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

class Avl
{
private:
	struct Node
	{
		int value;
		Node *left;
		Node *right;
		Node(int value):value(value), left(nullptr), right(nullptr)
		{}
	};

	Node *root;//pointer for passing node adddress

	//Adding the nodes into the tree
	Node *add(Node *node, int value)
	{
		if(nullptr==node)
			return new Node(value);
		if(value > node->value)
			node->right =add(node->right, value);
		else
			node->left=add(node->left, value);
		//calling Rebalance function
		node=Rebalance(node);
		return node;
	}

	void Inorder(Node*node)
	{
		if(node)
		{
			Inorder(node->left);
			cout<<node->value<<"\t";
			Inorder(node->right);
		}
	}

	//Removing the nodes from the tree
	Node *Remove(Node *node, int value)
	{
		if(node==nullptr)
			return 0;

		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->left && nullptr == node->right)
			{
				delete node;
				return nullptr;
			}
			if(nullptr != node->left && nullptr == node->right)
			{
				Node *orphan = node->left;
				delete node;
				return orphan;
```

```
                }
                if(nullptr == node->left && nullptr != node->right)
                {
                        Node *orphan = node->right;
                        delete node;
                        return orphan;
                }

                Node *successor = node->right;
                while(nullptr != successor->left)
                        successor = successor->left;

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


        }
        //calling Rebalance function
        node=Rebalance(node);
        return node;
}

//For Displaying the output
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--;
        }
}

//Solution for Right-Right Heavy
Node *RotateLeft(Node *node)
{
        Node *child = node->right;
        node->right=child->left;
        child->left=node;
        return child;
}

//Solution for Left-Left Heavy
Node *RotateRight(Node *node)
{
        Node *child = node->left;
        node->left = child->right;
        child->right = node;
        return child;
}

//Solution for Right-Left Heavy
Node *RotateRightLeft(Node *node)
{
        node->right = RotateRight(node->right);
        node = RotateLeft(node);
        return node;

}

//Solution for Left-Right Heavy
Node *RotateLeftRight(Node *node)
```

```
	{
		node->left = RotateLeft(node->left);
		node = RotateRight(node);
		return node;

	}

	//Finding the height values of node
	int height(Node *node)
	{
		if(!node)
			return 0;
		int leftheight = height(node->left);
		int rightheight = height(node->right);
		int h = max( leftheight, rightheight);
		return h+1;
	}

	//Finding the balance of the node usinf height
	int balance(Node *node)
	{
		if(node)
			return height(node->right)- height(node->left);
		else
			return 0;
	}

	//Rebalancing the node if it got imbalance by using balance function and height
function
	Node *Rebalance(Node *node)
	{
		int nodeBalance = balance(node);
		if (nodeBalance == +2)//check condition if its right
		{
			int childBalance = balance(node->right);
			if (childBalance == +1)//checking condition for Right-Right or Right-
Left
				node = RotateLeft(node);
			else
				node = RotateRightLeft(node);
		}
		else if (nodeBalance == -2)//check condition if its left
		{
			int childBalance = balance(node->left);
			if(childBalance == -1)//checks condition for Left-Left or LeftRight
				node = RotateRight(node);
			else
				node = RotateLeftRight(node);
		}
		return node;
	}

public:


	//Abstraction method calling class functions
	//using another function
	Avl():root(nullptr)
	{}
	void Addnode(int value)
	{
		root = add(root, value);
	}
	void dele(int value)
	{
```

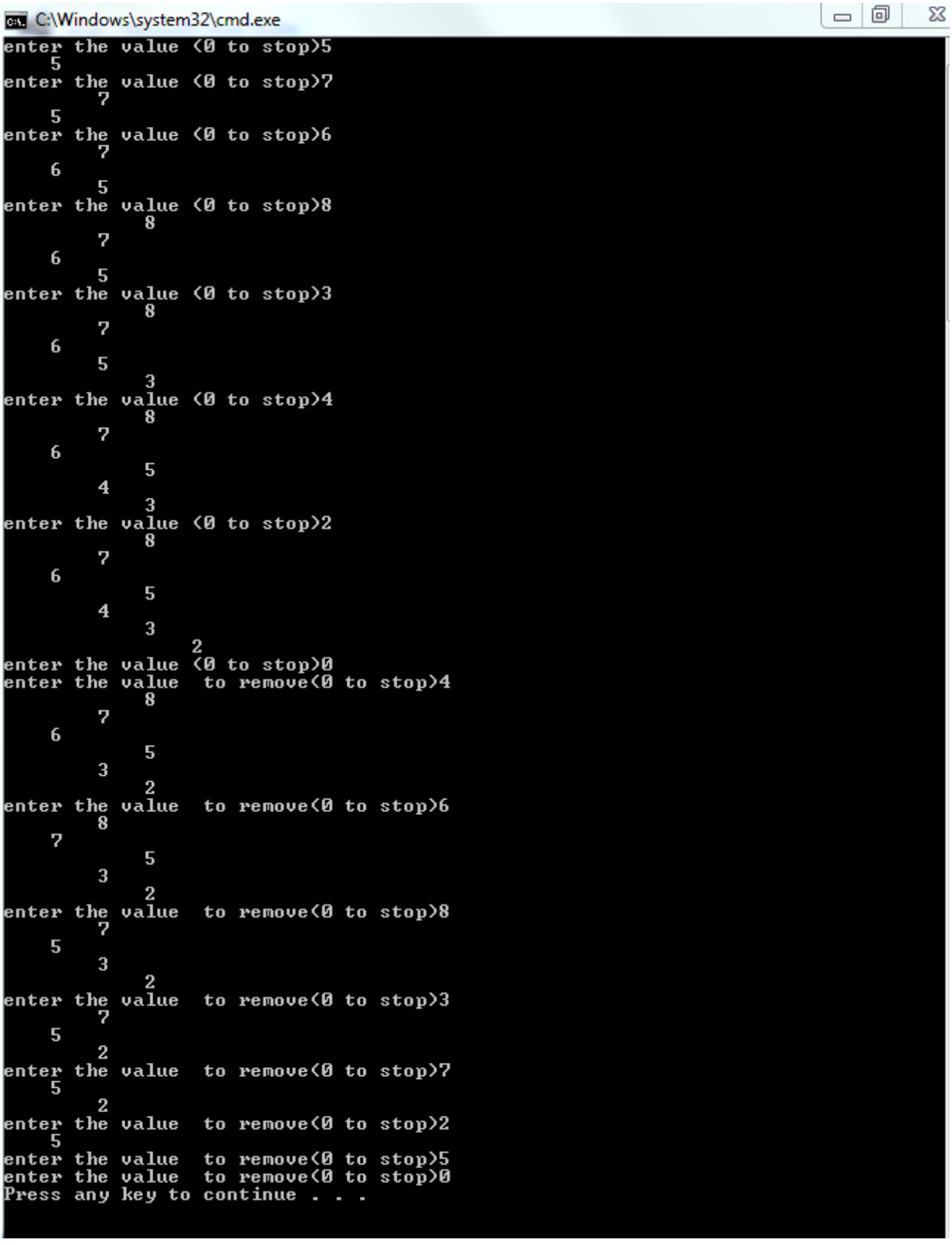
```
                root=Remove(root, value);
        }
        void Print()
        {
                printDebug(root);
        }
};


int main()
{
        Avl B;
        int value,n;

        //inserting the nodes
        while(cout<<"enter the value (0 to stop)",
                cin>>value,
                value!=0)
        {
                B.Addnode(value);
                B.Print();
        }

        //entering nodes to remove
        while(cout<<"enter the value  to remove(0 to stop)",
                cin>>n,
                n!=0)
        {
                B.dele(n);
                B.Print();
        }
        return 0;
}
```

OUTPUT:


C:\Windows\system32\cmd.exe
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)3
8
7
6
5
3
enter the value (0 to stop)4
8
7
6
5
4
3
enter the value (0 to stop)2
8
7
6
5
4
3
2
enter the value (0 to stop)0
enter the value to remove(0 to stop)4
8
7
6
5
3
2
enter the value to remove(0 to stop)6
8
7
5
3
2
enter the value to remove(0 to stop)8
7
5
3
2
enter the value to remove(0 to stop)3
7
5
2
enter the value to remove(0 to stop)7
5
2
enter the value to remove(0 to stop)2
5
enter the value to remove(0 to stop)5
enter the value to remove(0 to stop)0
Press any key to continue . . .