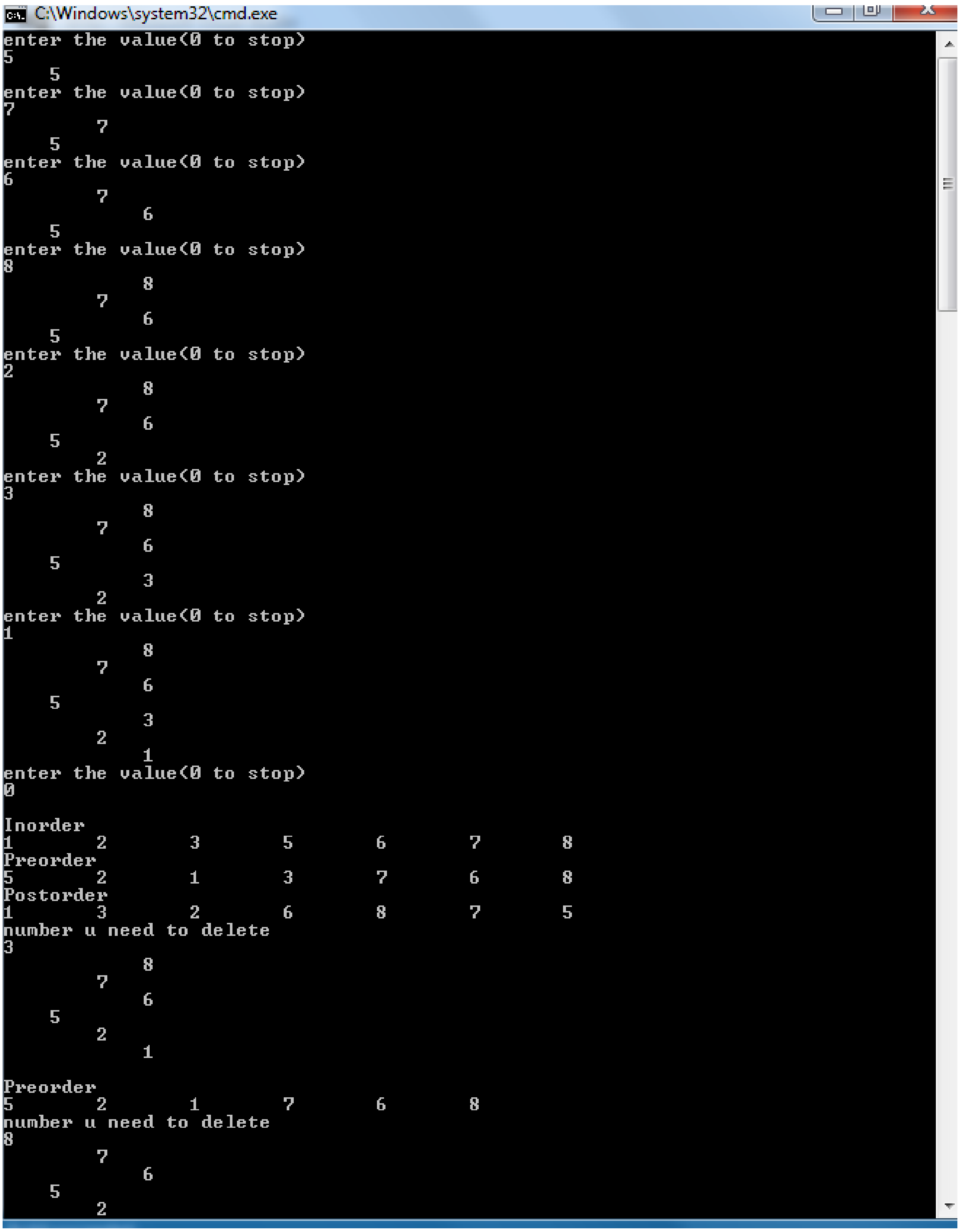
```cpp
#include <iostream>
#include <iomanip>//for setw()
using namespace std;
class BinaryTree
{
public:
	struct Node
	{
		int value;
		Node *right;
		Node *left;
		Node(int value):value(value),right(nullptr),left(nullptr)
		{}
	};

	Node *node;
	BinaryTree():node(nullptr)
	{}
	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;
	}
	void Inorder(Node *node)
	{
		if(node)
		{
			Inorder(node->left);
			cout<<node->value<<'\t';
			Inorder(node->right);
		}
	}

	void Preorder(Node *node)
	{
		if(node)
		{
			cout<<node->value<<'\t';
			Preorder(node->left);
			Preorder(node->right);
		}
	}

	void Postorder(Node *node)
	{
		if(node)
		{
			Postorder(node->left);

			Postorder(node->right);
			cout<<node->value<<'\t';
		}
	}

	Node* Remove(Node *node,int value)
	{
		//for parent node but not child nodes
		if(value > node->value)
			node->right=Remove(node->right, value);
		else if(value < node->value)
```

```
				node->left=Remove(node->left,value);
			else
			{
				if(nullptr==node->right && nullptr==node->left)
				{
					delete node;
					return nullptr;
				}

				//if node has left child but not right child
				if(nullptr!= node->left && nullptr==node->right)
				{
					Node *orphan;
					orphan = node->left;
					delete node;
					return orphan;
				}

				//if node has right child but not  left  child
				if(nullptr!= node->right && nullptr==node->left)
				{
					Node *orphan;
					orphan = node->right;
					delete node;
					return orphan;
				}

				Node *successor=node->right;

				while(successor->left!=nullptr)
					successor=successor->left;

				node->value=successor->value;

				node->right=Remove(node->right, successor->value);

			}
			return node;
		}
		void printDebug(Node*node)
		{
			static int level=0;
			if(node)
			{
				level++;
				printDebug(node->right);
				cout<<setw(level*4)<<"  "<<node->value<<endl;
				printDebug(node->left);
				level--;
			}
		}
};
int main()
{
		BinaryTree B;
		B.node=nullptr;
		int n,d;
		while(cout<<"enter the value(0 to stop)"<<endl,
			cin>>n,
			n!=0)
		{
			B.node=B.addNode(B.node, n);
			B.printDebug(B.node);
		}
		cout<<endl<<"Inorder"<<endl;
```

```
	B.Inorder(B.node);
	cout<<endl<<"Preorder"<<endl;
	B.Preorder(B.node);
	cout<<endl<<"Postorder"<<endl;
	B.Postorder(B.node);
	while(cout<<endl<<"number u need to delete"<<endl,
		cin>>n,
		n)
	{
		B.node=B.Remove(B.node,n);
		B.printDebug(B.node);

		cout<<endl<<"Preorder"<<endl;
		B.Preorder(B.node);

	}
	return 0;
}
```



```
enter the value(0 to stop)
5
    5
enter the value(0 to stop)
7
        7
    5
enter the value(0 to stop)
6
        7
            6
    5
enter the value(0 to stop)
8
            8
        7
            6
    5
enter the value(0 to stop)
2
            8
        7
            6
    5
        2
enter the value(0 to stop)
3
            8
        7
            6
    5
            3
        2
enter the value(0 to stop)
1
            8
        7
            6
    5
            3
        2
            1
enter the value(0 to stop)
0

Inorder
1       2       3       5       6       7       8
Preorder
5       2       1       3       7       6       8
Postorder
1       3       2       6       8       7       5
number u need to delete
3
            8
        7
            6
    5
        2
            1

Preorder
5       2       1       7       6       8
number u need to delete
8
        7
            6
    5
        2
```