Data Structures

Deleting a node from Binary Search Tree

Deleting a node from a binary search tree:

- A Binary Search tree is a binary tree in which value of left sub-tree node is Less than or equal to its parent nodes value and the value of right sub-tree Node is greater than its parent nodes value.
- After deleting a node from a binary search tree, the obtained binary tree Must also be a binary search tree.
- When we want to delete a node from a binary search tree we have to consider Three cases.

    1)when the node to be deleted is the leaf node.

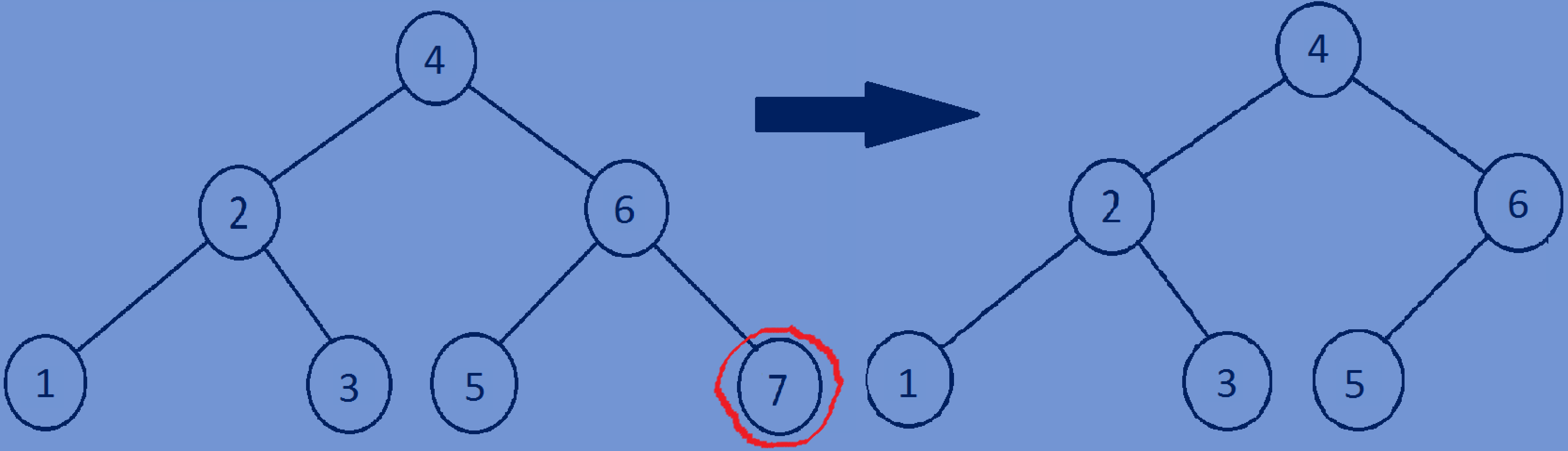
    2)when the node to be deleted has only one child(left or right).

    3)when the node to be deleted has both left and right child.
- Write a recursive function to delete a node of BST considering these three cases.

The node to be deleted is the leaf node:

- If the node to be deleted is the leaf node(leaf node doesn't have any children) Simple delete the node from the binary search tree.

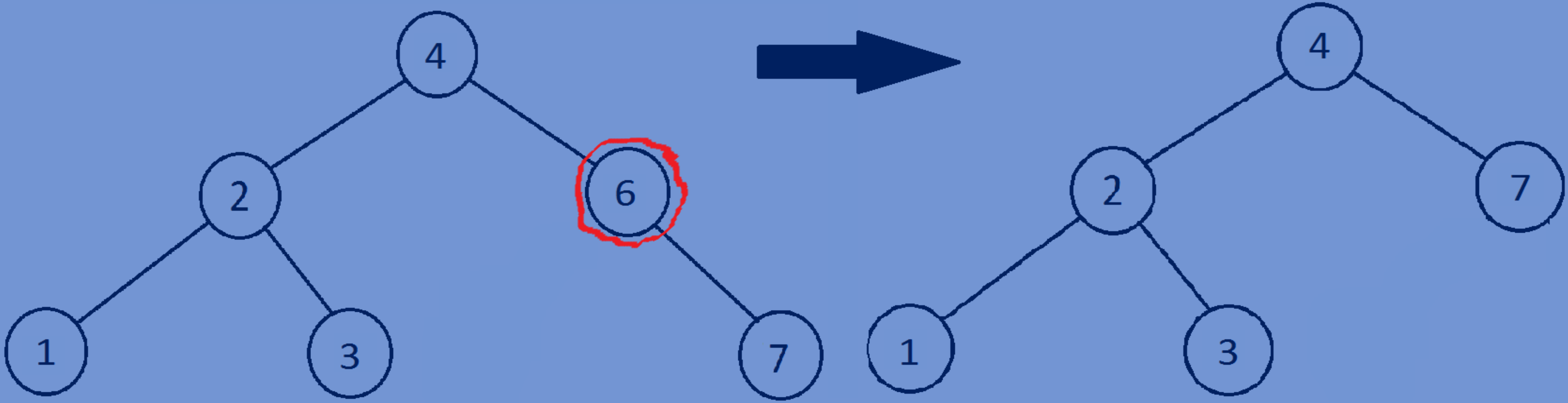
```
if (root->left == NULL && root->right == NULL)
    free(root);
    root = NULL;
    return root;
```



The node to be deleted has only one child(left or right):

- If the node to be deleted has only one child then store the address of left child (If right subtree is equal to NULL) in temp pointer or store the address of Right child(if left subtree of node is equal to NULL)in temp pointer and Return temp.

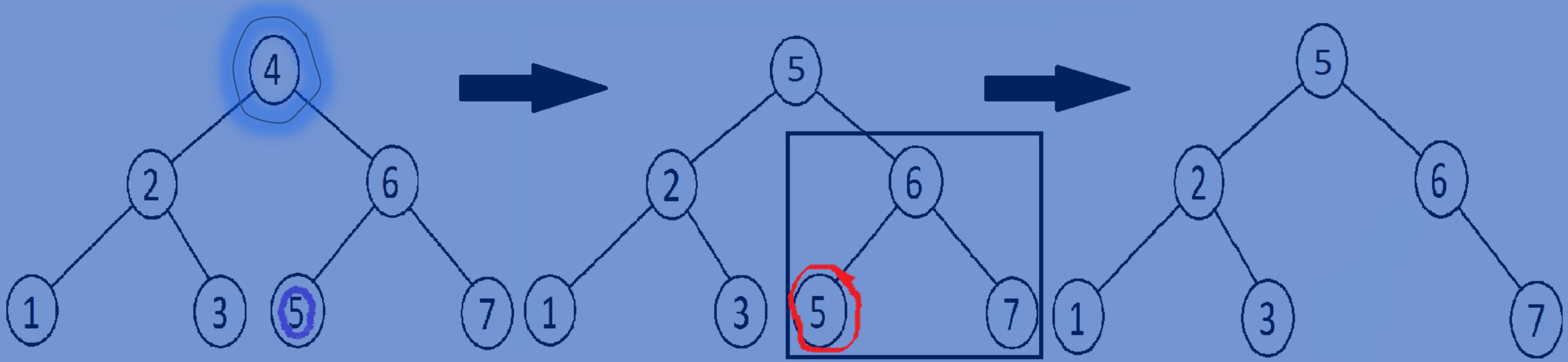
```
temp = root->right(or)temp=root->left;
free(root);
return temp;
```



The node to be deleted has both left and right child(two children):

- Find the minimum value node in the right subtree.
- replace the data of node to be deleted with the data of minimum value node. So now,the right subtree contains a duplicate node having same data of the node.
- Now call the delete function with the right subtree as an argument to delete the Duplicate node in the right subtree.

```
temp = FindMin(root->right);
root->data = temp->data;
root->right = deleteNodeInBstree(root->right, root->data);
```



Function for deleting a node from binary search tree:

```
//function for deleting a node from binary search tree.
Bstree *deleteNodeInBstree(Bstree *root,int data){
    if(root==NULL){
        return root;
    }
    if(root->data > data){
        root->left=deleteNodeInBstree(root->left,data);
    }
    else if(root->data < data){
        root->right=deleteNodeInBstree(root->right,data);
    }
    else {
        Bstree *temp=NULL;
        if (root->left == NULL && root->right == NULL)  {
            free(root);
            root = NULL;
            return root;
        }
        else if (root->left == NULL){
            temp=root->right;
            free(root);
            return temp;
        }
        else if(root->right==NULL){
            temp=root->left;
            free(root);
            return temp;
        }
        temp=FindMin(root->right);
        root->data=temp->data;
        root->right=deleteNodeInBstree(root->right,root->data);
    }
    return root;
}

//finding minimum value node of a binary search tree
Bstree *FindMin(Bstree *root){
    if(root->left==NULL){
        return root;
    }
    return FindMin(root->left);
}
```

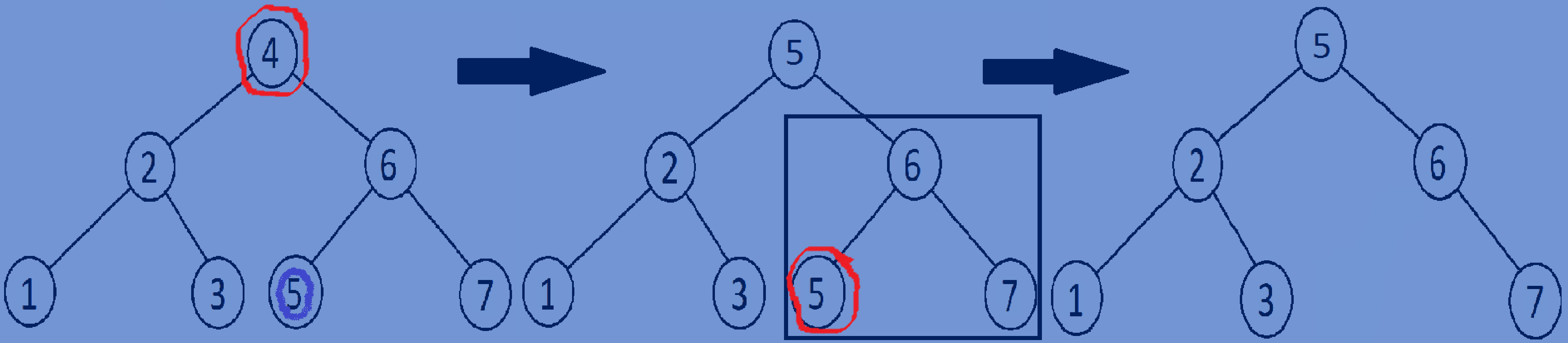
Whole program:

```c
#include<stdio.h>
#include<stdlib.h>
//creating a node
typedef  struct Bstree{
    int data;
    struct Bstree *left;
    struct Bstree *right;
}Bstree;
//creating new node
Bstree *createnewnode(int data){
    Bstree *newnode=(Bstree*)malloc(sizeof(Bstree));
    newnode->data=data;
    newnode->left=NULL;
    newnode->right=NULL;
    return newnode;
}
//function for inserting node into binary search tree
Bstree *InsertNodeintoBStree(Bstree *root,int data){
    if(root==NULL){
        return createnewnode(data);
    }
    if(data<=root->data){
        root->left=InsertNodeintoBStree(root->left,data);
    }
    else if(data>root->data){
        root->right=InsertNodeintoBStree(root->right,data);
    }
    return root;
}
//function for finding minimum value node of a binary search tree
Bstree *FindMin(Bstree *root){
    if(root->left==NULL){
        return root;
    }
    return FindMin(root->left);
}
//function for deleting a node from binary search tree
Bstree *deleteNodeInBstree(Bstree *root,int data){
    if(root==NULL){
        return root;
    }
    if(root->data > data){
        root->left=deleteNodeInBstree(root->left,data);
    }
    else if(root->data < data){
        root->right=deleteNodeInBstree(root->right,data);
    }
    else {
        Bstree *temp=NULL;
        if (root->left == NULL && root->right == NULL) {
            free(root);
            root = NULL;
            return root;
        }
        else if (root->left == NULL){
            temp=root->right;
            free(root);
            return temp;
        }
        else if(root->right==NULL){
            temp=root->left;
            free(root);
            return temp;
        }
        temp=FindMin(root->right);
        root->data=temp->data;
        root->right=deleteNodeInBstree(root->right,root->data);
    }
    return root;
}
//print all the nodes of a tree in preorder fashion
void preorder(Bstree *root){
    if(root){
        printf("%d ",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
//main function
int main(){
    Bstree *tree=NULL;
    tree=InsertNodeintoBStree(tree,4);
    tree=InsertNodeintoBStree(tree,2);
    tree=InsertNodeintoBStree(tree,6);
    tree=InsertNodeintoBStree(tree,1);
    tree=InsertNodeintoBStree(tree,3);
    tree=InsertNodeintoBStree(tree,5);
    tree=InsertNodeintoBStree(tree,7);
    preorder(tree);printf("\n");
    //calling the deleteNodeInBstree function
    tree=deleteNodeInBstree(tree,4);
    preorder(tree);
    return 0;
}
```

Output:

```
4 2 1 3 6 5 7

5 2 1 3 6 7
```



4
2
6
1
3
5
7
5
2
6
1
3
5
7
5
2
6
1
3
7