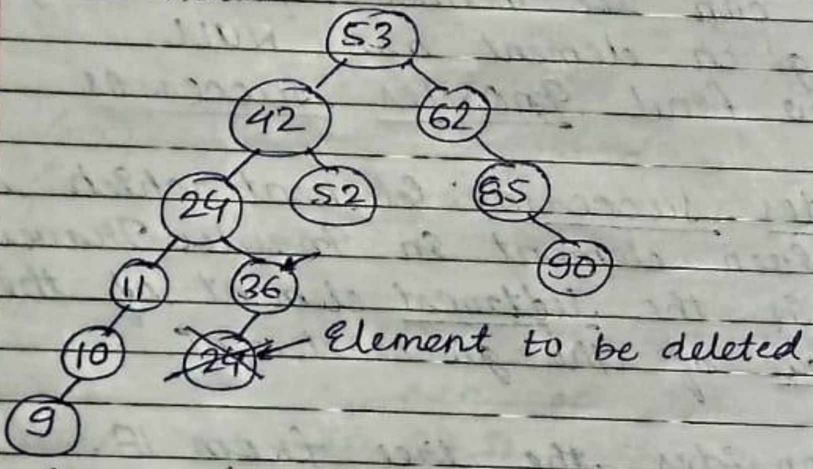
Adwait S. Purao
2021300101
B2

Inorder successor = 24

⇒ Now replace Data of node to be deleted with 24.



View now:



→ Now call function deletion with root = 36 & element to be deleted = 24

→ By the same process we would find 24

→ We can see as left & right both of the node to be deleted are NULL. Delete it

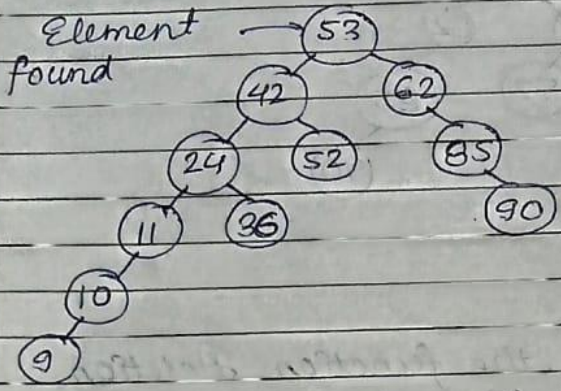
Adwait S. Purao
202130010I
B2

Date
Page

③

Deletion: Element=53

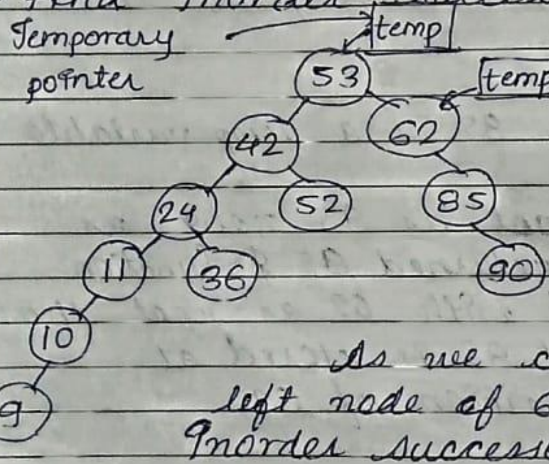
Inorder: 9, 10, 11, 24, 36, 42, 52, 53, 62, 85, 90



1) search for the element

→ Left & right subtrees of element to be deleted are not null, not even one of them is null.

→ Find Inorder successor now.



1) First go right
2) Now go left until temp→left is not null

As we can see the left node of 62 is null, Inorder successor is 62

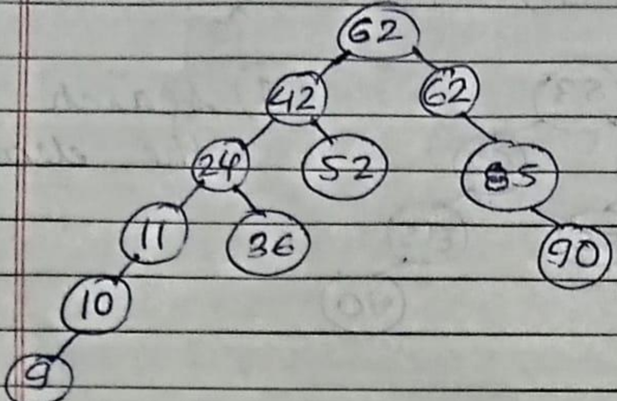
→ Now replace 53 (Node to be deleted) with 62

Adwait .S. Purao
2021300101
B2

Date
Page

(4)

Tree now:



Now call the function Deletion with root = 62 & element to be deleted = 62

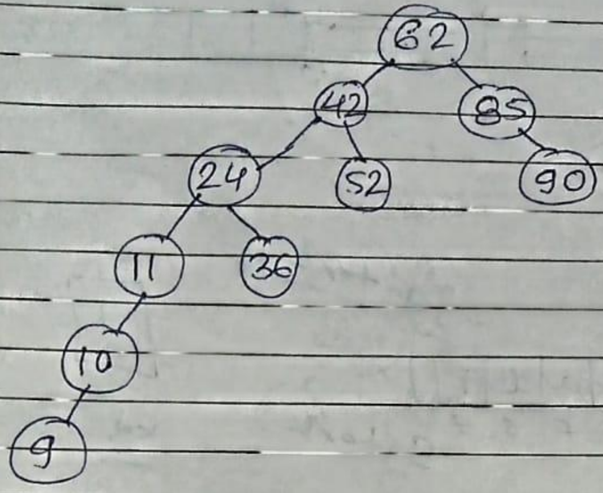
As the element to be deleted is root only, we have found the element.

→ Now store 85 in a temp variable
→ Free 62
→ Return root as 85 now; as we have returned 85 & function was called with 62 as root, the position of 85 is stored at previous position of 62.

Adwait S. Purao
2021300101
B2

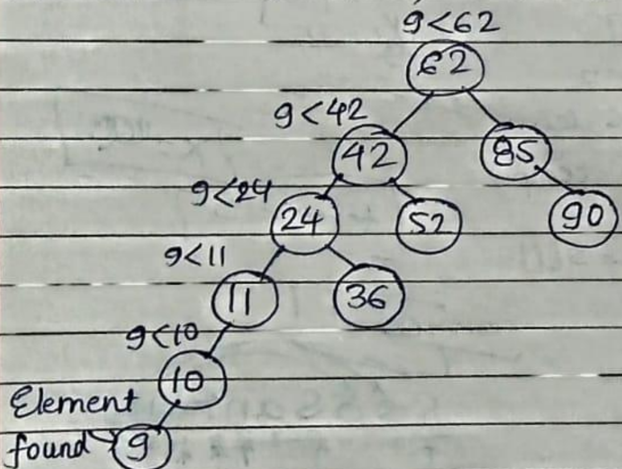
Date
Page

View now:



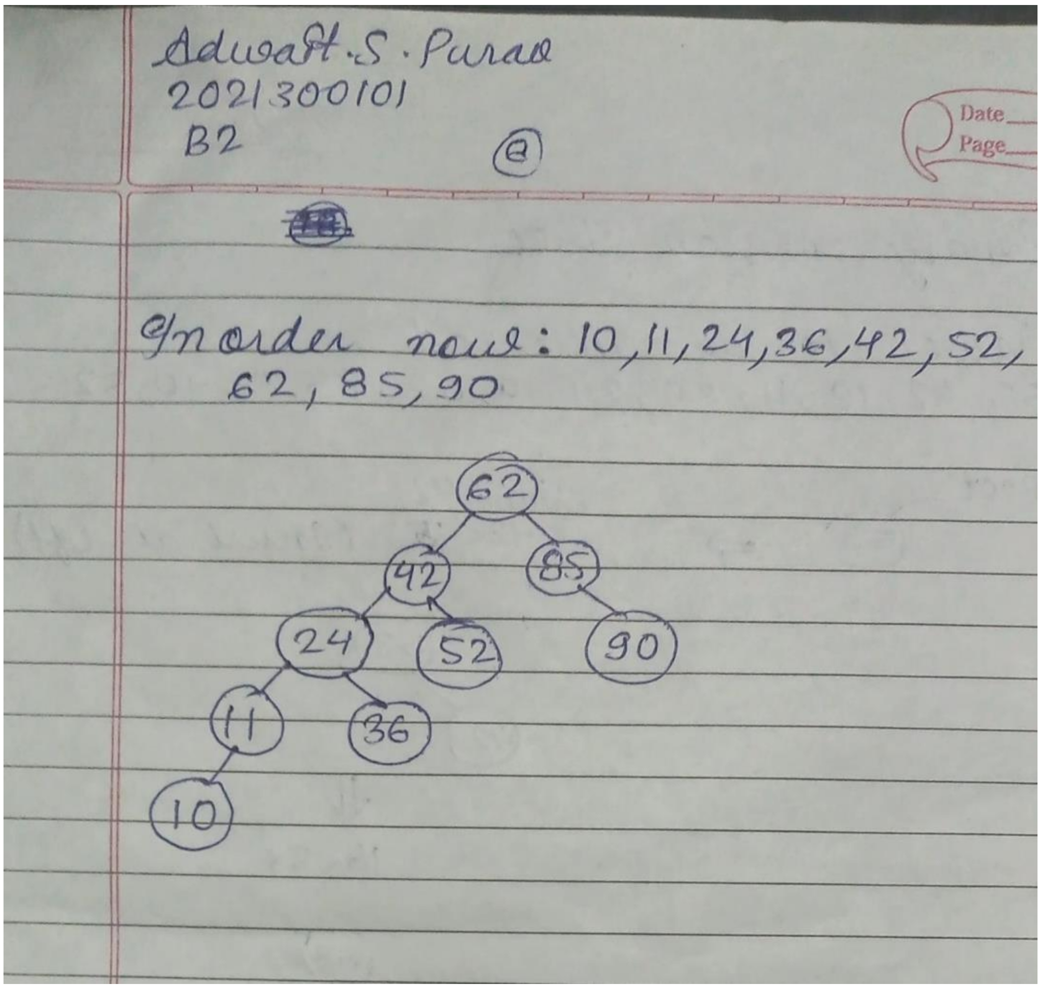
Inorder now: 9, 10, 11, 24, 36, 42, 52, 62, 85, 90

Deletion: Element = 9
Inorder: 9, 10, 11, 24, 36, 42, 52, 62, 85, 90



1) search for element

→ As we can see both left node & right node of the element to be deleted are null.
→ Directly delete 9.



Adwait.S.Purao
2021300101
B2

Date
Page

a

~~[illegible]~~

Inorder now: 10, 11, 24, 36, 42, 52,
62, 85, 90

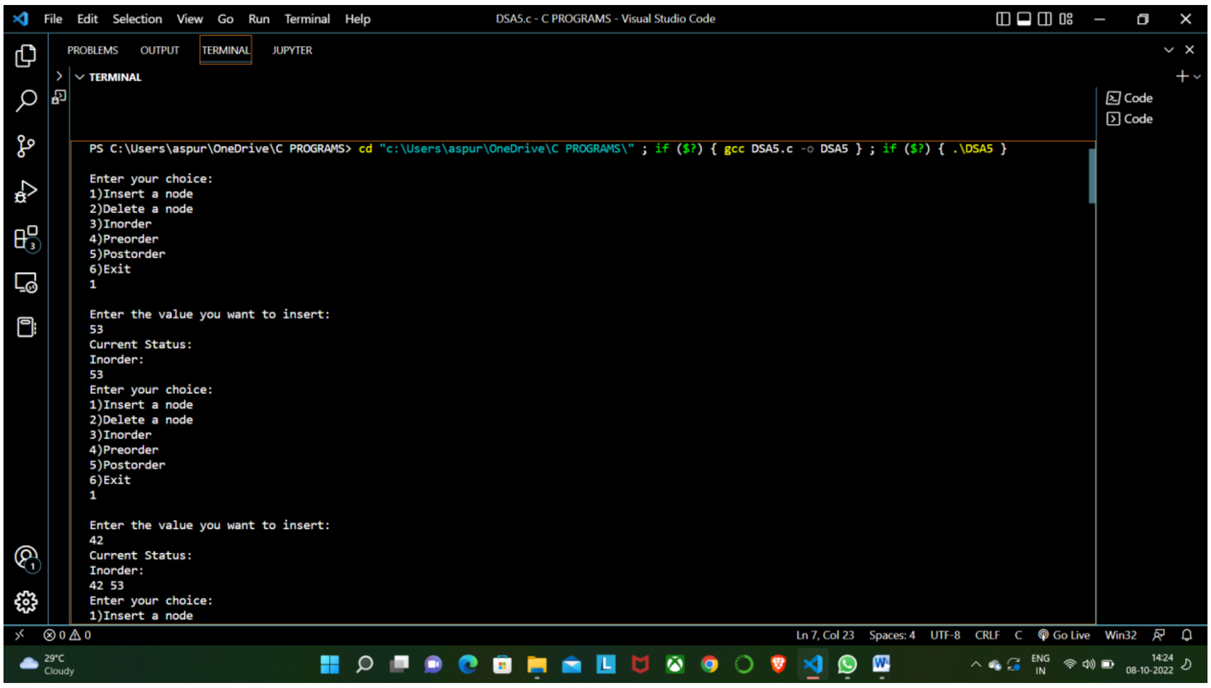
CODE:

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

struct node{
    struct node*left;
    int data;
    struct node*right;
};

//Postorder is VLR
void Preorder(struct node*root){
//Call the function recursively until we have reached the end
    if(root!=NULL){
    printf("%d ",root->data);//Print data
    Preorder(root->left);//Recur the function to the left of the current node
    Preorder(root->right);//Recur the function to the right of the current
node
    }
```

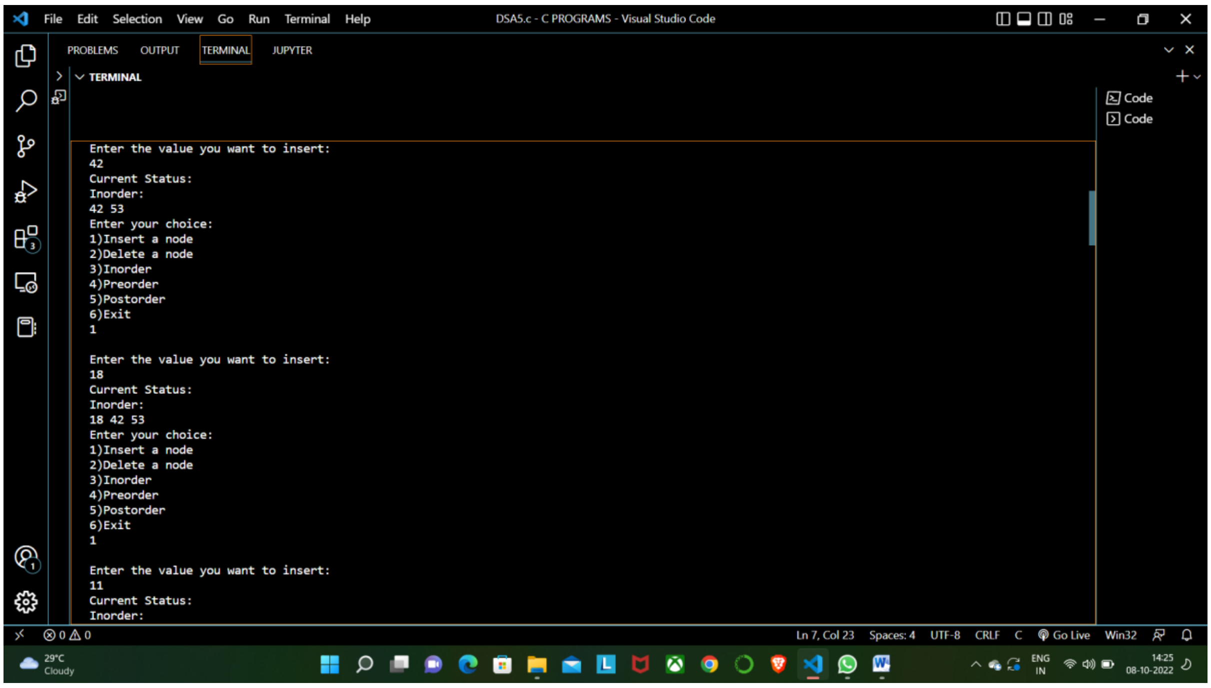
```c
}

//Postorder is LRV
void Postorder(struct node*root){
//Call the function recursively until we have reached the end
    if(root!=NULL){
        Postorder(root->left);//Recur the function to the left of the current
node
        Postorder(root->right);//Recur the function to the right of the
current node
        printf("%d ",root->data);//Print data
    }
}

//Inorder is LVR
void Inorder(struct node*root){
    //Call the function recursively until we have reached the end
    if(root!=NULL){
    Inorder(root->left);//Recur the function to the left of the current node
    printf("%d ",root->data);//Print data
    Inorder(root->right);//Recur the function to the right of the current node
    }
}

struct node*Creation(int val){
    struct node*ptr=(struct node*)malloc(sizeof(struct node));
    ptr->data=val;
    ptr->left=NULL;
    ptr->right=NULL;
    return ptr;
}

void Insertion(int value,struct node*root){
    //For storing the leaf node
    struct node*prev=NULL;
    //While loop is for searching for the node hence it goes while it's not
equal to null
    while(root!=NULL){
        prev=root;
        //If value is already present get out of function
        if(value ==root->data){
            printf("The value you entered is already present in the tree.So
can't insert!\n");
            return ;
        }
        //If value is less than root value go to the left
        else if(value<root->data){
            root=root->left;
        }
        //If value is greater than root value go to the right
        else{
            root=root->right;
        }
    }
    //After appropriate leaf node has been found create a new node
    struct node*new_node =Creation(value);

    //If value is less than value of leaf node insert to the left
    if(value<prev->data){
        prev->left=new_node;
    }

    //If value is greater than value of leaf node insert to the right
    else{
        prev->right=new_node;
    }

}

//Inorder successor is the element which succeeds the current element in
Inorder Traversal
//Pass in the node to the right of the current node
int InorderSuccessor(struct node*root){
    struct node*temp=root;
    //Inorder successor is the leftmost node of the right subtree of the
current node
    while(temp && temp->left!=NULL){
        temp=temp->left;
    }
    return temp->data;
}

int Search(struct node*root,int value){
    while(root!=NULL){
        // If value is found stop
        if(value==root->data){
            return 1;
        }
        //If value is less than root value go to the left
        else if(value<root->data){
            root=root->left;
        }
        //If value is greater than root value go to the right
        else{
            root=root->right;
        }
    }
    return 0;
}
struct node*Deletion(struct node*root,int data){

    //If data is less than data of root node go to the left subtree
    if(data < root->data){
        root->left=Deletion(root->left,data);
    }

    //If data is greater than data of root node go to the right subtree
    else if(data > root->data){
        root->right=Deletion(root->right,data);
    }

    //Steps after reaching the node to be deleted
    else{

        //If the left subtree of the node to be deleted is NULL
        //free the node and replace it by the node at the right of it
        if(root->left == NULL){
            struct node*ptr=root->right;
            free(root);
            return ptr;
        }

        //If the right subtree of the node to be deleted is NULL
        //free the node and replace it by the node at the left of it
        else if(root->right == NULL){
            struct node*ptr=root->left;
            free(root);
            return ptr;
        }

        //If the node to the right and left are not null
        else{
            int temp=InorderSuccessor(root->right);//Store the value of the
Inorder successor in a temporary variable
            root->data=temp;//Store the value of Inorder successor in root
            root->right=Deletion(root->right,temp);//Delete the Inorder
successor of root
        }

    }
    return root;

}
int main(){
```

```c
    int ch;
    int flag=0;
    struct node*root=NULL;
    do{
        printf("\nEnter your choice:\n");
        printf("1)Insert a node\n2)Delete a
node\n3)Inorder\n4)Preorder\n5)Postorder\n6)Exit\n");
        scanf("%d",&ch);

        switch(ch){
            case 1:
                {
                    int value;
                    printf("\nEnter the value you want to insert:\n");
                    scanf("%d",&value);
                    if(root==NULL){
                        root=Creation(value);
                    }
                    else{
                        Insertion(value,root);
                    }
                    printf("Current Status:\n");
                    printf("Inorder:\n");
                    Inorder(root);
                    break;
                }
            case 2:
                {
                    int value;
                    printf("Enter the value you want to Delete:\n");
                    scanf("%d",&value);
                    if(Search(root,value)==1){
                        printf("Element found!Now deleting %d\n",value);
                        root=Deletion(root,value);
                    }
                    else{
                        printf("Element not found!\n");
                    }
                    printf("Current Status:\n");
                    printf("Inorder:\n");
                    Inorder(root);
                    printf("\n");
                    break;
                }
            case 3:
                {
                    printf("\nInorder:\n");
                    Inorder(root);
                    break;
                }
            case 4:
                {
                    printf("\nPreorder:\n");
                    Preorder(root);
                    break;
                }
            case 5:
                {
                    printf("\nPostorder:\n");
                    Postorder(root);
                    break;
                }
            case 6:
                {
                    printf("\nProgram finished!\n");
                    flag=1;
                    break;
                }
                default:
                {
                    printf("Invalid input\n");
                    break;
                }
        }
    }while(flag!=1);

    return 0;
}
```

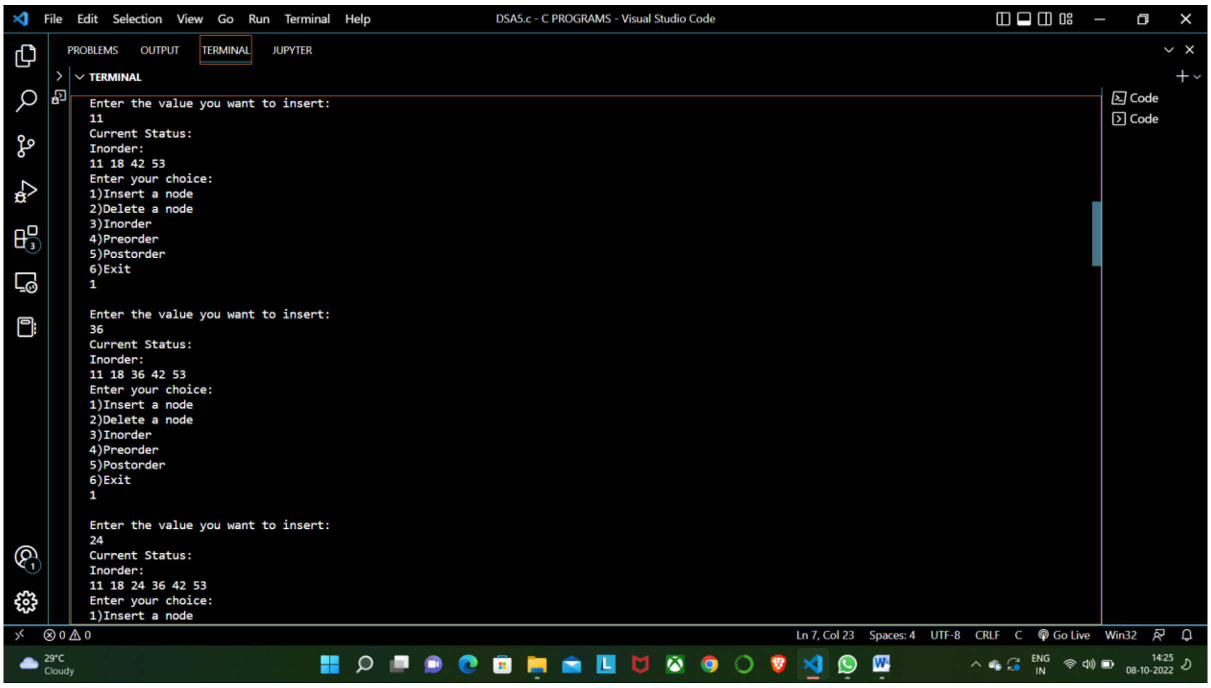
OUTPUT SCREENSHOT:



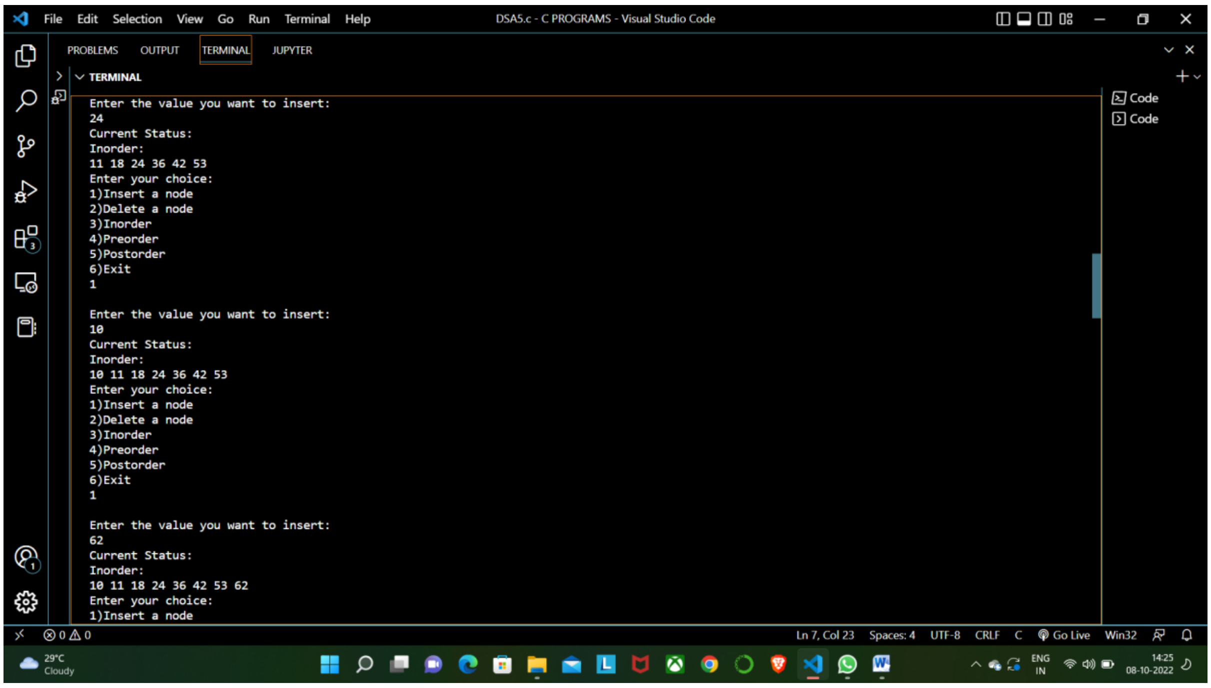
```
PS C:\Users\aspur\OneDrive\C PROGRAMS> cd "c:\Users\aspur\OneDrive\C PROGRAMS\" ; if ($?) { gcc DSA5.c -o DSA5 } ; if ($?) { .\DSA5 }

Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
53
Current Status:
Inorder:
53
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
42
Current Status:
Inorder:
42 53
Enter your choice:
1)Insert a node
```



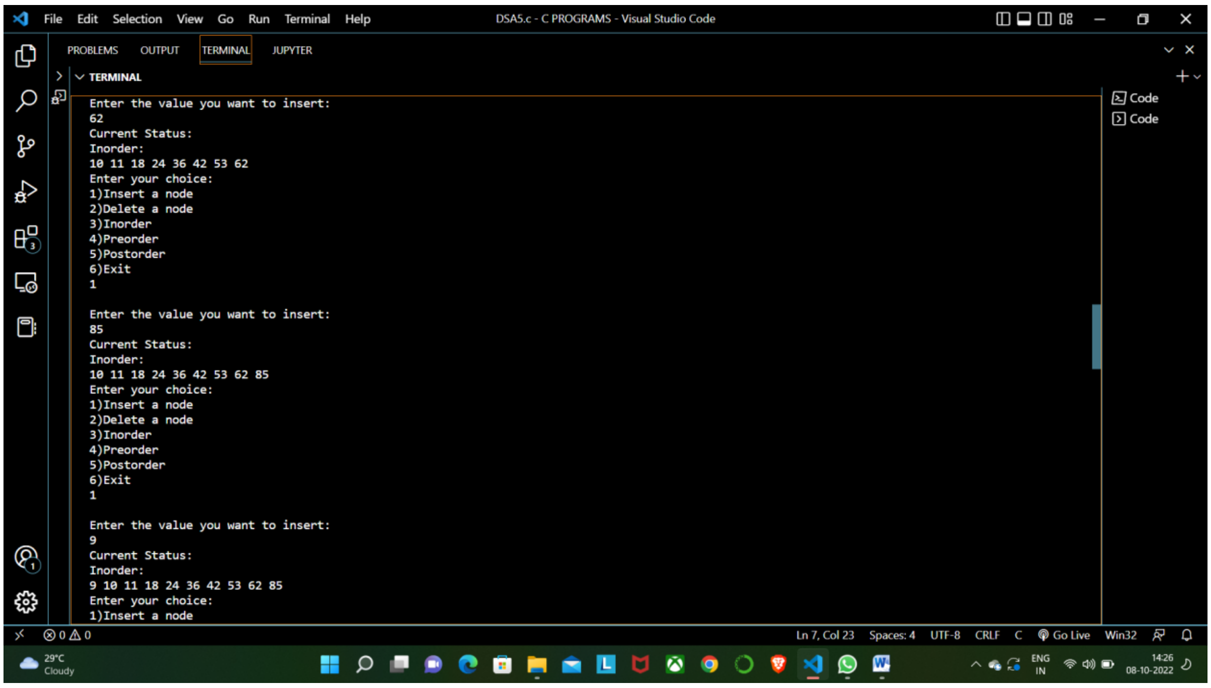
```
Enter the value you want to insert:
42
Current Status:
Inorder:
42 53
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
18
Current Status:
Inorder:
18 42 53
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
11
Current Status:
Inorder:
```


File Edit Selection View Go Run Terminal Help
DSA5.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Code
Ln 7, Col 23 Spaces: 4 UTF-8 CRLF C Go Live Win32
29°C Cloudy
14:25 08-10-2022

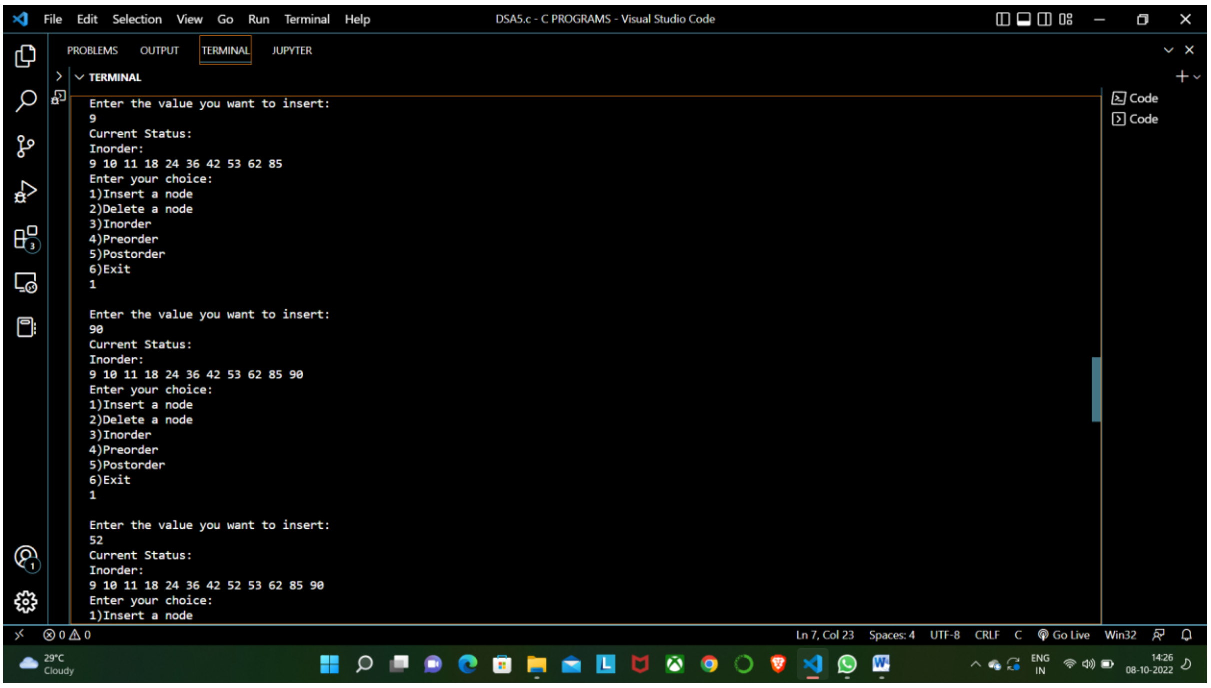
```
Enter the value you want to insert:
11
Current Status:
Inorder:
11 18 42 53
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
36
Current Status:
Inorder:
11 18 36 42 53
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
24
Current Status:
Inorder:
11 18 24 36 42 53
Enter your choice:
1)Insert a node
```


File Edit Selection View Go Run Terminal Help
DSA5.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Code
Ln 7, Col 23 Spaces: 4 UTF-8 CRLF C Go Live Win32
29°C Cloudy
14:25 08-10-2022

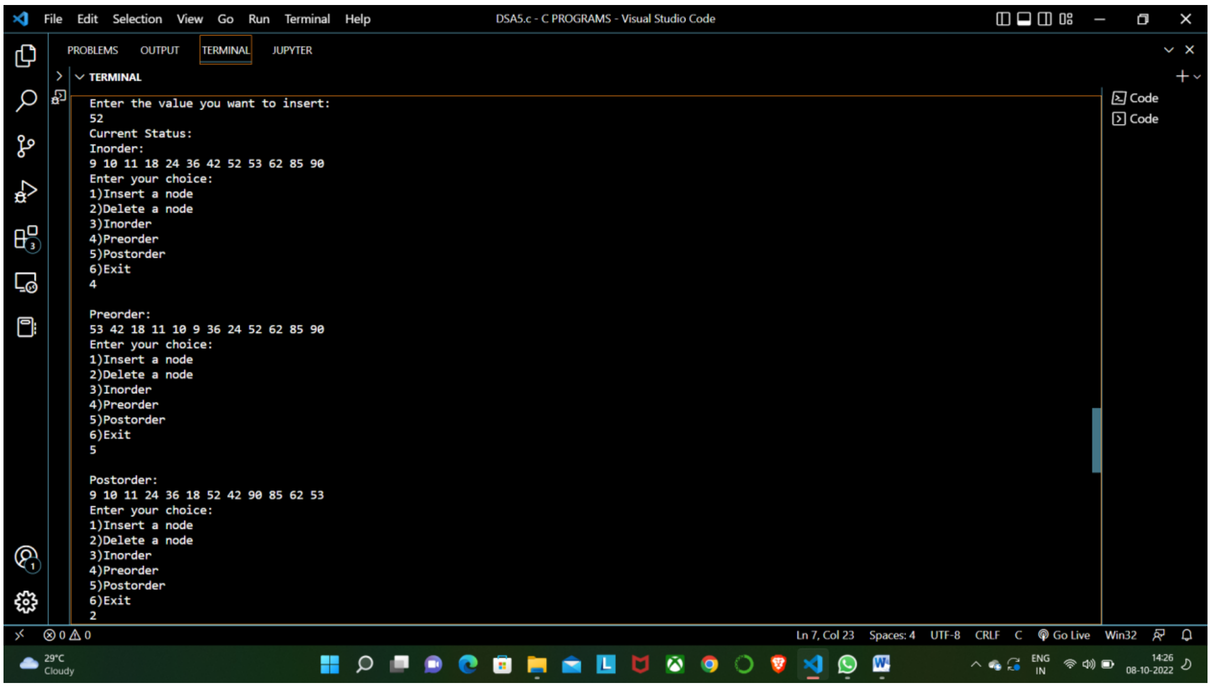
```
Enter the value you want to insert:
24
Current Status:
Inorder:
11 18 24 36 42 53
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
10
Current Status:
Inorder:
10 11 18 24 36 42 53
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
62
Current Status:
Inorder:
10 11 18 24 36 42 53 62
Enter your choice:
1)Insert a node
```


File Edit Selection View Go Run Terminal Help
DSA5.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Code
Ln 7, Col 23 Spaces: 4 UTF-8 CRLF C Go Live Win32
29°C Cloudy
ENG IN
14:26 08-10-2022

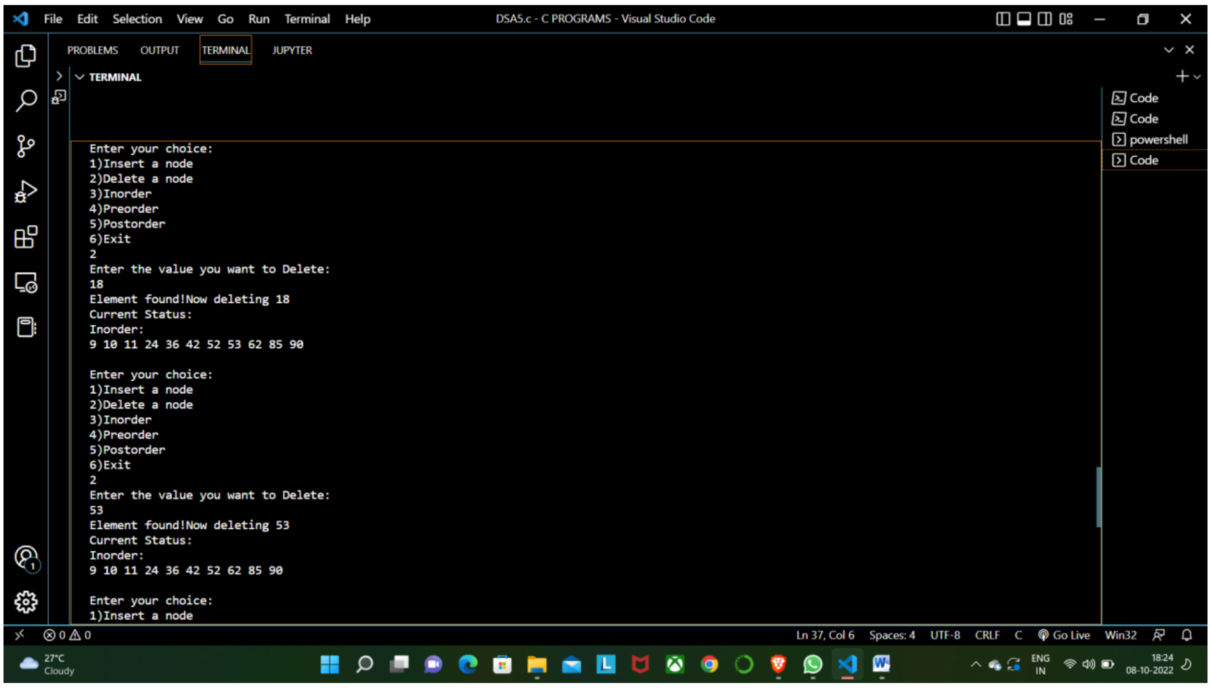
```
Enter the value you want to insert:
62
Current Status:
Inorder:
10 11 18 24 36 42 53 62
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
85
Current Status:
Inorder:
10 11 18 24 36 42 53 62 85
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
9
Current Status:
Inorder:
9 10 11 18 24 36 42 53 62 85
Enter your choice:
1)Insert a node
```


File Edit Selection View Go Run Terminal Help
DSA5.c - C PROGRAMS - Visual Studio Code
PROBLEMS OUTPUT TERMINAL JUPYTER
TERMINAL
Code
Code
Ln 7, Col 23 Spaces: 4 UTF-8 CRLF C Go Live Win32
29°C Cloudy
ENG IN
14:26 08-10-2022

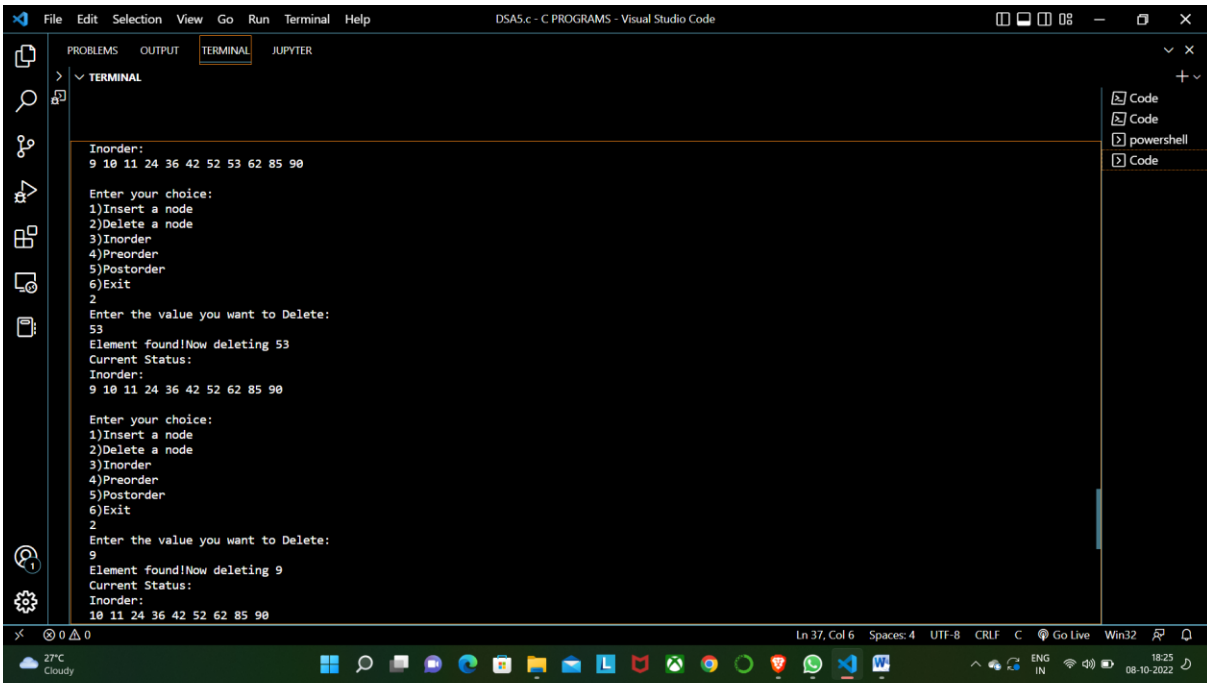
```
Enter the value you want to insert:
9
Current Status:
Inorder:
9 10 11 18 24 36 42 53 62 85
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
90
Current Status:
Inorder:
9 10 11 18 24 36 42 53 62 85 90
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
1

Enter the value you want to insert:
52
Current Status:
Inorder:
9 10 11 18 24 36 42 52 53 62 85 90
Enter your choice:
1)Insert a node
```



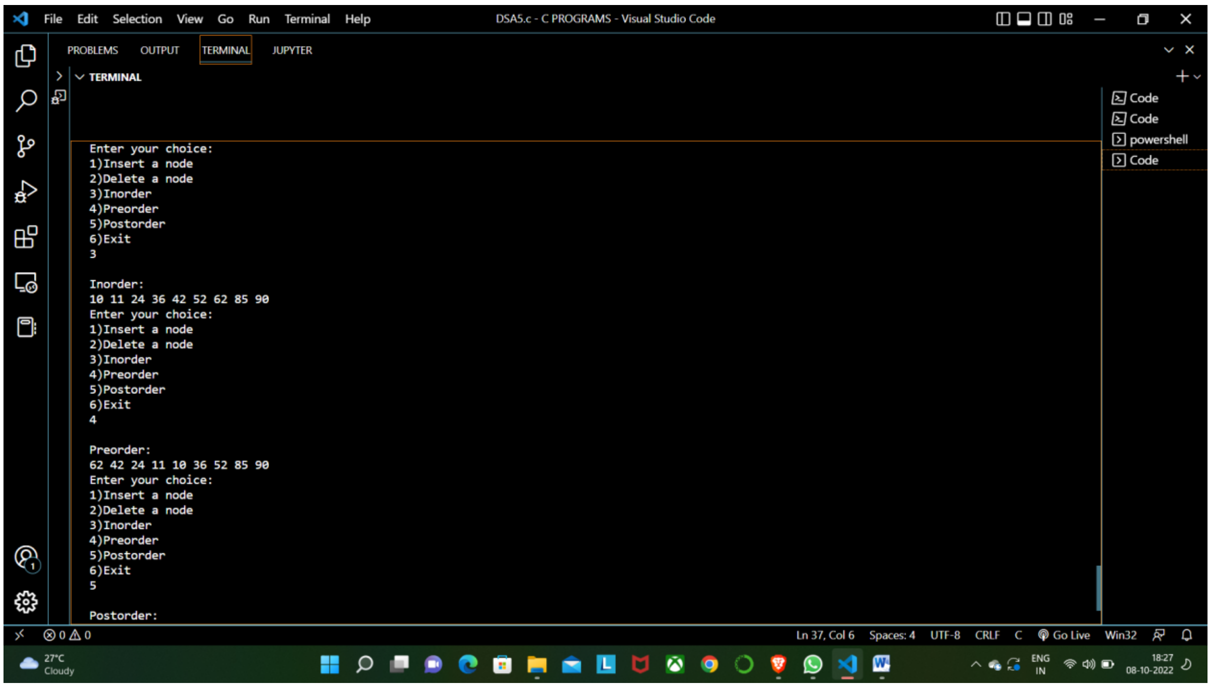
```
Enter the value you want to insert:
52
Current Status:
Inorder:
9 10 11 18 24 36 42 52 53 62 85 90
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
4

Preorder:
53 42 18 11 10 9 36 24 52 62 85 90
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
5

Postorder:
9 10 11 24 36 18 52 42 90 85 62 53
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
2
```



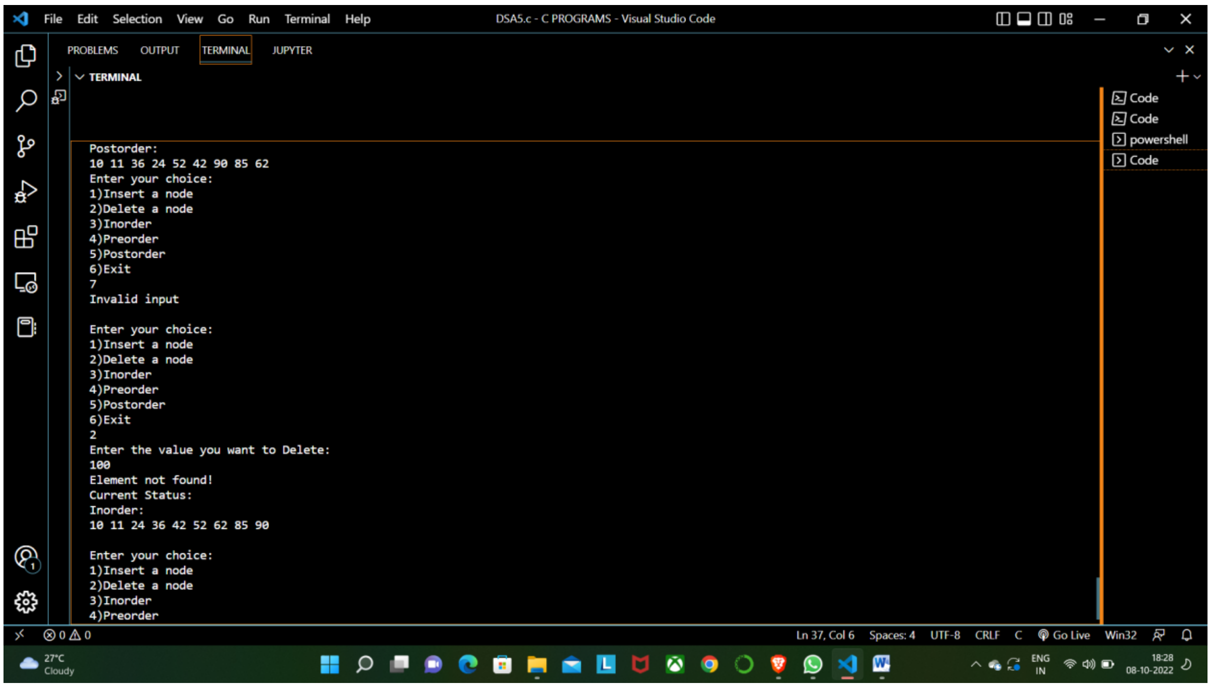
```
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
2
Enter the value you want to Delete:
18
Element found!Now deleting 18
Current Status:
Inorder:
9 10 11 24 36 42 52 53 62 85 90

Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
2
Enter the value you want to Delete:
53
Element found!Now deleting 53
Current Status:
Inorder:
9 10 11 24 36 42 52 62 85 90

Enter your choice:
1)Insert a node
```



```
Inorder:
9 10 11 24 36 42 52 53 62 85 90

Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
2
Enter the value you want to Delete:
53
Element found!Now deleting 53
Current Status:
Inorder:
9 10 11 24 36 42 52 62 85 90

Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
2
Enter the value you want to Delete:
9
Element found!Now deleting 9
Current Status:
Inorder:
10 11 24 36 42 52 62 85 90
```

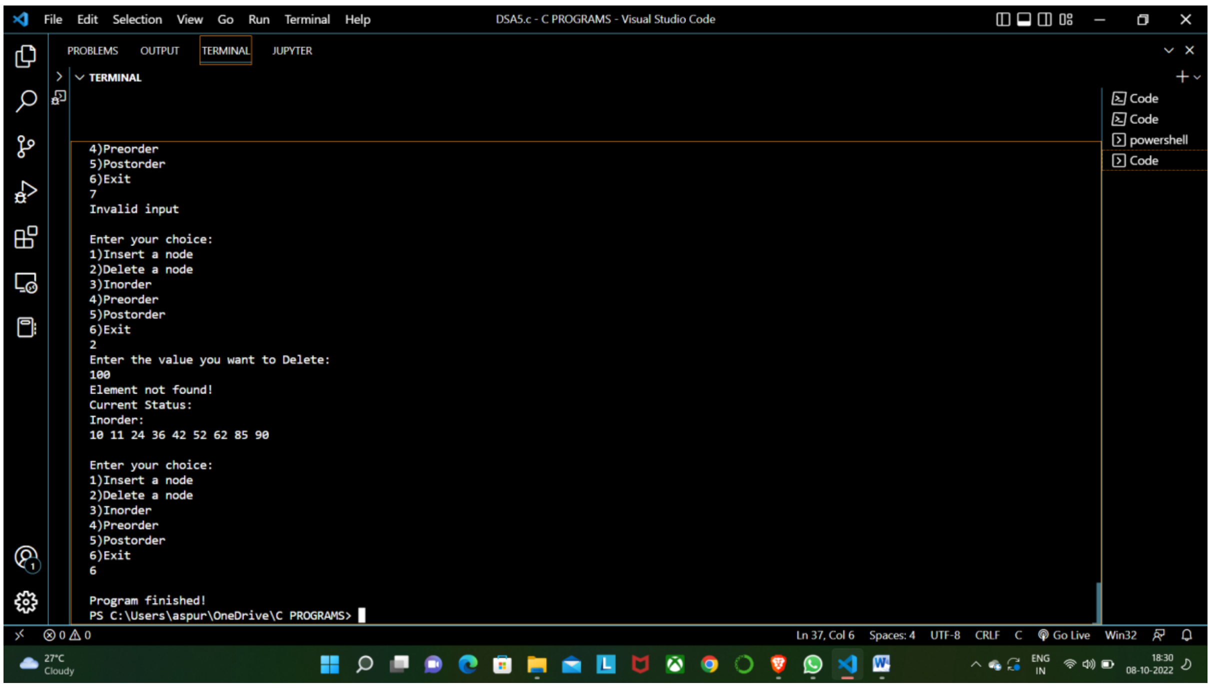


```
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
3

Inorder:
10 11 24 36 42 52 62 85 90
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
4

Preorder:
62 42 24 11 10 36 52 85 90
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
5

Postorder:
```



```
Postorder:
10 11 36 24 52 42 90 85 62
Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
7
Invalid input

Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
2
Enter the value you want to Delete:
100
Element not found!
Current Status:
Inorder:
10 11 24 36 42 52 62 85 90

Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
```

Test case : If element not present is tried to be deleted.



```
4)Preorder
5)Postorder
6)Exit
7
Invalid input

Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
2
Enter the value you want to Delete:
100
Element not found!
Current Status:
Inorder:
10 11 24 36 42 52 62 85 90

Enter your choice:
1)Insert a node
2)Delete a node
3)Inorder
4)Preorder
5)Postorder
6)Exit
6

Program finished!
PS C:\Users\aspur\OneDrive\C PROGRAMS> 
```

CONCLUSION:

In the above experiment we learnt about the concept of binary search trees. We understood the internal structure of a node, which contains a data part another node to the right and a node to the left. We understood that a binary tree has at most two children nodes. We understood various terminologies related to trees like root node, leaf node , left subtree and right subtree. We also understood that leaf node has no children. We implemented the Insertion , Deletion and

Inorder , Preorder , Postorder traversals also we implemented the functions Search , InorderSuccessor of the tree with the help of a menu driven program.