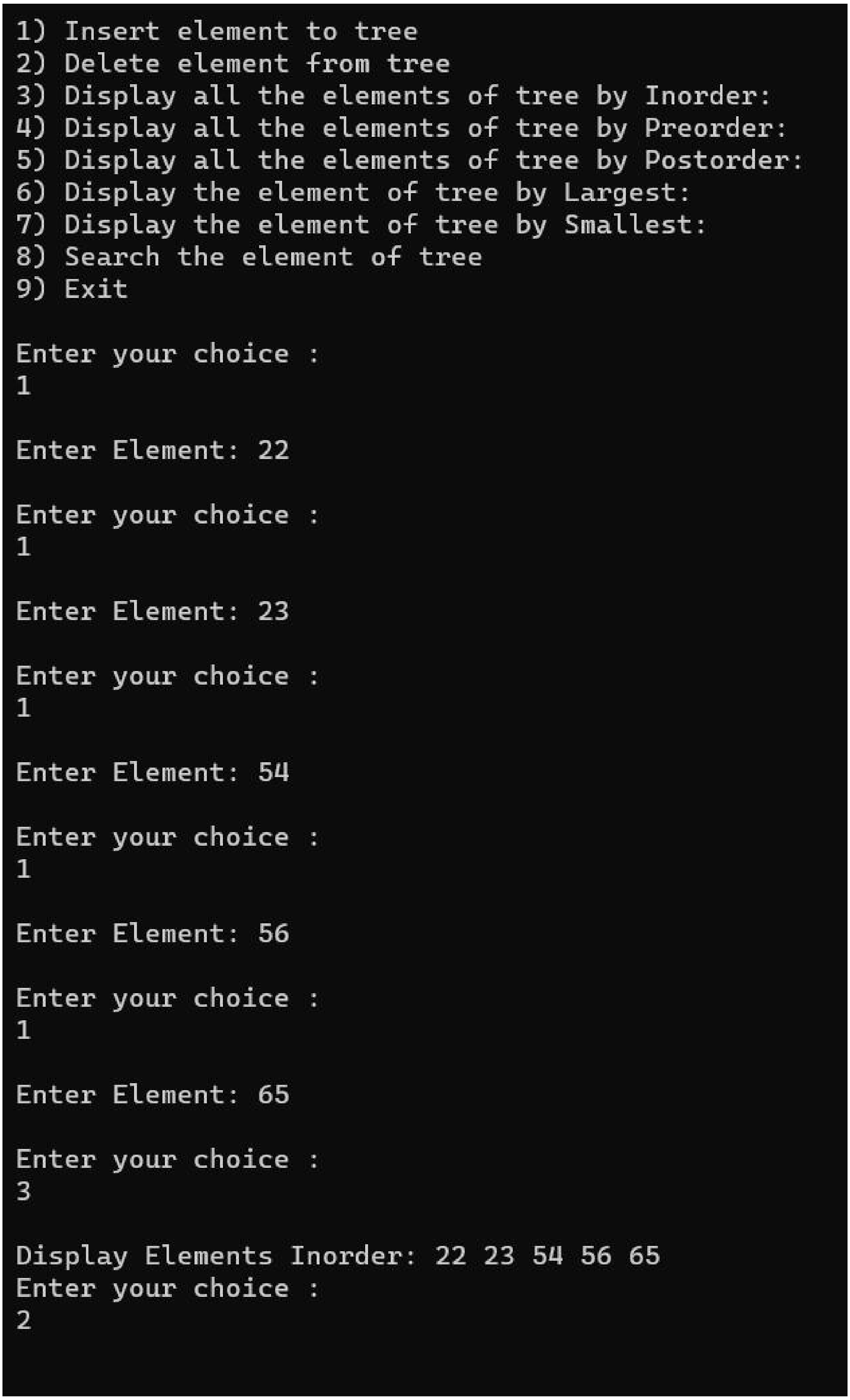
Binary Tree using Stack:

Code:

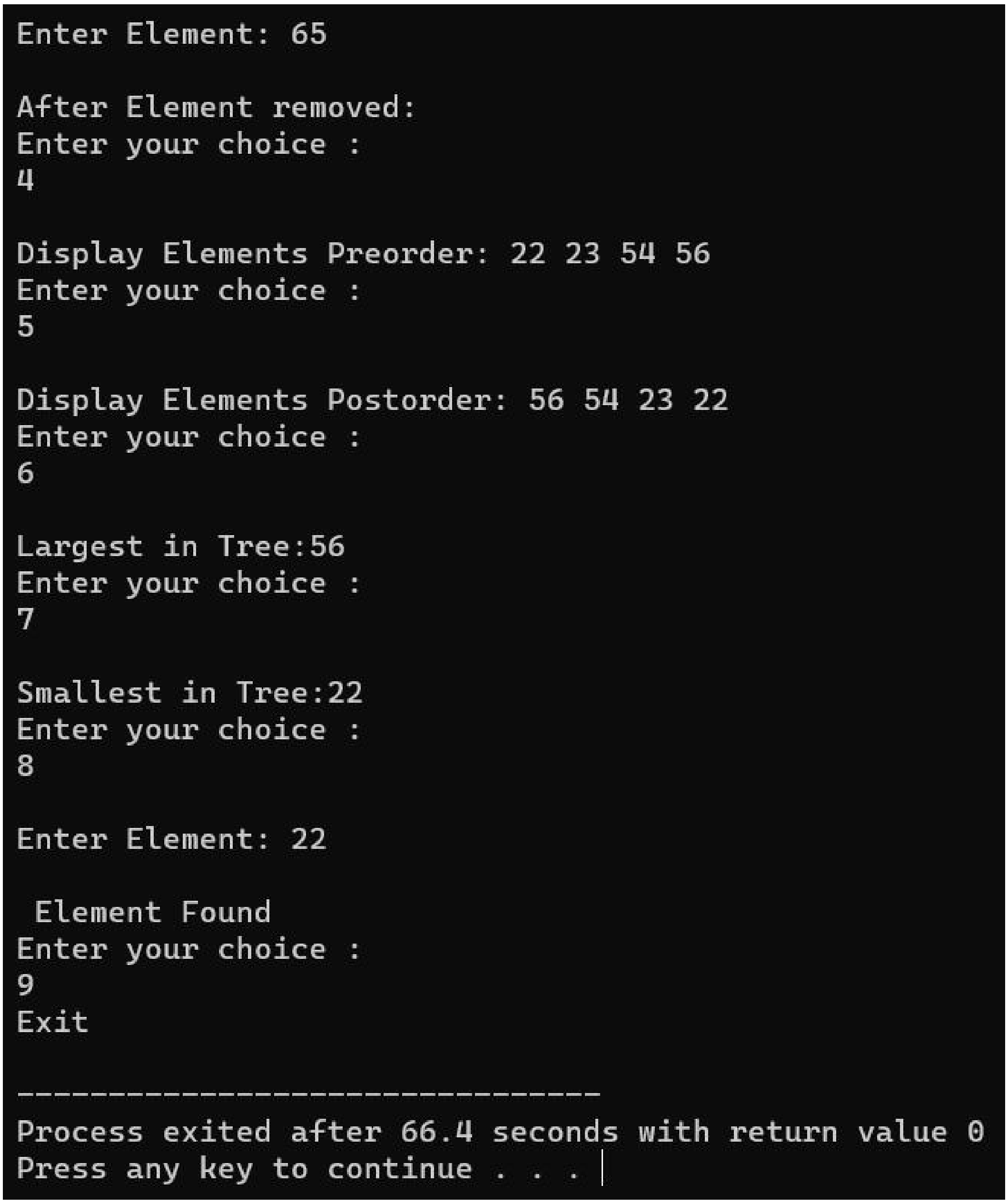
```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
class myBT;
class stack
{
    private:
        myBT*arr[25];
        int stack_top;
        int STACKSIZE;
        public:
            stack()
            {
                stack_top=-1;
                STACKSIZE=25;
            }
            void push(myBT*val)
            {
                stack_top=stack_top+1;
                arr[stack_top]=val;
            }
            myBT*pop()
            {
                myBT*val;
                val=arr[stack_top];
                --stack_top;
                return val;
            }
            bool is_empty()
            {
                if(stack_top==-1)
                return true;
                else
                return false;
            }
            bool is_full()
            {
                if(stack_top==STACKSIZE-1)
                return true;
                else
                return false;
            }
            int size(){
                return stack_top+1;
            }
            void display(){
                if(stack_top==-1){
                    cout<<"No element to display"<<endl;
                    return;
                }
                cout<<"Elements in the stack are : ";
                for(int i=0;i<=stack_top;i++)
                cout<<arr[i]<<" ";
                cout<<endl;
            }
};
class myBT{
    public:
        int data;
        myBT*left;
        myBT*right;
        static int node_count;
        myBT(int dataValue);
        myBT();
        void insertNode(int dataValue);
        void removeNode(int dataValue);
        void inOrder(myBT*r);
        void preOrder(myBT*r);
        void postOrder(myBT*r);
        void search(int targetValue);
        int smallest(myBT*r);
        int largest(myBT*r);
};
myBT*root;
myBT::myBT(){
    data=0;
    left=NULL;
    right=NULL;
}
myBT::myBT(int val){
    data=val;
    left=NULL;
    right=NULL;
    node_count++;
}
void myBT::insertNode(int dataValue){
    myBT*temp=new myBT(dataValue);
    myBT*trav=root;
    myBT*hold=NULL;
    if(trav!=NULL){
        while(trav!=NULL){
            hold=trav;
            if(dataValue>=trav->data){
                trav=trav->right;
            }else{
                trav=trav->left;
            }
        }
        if(hold->data>dataValue){
            hold->left=temp;
        }else{
            hold->right=temp;
        }
    }else{
        root=temp;
    }
}
void myBT::inOrder(myBT*r){
    myBT*trav=r;
    stack myStack;
    while(trav!=NULL){
        myStack.push(trav);
        trav=trav->left;
    }
    trav=myStack.pop();
    while(trav!=NULL){
        cout<<trav->data<<" ";
        if(trav->right!=NULL){
            trav=trav->right;
            while(trav!=NULL){
                myStack.push(trav);
                trav=trav->left;
            }
        }if(myStack.is_empty()==false)
        trav=myStack.pop();
        else
        trav=NULL;
    }
}
void myBT::preOrder(myBT*r){
    myBT*trav=r;
    stack myStack;
    while(trav!=NULL){
    cout<<trav->data<<" ";
    if(trav->right!=NULL){
        myStack.push(trav->right);
    }if(trav->left!=NULL){
        trav=trav->left;
    }else{
        if(myStack.is_empty()==false)
        trav=myStack.pop();
        else
        trav=NULL;
    }
    }
}
void myBT::postOrder(myBT*r){
    myBT*previous=r;
    myBT*s=NULL;
    stack myStack;
    myStack.push(r);
    while(myStack.is_empty()==false){
        s=myStack.pop();
        if(s->right==NULL && s->left==NULL){
            previous=s;
            cout<<s->data<<" ";
        }else{
            if(s->right==previous||s->left==previous){
                previous=s;
                cout<<s->data<<" ";
            }else{
                myStack.push(s);
                if(s->right!=NULL){
                    myStack.push(s->right);
                }if(s->left!=NULL){
                    myStack.push(s->left);
                }
            }
        }
    }
}
int myBT::smallest(myBT*r){
    myBT*trav=r;
    while(trav->left!=NULL){
        trav=trav->left;
    }
    return trav->data;
}
int myBT::largest(myBT*r){
    myBT*trav=r;
    while(trav->right!=NULL){
        trav=trav->right;
    }
    return trav->data;
}
void myBT::search(int targetValue){
    myBT*trav=root;
    bool findFlag=false;
    while(trav!=NULL){
        if(targetValue<trav->data){
            trav=trav->left;
        }else if(targetValue>trav->data){
            trav=trav->right;
        }else{
            findFlag=true;
            break;
        }
    }if(findFlag==true)
    cout<<"\n Element Found";
    else
    cout<<"\n Element Not Found ";
}
void myBT::removeNode(int dataValue){
    myBT*trav=root;
    myBT*hold=root;
    myBT*temp=NULL;
    bool findFlag=false;
    bool isLeft=false;
    while(trav!=NULL){
        if(dataValue<trav->data){
            hold=trav;
            trav=trav->left;
            isLeft=true;
        }else if(dataValue>trav->data){
            hold=trav;
            trav=trav->right;
            isLeft=false;
        }else{
            findFlag=true;
            break;
        }
    }
    if(findFlag==true){
        if(trav->left==NULL && trav->right==NULL){
```

```
                        free(trav);
                        if(isLeft==true)
                        hold->left=NULL;
                        else hold->right=NULL;
                }
                else if (trav->left==NULL && trav->right!=NULL){
                        if(isLeft==true)
                        hold->left=trav->right;
                        else
                        hold->right=trav->right;
                        free(trav);
                }else if(trav->left!=NULL && trav->right==NULL){
                        if(isLeft==true)
                        hold->left=trav->left;
                        else hold->right=trav->left;;
                        free(trav);
                }
                else
                {
                        int largest=trav->left->largest(trav->left);
                        root->removeNode(largest);
                        trav->data=largest;
                }
        }
        else
        cout<<"\nElement Not Found";
}
int myBT::node_count=0;
int main(){
        system("cls");
        int ch, p;
   cout << "1) Insert element to tree " << endl;
   cout << "2) Delete element from tree " << endl;
   cout << "3) Display all the elements of tree by Inorder:" << endl;
   cout << "4) Display all the elements of tree by Preorder:" << endl;
   cout << "5) Display all the elements of tree by Postorder:" << endl;
   cout << "6) Display the element of tree by Largest:" << endl;
   cout << "7) Display the element of tree by Smallest:" << endl;
   cout << "8) Search the element of tree " << endl;
   cout << "9) Exit" << endl;

   do {
      cout << "\nEnter your choice : " << endl;
      cin >> ch;
      switch (ch) {
      case 1:
         cout << "\nEnter Element: ";
         cin >> p;
         root->insertNode(p);
         break;
      case 2:
         cout << "\nEnter Element: ";
         cin >> p;
         root->removeNode(p);
         cout << "\nAfter Element removed: ";
         break;
      case 3:
            cout << "\nDisplay Elements Inorder: ";
            root->inOrder(root);
            break;
        case 4:
            cout << "\nDisplay Elements Preorder: ";
            root->preOrder(root);
            break;
        case 5:
            cout << "\nDisplay Elements Postorder: ";
            root->postOrder(root);
            break;
        case 6:
            cout << "\nLargest in Tree:" << root->largest(root);
            break;
        case 7:
            cout << "\nSmallest in Tree:" << root->smallest(root);
            break;
        case 8:
            cout << "\nEnter Element: ";
            cin >> p;
            root->search(p);
            break;
        case 9:
            cout << "Exit" << endl;
            exit(0);
        default:
            cout << "Invalid choice" << endl;
        }
    } while (ch != 9);
            return 0;
}
```

Output:



```
1) Insert element to tree
2) Delete element from tree
3) Display all the elements of tree by Inorder:
4) Display all the elements of tree by Preorder:
5) Display all the elements of tree by Postorder:
6) Display the element of tree by Largest:
7) Display the element of tree by Smallest:
8) Search the element of tree
9) Exit

Enter your choice :
1

Enter Element: 22

Enter your choice :
1

Enter Element: 23

Enter your choice :
1

Enter Element: 54

Enter your choice :
1

Enter Element: 56

Enter your choice :
1

Enter Element: 65

Enter your choice :
3

Display Elements Inorder: 22 23 54 56 65
Enter your choice :
2
```



```
Enter Element: 65

After Element removed:
Enter your choice :
4

Display Elements Preorder: 22 23 54 56
Enter your choice :
5

Display Elements Postorder: 56 54 23 22
Enter your choice :
6

Largest in Tree:56
Enter your choice :
7

Smallest in Tree:22
Enter your choice :
8

Enter Element: 22

 Element Found
Enter your choice :
9
Exit

--------------------------------
Process exited after 66.4 seconds with return value 0
Press any key to continue . . .
```