```
/*
280
BST_Development_2
Elliot Shaw
*/

#include <iostream>
#include <string>
#include <ctime>
using namespace std;

struct Node {
    int data;
    Node* left, * right;
};

class BST {
private:
    Node* root;
    Node* insert(int, Node*); //helper
    int getSize(Node*); //helper
    void displayInOrder(Node*); //helper
    Node* find(int, Node*);//helper
    int getHeight(Node*, int);//helper
    void displayPreOrder(Node*);//helper
    bool isLeaf(Node*);
    int countLeaves(Node*);//helper
    void treeClear(Node*);//helper
    int sumInRange(int, int, Node*);//helper
public:
    BST();
    //setters
    void treeClear();
    void load(int, int, int);
    void insertNonRecursive(int);
    void del(int);
    //getters
    int getHeight();
    int getSize();
    int treeSize();
    int count(int);
    //utility
    int sumInRange(int, int);
    void displayPreOrder();
    void insert(int);
    void displayInOrder();
    Node* find(int);
    int countLeaves();
    int maxValue();
```

```cpp
}; //BST class


//helper functions

Node* BST::insert(int v, Node* r) {
    if (r == nullptr) {
        r = new Node;
        r->left = r->right = nullptr;
        r->data = v;
        return r;
    }
    else if (v < r->data) {
        r->left = insert(v, r->left);
        return r;
    }
    else {
        r->right = insert(v, r->right);
        return r;
    }
} //insert helper

Node* BST::find(int v, Node* r) {
    while (r != nullptr) {
        if (r->data == v) {
            return r;
        }
        else if (v < r->data) {
            r = r->left;
        }
        else {
            r = r->right;
        }
    }
    return nullptr;
}//find helper

void BST::displayPreOrder(Node* r) {
    if (r != nullptr)
    {
        cout << r->data << endl;
        displayPreOrder(r->left);
        displayPreOrder(r->right);
    }
}//displayPreOrder helper

void BST::displayInOrder(Node* r) {
    if (r != nullptr)
    {
        displayInOrder(r->left);
        cout << r->data << endl;
        displayInOrder(r->right);
    }
} //displayInOrder helper

int BST::getHeight(Node* r, int c) {
    if (r != nullptr)
    {
        c++;
        int left = getHeight(r->left, c);
        int right = getHeight(r->right, c);
        if (left > right) {
            c = left;
        }
        else {
            c = right;
        }
    }
    return c;
}//getHeight helper

bool BST::isLeaf(Node* r) {
    if (r->right == nullptr && r->left == nullptr) {
        return true;
    }
    return false;
}//isLeaf

int BST::countLeaves(Node* r) {
    int c = 0;
    if (isLeaf(r)) {
        return 1;
    }
    if (r->left != nullptr) {
        c += countLeaves(r->left);
    }
    if (r->right != nullptr) {
        c += countLeaves(r->right);
    }
    return c;
}//countLeaves helper

int BST::getSize(Node* r) {
    if (r == nullptr)
        return 0;
    else
        return 1 + getSize(r->right) + getSize(r->left);
} //getSize helper
```

```cpp

void BST::treeClear(Node* r) {
    Node* runner = r;
    if (runner->right != nullptr) {
        treeClear(runner->right);
        runner->right = nullptr;
    }
    if (runner->left != nullptr) {
        treeClear(runner->left);
        runner->left = nullptr;
    }
    free(runner);
    //cout << "freed" << endl;
} //treeClear Helper

int BST::sumInRange(int min, int max, Node* r) {
    int sum = 0;
    if (r != nullptr) {
        if (r->data <= max && r->data >= min) {
            return (sum + r->data + sumInRange(min, max, r->left) + sumInRange(min, max, r->right));
        }
        else {
            return sum + sumInRange(min, max, r->left) + sumInRange(min, max, r->right);
        }
    }
    return 0;
}


//constructors

BST::BST() {
    root = nullptr;
} //BST


//setters

void BST::insert(int v) {
    root = insert(v, root);
} //insert

void BST::load(int c, int min, int max) {
    srand(time(NULL));
    for (int i = 0; i < c; i++) {
        root = insert((rand() % (max - min + 1)) + min, root);
    }
```

```
}//load

void BST::insertNonRecursive(int v) {
    Node* check = root;
    Node* checkptr = nullptr;
    while (check != nullptr) {
        checkptr = check;
        if (v < checkptr->data) {
            check = check->left;
        }
        else {
            check = check->right;
        }
    }
    if (checkptr == nullptr) {
        root = new Node;
        root->left = root->right = nullptr;
        root->data = v;
    }
    else if (v < checkptr->data) {
        checkptr->left = new Node;
        checkptr->left->data = v;
        checkptr->left->left = nullptr;
        checkptr->left->right = nullptr;
    }
    else {
        checkptr->right = new Node;
        checkptr->right->data = v;
        checkptr->right->left = nullptr;
        checkptr->right->right = nullptr;
    }
}//insertNonRecursive


//getters

int BST::getHeight() {
    return getHeight(root, 0);
} //getHeight;

int BST::getSize() {
    return getSize(root);
} //getSize

int BST::maxValue() {
    //pre-req: the tree is not an empty tree
    Node* r = root;
    while (r->right != nullptr) {
        r = r->right;
    }
    return r->data;
}//maxValue

int BST::treeSize() {
    return getSize(root);
}

void BST::treeClear() {
    if (root != nullptr) {
        treeClear(root);
    }
    root = nullptr;
}//treeClear


//utility

Node* BST::find(int v) {
    return find(v, root);
}//find

int BST::count(int v) {
    Node* r = root;
    int count = 0;
    while (r != nullptr) {
        if (r->data == v) {
            count++;
        }
        if (v < r->data) {
            r = r->left;
        }
        else {
            r = r->right;
        }
    }
    return count;
}//count

void BST::displayInOrder() {
    displayInOrder(root);
} //displayInOrder

void BST::displayPreOrder() {
    displayPreOrder(root);
}//displayPreOrder

int BST::countLeaves() {
    return countLeaves(root);
```

```
}//countLeaves

void BST::del(int v) {
    Node* t = find(v, root);
    int tval = t->data;
    Node* p = root;
    if (t != root) {
        while (p->left != t && p->right != t) {
            if (tval < p->data) {
                p = p->left;
            }
            else {
                p = p->right;
            }
        }
        if (p->left == t) {
            p->left = nullptr;
        }
        else {
            p->right = nullptr;
        }
    }
    else {
        root = nullptr;
    }
    Node* ip = nullptr;
    while (!isLeaf(t)) {
        Node* i = t;

        while (!isLeaf(i)) {
            ip = i;
            if (i->right != nullptr) {
                i = i->right;
            }
            else {
                i = i->left;
            }
        }
        insert(i->data);
        if (ip->right != nullptr) {
            ip->right = nullptr;
        }
        else {
            ip->left = nullptr;
        }
        free(i);
    }
    free(t);
}//delete
```

```cpp

int BST::sumInRange(int min, int max) {
    int sum = 0;
    if (root != nullptr) {
        return sumInRange(min, max, root);
    }
    else {
        return sum;
    }
}//sumInRange

int main() {
    BST bst1 = BST();

    int x[] = { 40, 10, 80, 70, 50, 30, 10, 90, 10, 60, 5, 25, 35 };
    int upto = size(x);

    for (int i = 0; i < upto; i++) {
        bst1.insert(x[i]);
    }

    cout << "Tree before treeClear: \n"; bst1.displayInOrder();
    cout << "Tree size: " << bst1.treeSize() << endl << endl;

    cout << "sum in range [4,31]: " << bst1.sumInRange(4, 31) << endl;
    cout << "should be 90 ..." << endl << endl;
    cout << "sum in range [60,80]: " << bst1.sumInRange(60, 80) << endl;
    cout << "should be 210 ..." << endl << endl;

    bst1.treeClear();
    cout << "Cleared the tree" << endl;
    cout << "Tree size: " << bst1.treeSize() << endl;

    /*
    BST bst1 = BST();
    cout << "Size: " << bst1.getSize() << endl << endl;

    bst1.insert(20);
    bst1.displayInOrder();
    cout << "Size: " << bst1.getSize() << endl << endl;

    bst1.insert(10);
    bst1.insert(30);
    bst1.displayInOrder();
    cout << "Size: " << bst1.getSize() << endl << endl;

    bst1.insert(5);
    bst1.insert(40);
    bst1.insert(25);
    bst1.displayInOrder();
    cout << "Size: " << bst1.getSize() << endl << endl;

    bst1.insert(0);
    bst1.insert(2);
    bst1.insert(-5);
    bst1.insert(-2);
    bst1.displayInOrder();
    cout << "Size: " << bst1.getSize() << endl << endl;

    BST bst2 = BST();
    bst2.load(10, -20, 20);
    bst2.displayInOrder();
    cout << "Size: " << bst2.getSize() << endl << endl;
    bst2.insertNonRecursive(20);
    bst2.displayInOrder();
    cout << "Size: " << bst2.getSize() << endl << endl;
    cout << "address of 20: " << bst2.find(20) << endl << endl;
    bst2.insert(45);
    bst2.insert(45);
    bst2.insert(45);
    bst2.insert(45);
    bst2.insert(45);
    cout << "count of 45s: " << bst2.count(45) << endl << endl;
    bst2.displayInOrder();
    cout << endl;
    bst2.del(20);
    bst2.displayInOrder();
    cout << endl;
    cout << "height of bst2: " << bst2.getHeight() << endl << endl;
    bst2.displayPreOrder();
    cout << endl << "number of leaves: " << bst2.countLeaves() << endl << endl;
    cout << "Max value: " << bst2.maxValue(); */
} //main
```