```
#include <iostream>
#include <iomanip>
using namespace std;



class Binary
{
public:
	struct Node
	{
		Node *left ;
		Node *right;
		int value;

		Node(int value):left(nullptr),right(nullptr),value(value)
		{}
	};
	int count;

	Node *head;
	Binary():head(nullptr),count(0)
	{}

	//to add NODE
	Node* addNode(Node *node,int value)
	{
		if (nullptr == node)
			return new Node(value);
		if (value > node->value)
			node->right=addNode(node->right,value);
		else
			node->left=addNode(node->left,value);
		return node;
	}

	//to Print InOrder
	void printInOrder(Node *node)
	{
		if (node)
		{
			printInOrder(node->left);
			cout<<node->value<<"\t";
			printInOrder(node->right);

		}
	}

	//To PreOrder
	void printPreOrder(Node*node)
	{
		if (node)
		{
			cout<<node->value<<"\t";
			printPreOrder(node->left);
			printPreOrder(node->right);

		}
	}

	//To PostOrder
	void printPostOrder(Node*node)
	{
		if (node)
		{
```

```
                printPostOrder(node->left);
                printPostOrder(node->right);
                cout<<node->value<<"\t";

        }
}

//To Remove Node
Node *remove (Node*node ,int value)
{
        if (value>node->value)
                node->right=remove(node->right,value);
        else if (value<node->value)
                node->left=remove(node->left,value);
        else
        {
                //if right and left are null
                if (node->left==nullptr && node->right==nullptr)
                {
                        delete node;
                        return nullptr;
                }
                //if right is not null
                if (node->left==nullptr && node->right!=nullptr)
                {
                        Node*o=node->right;
                        delete node;
                        return o;
                }
                //if left is not null
                if (node->right==nullptr && node->left!=nullptr)
                {
                        Node*o=node->left;
                        delete node;
                        return o;
                }
                //if both are present
                Node *success = node->right;

                //To get successor pointer
                while (success->left)
                {
                        success=success->left;
                }

                //replace node value with successor value
                node->value=success->value;

                //to delete successors child
                //we have to call remove func with HEAD->right, successor->value
                //AND assign to node->right
                node->right=remove(node->right,success->value);
        }
        return node;

}

void add(int value)
{
        head=addNode(head,value);
}

void removeNode(int value)
{
        head=remove(head,value);
```

```
	}
	void print()
	{
		printInOrder(head);
	}
	void printPre()
	{
		printPreOrder(head);
	}
	void printPost()
	{
		printPostOrder(head);
	}
	void debug(Node*node)
	{
	static int level = 0;
	if(node)
	{

	level++;
	debug(node->right);
	cout<< setw(level*4)<<" "<<node->value<<endl;
	debug(node->left);
	level--;
	}
	}
	void Debug_kk()
	{
	debug(head);
	}

};

int main()
{
	Binary b1;
	int num;
	while (cout<<"Enter value to store in tree (press 0to STOP)  ",
		cin>>num,
		num)
	{
		b1.add(num);
		b1.Debug_kk();

	}


	while (cout<<"Enter value to remove from tree (press 0to STOP)  ",
		cin>>num,
		num)
	{
		b1.removeNode(num);
		cout<<"\nin Order print"<<endl;
		b1.Debug_kk();

	}
	cout<<"\nin Order print"<<endl;
	b1.print();
	cout<<"\nPreOrder print"<<endl;
	b1.printPre();
	cout<<"\nPostOrder print"<<endl;
	b1.printPost();
}
```

```
                9
            8
        7
            6
    5
            4
        3
            2
                1
Enter value to store in tree (press 0to STOP)  5
                9
            8
        7
            6
    5
                5
            4
        3
            2
                1
Enter value to store in tree (press 0to STOP)  0
Enter value to remove from tree (press 0to STOP)  5

in Order print
                9
            8
        7
    6
                5
            4
        3
            2
                1
Enter value to remove from tree (press 0to STOP)  3

in Order print
                9
            8
        7
    6
            5
        4
            2
                1
Enter value to remove from tree (press 0to STOP)  5

in Order print
                9
            8
        7
    6
        4
            2
                1
Enter value to remove from tree (press 0to STOP)
```