```
//============================================================================
// Name        : 21118_DSA_A03.cpp
// Author      : Shubham (Roll No: 21118)
//============================================================================

#include <iostream>
using namespace std;

class Node {
private:
	int data;
	Node *lChild, *rChild;
	bool lTh, rTh;
public:
	Node(int x = 0) {
		data = x;
		lChild = rChild = NULL;
		lTh = rTh = true;
	}
	friend class TBST;
};

class TBST {
private:
	Node *root, *head;
public:
	TBST() {
		root = NULL;
		head = new Node(0);
	}
	Node* getRoot() {return root;}
	void InsertNode(int x) {
		if (root == NULL) {
			root = new Node(x);
			root->lChild = root->rChild = head;
			return;
		}
		Node *curr = root, *prev = NULL;
		while (curr != head) {
			prev = curr;
			if (x < curr->data) {
				if (curr->lTh == false)
					curr = curr->lChild;
				else
					break;
			}
			else if (x > curr->data) {
				if (curr->rTh == false)
					curr = curr->rChild;
				else
					break;
			}
			else if (x == curr->data)
				return;
		}
```

```
	curr = new Node(x);
	if (x < prev->data) {
		curr->rChild = prev;
		curr->lChild = prev->lChild;
		prev->lChild = curr;
		prev->lTh = false;
	}
	else if (x > prev->data) {
		curr->lChild = prev;
		curr->rChild = prev->rChild;
		prev->rChild = curr;
		prev->rTh = false;
	}
}
void CreateTree() {
	while (true) {
		cout << "Enter data of node or -1:\n";
		int x; cin >> x;
		if (x == -1)
			break;
		InsertNode(x);
		ThInorder();
	}
}
void ThInorder() {
	if (root == NULL) {
		cout << "EMPTY TREE\n";
		return;
	}

	Node* curr = root;

	while (curr->lTh == false)
      curr = curr->lChild;

	while (curr != head) {
		cout << curr->data << " ";
		if (curr->rTh == false) {
			curr = curr->rChild;
			while (curr->lTh == false)
		curr = curr->lChild;
		}
		else
			curr = curr->rChild;
	}
	cout << endl;
}
void ThPreorder() {
	if (root == NULL) {
		cout << "EMPTY TREE\n";
		return;
	}

	Node* curr = root;
```

```
	while (curr != head) {
		cout << curr->data << " ";
		if (curr->lTh == false)
			curr = curr->lChild;
		else {
			if (curr->rTh == false)
				curr = curr->rChild;
			else {
				while (curr != head && curr->rTh == true)
					curr = curr->rChild;
			 if (curr == head)
			     break;
				curr = curr->rChild;
			}
		}
	}

	cout << endl;
}
bool Search(Node* curr_root, int x, Node*& curr, Node*& parent) {
	if (curr_root == head)
		return false;

	curr = curr_root;
	if (curr_root->data == x)
		return true;

	parent = curr;
	if (x < curr_root->data)
		return Search(curr_root->lChild, x, curr, parent);
	else
		return Search(curr_root->rChild, x, curr, parent);
}
void deleteNode(int x) {
	Node *curr = NULL, *parent = NULL;

	if (!Search(root, x, curr, parent)) {
		cout << "NOT FOUND\n";
		return;
	}

	if (curr->lTh == false && curr->rTh == false) {
		Node* temp = curr->rChild;
		parent = curr;
		while (temp->lTh == false) {
			parent = temp;
			temp = temp->lChild;
		}
		curr->data = temp->data;
		x = temp->data;
		curr = temp;
	}

	if (curr->lTh == true && curr->rTh == true) {
		if (curr == parent->lChild) {
```

```
                        parent->lChild = curr->lChild;
                        parent->lTh = true;
                }
                else if (curr == parent->rChild) {
                        parent->rChild = curr->rChild;
                        parent->rTh = true;
                }
                delete curr;
        }
        else if (curr->lTh == false && curr->rTh == true) {
                Node* temp = curr->lChild;
                if (parent->lChild == curr)
                        parent->lChild = temp;
                else
                        parent->rChild = temp;
                while (temp->rTh == false)
                        temp = temp->rChild;
                temp->rChild = curr->rChild;
                delete curr;

        }
        else if (curr->lTh == true && curr->rTh == false) {
                Node* temp = curr->rChild;
                if (parent->lChild == curr)
                        parent->lChild = temp;
                else
                        parent->rChild = temp;
                while (temp->lTh == false)
                        temp = temp->lChild;
                temp->lChild = curr->lChild;
                delete curr;
        }
    }
};

int main() {
    TBST tbst;
    while (true) {
        cout << "\nChoose Option:\n";
        cout << "\t1 for Insert\n\t2 for Delete\n\t3 for Traversal\n\t0 to
Exit\n:";
        int choice = 0; cin >> choice;

        if (choice == 0)
            break;

        switch (choice) {
        case 1: {
            cout << "Enter data (for insert): ";
            int x; cin >> x;
            tbst.InsertNode(x);
            cout << "Inorder: "; tbst.ThInorder();
            break;
        }
        case 2: {
```

```
                cout << "Enter data (for delete): ";
                int x; cin >> x;
                tbst.deleteNode(x);
                cout << "Inorder: "; tbst.ThInorder();
                break;
            }
            case 3: {
                cout << "Tree Traversals:\n";
                cout << "Inorder: "; tbst.ThInorder();
                cout << "Preorder: "; tbst.ThPreorder();
                break;
            }
            default:
                cout << "INVALID CHOICE. Try Again.\n";
            }
        }
        cout << "\n----END-----\n";
        return 0;
}
```

TESTCASE 1:

```
Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 100
Inorder: 100

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 150
Inorder: 100 150

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 120
Inorder: 100 120 150

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
```

```
:1
Enter data (for insert): 50
Inorder: 50 100 120 150

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 30
Inorder: 30 50 100 120 150

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 70
Inorder: 30 50 70 100 120 150

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:3
Tree Traversals:
Inorder: 30 50 70 100 120 150
Preorder: 100 50 30 70 150 120

Choose Option:
```

```
Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:2
Enter data (for delete): 50
Inorder: 30 70 100 120 150

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:0

----END-----
```

TESTCASE 2:

```
Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 200
Inorder: 200

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 100
Inorder: 100 200

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 80
Inorder: 80 100 200

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
```

```
Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 60
Inorder: 60 80 100 200

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:1
Enter data (for insert): 40
Inorder: 40 60 80 100 200

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:3
Tree Traversals:
Inorder: 40 60 80 100 200
Preorder: 200 100 80 60 40

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
```

```
Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:2
Enter data (for delete): 100
Inorder: 40 60 80 200

Choose Option:
        1 for Insert
        2 for Delete
        3 for Traversal
        0 to Exit
:0

----END-----
```