Eric Blasko

CSE 330 Data Structures

Winter 2018

Lab 8 – Set

* **Status**

100% complete

* **Time Complexity**

**Methods of Set:**

The following methods are O(1) as they are constant in time

* Set():
* Set(Set && rhs);
* Set & operator=(const Set & rhs);
* Set & operator=(Set && rhs);
* bool empty() const
* unsigned int size()
* iterator end() const;

The following methods are O(logn) were n is the size of the set. Each method only has to iterate over a portion of the set.

* const Comparable & findMin() const;
* BinaryNode \* findMin(BinaryNode \* t) const;
* iterator insert(const Comparable & x)
* iterator insert(Comparable && x)
* iterator insert(const Comparable & x, BinaryNode \* p, BinaryNode \* & t);
* iterator insert(Comparable && x, BinaryNode \* p, BinaryNode \* & t);
* unsigned int count(const Comparable & x) const;
* unsigned int count(const Comparable & x, BinaryNode \* t) const;
* void erase(const Comparable & x)
* iterator begin() const;
* iterator find(const Comparable &) const;
* iterator find(const Comparable & x, BinaryNode \* t) const;
* void erase(iterator itr);
* void erase(const Comparable & x, BinaryNode \* & t);

The following methods are O(n) were n is the Size of the set. Each method much iterate over every element in the set.

* ~Set()
* Set(const Set & rhs)
* void clear()
* BinaryNode \* clone(BinaryNode \* t) const;
* void clear(BinaryNode \* & t);
* void printTree(ostream & out = cout) const;
* void printTree(ostream & out, BinaryNode \* t, string indent, const string & tag) const;

**Methods of Set\_iterator**

The following are O(1) as they are constant in time

* Set\_iterator()
* Set\_iterator(const Set\_iterator & it)
* Set\_iterator(Set\_iterator && it)
* ~Set\_iterator() = default;
* Set\_iterator & operator=(const Set\_iterator & itr)
* Set\_iterator & operator=(Set\_iterator && itr)
* Set\_iterator(BinaryNode \* t)
* bool operator==(Set\_iterator itr) const
* bool operator!=(Set\_iterator itr) const
* Comparable & operator\*()

The following are O(logn) were n is the size of the set

* Set\_iterator & operator++();

The following are O(n) were n is the size of the set

* Set\_iterator operator++(int);
* **Source Code**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Set.h

\* 03/07/2018

\* This class is a replica of the class set in the standard library. While similar to the binary search

\* tree, each node also has a pointer to its parent, along with its pointers to its children. Iterators

\* allow for easy traversal of the tree.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef SET\_H

#define SET\_H

#include <ostream>

#include <cassert>

using namespace std;

template <typename Comparable> class Set\_iterator;

template <typename Comparable>

class Set

{

public:

typedef Set\_iterator<Comparable> iterator;

Set(): root{nullptr}, theSize{0} {}

Set(const Set & rhs) {root = clone(rhs.root); theSize = rhs.theSize;}

Set(Set && rhs);

~Set() {clear(root);}

Set & operator=(const Set & rhs);

Set & operator=(Set && rhs);

const Comparable & findMin() const;

unsigned int count(const Comparable & x) const;

bool empty() const {return root == nullptr;}

void printTree(ostream & out = cout) const;

void clear() {clear(root);}

iterator insert(const Comparable & x) { insert(x,nullptr, root);}

iterator insert(Comparable && x) {insert(move(x),nullptr,root);}

void erase(const Comparable & x) {erase(x,root);}

unsigned int size() {return theSize;}

iterator begin() const;

iterator end() const;

iterator find(const Comparable &) const;

void erase(iterator itr);

private:

struct BinaryNode

{

Comparable element;

BinaryNode \* parent;

BinaryNode \* left;

BinaryNode \* right;

BinaryNode(const Comparable & theElement, BinaryNode \* p, BinaryNode \* lt, BinaryNode \* rt):

element{theElement}, parent{p}, left{lt}, right{rt} {}

BinaryNode(Comparable && theElement, BinaryNode \* p, BinaryNode \* lt, BinaryNode \* rt):

element{move(theElement)}, parent{p}, left{lt}, right{rt} {}

};

BinaryNode \* root;

unsigned int theSize;

iterator insert(const Comparable & x, BinaryNode \* p, BinaryNode \* & t);

iterator insert(Comparable && x, BinaryNode \* p, BinaryNode \* & t);

void erase(const Comparable & x, BinaryNode \* & t);

BinaryNode \* findMin(BinaryNode \* t) const;

unsigned int count(const Comparable & x, BinaryNode \* t) const;

void clear(BinaryNode \* & t);

void printTree(ostream & out, BinaryNode \* t, string indent, const string & tag) const;

BinaryNode \* clone(BinaryNode \* t) const;

iterator find(const Comparable & x, BinaryNode \* t) const;

friend class Set\_iterator<Comparable>;

};

//Copies entire contents of a tree to create another identical instance

template <typename Comparable>

typename Set<Comparable>::BinaryNode \* Set<Comparable>::clone(BinaryNode \* t) const

{

if(t == nullptr)

return nullptr;

else

{

BinaryNode \* temp = new BinaryNode(t->element, nullptr, clone(t->left), clone(t->right));

assert(temp);

if(temp->left != nullptr)

temp->left->parent = temp;

if(temp->right != nullptr)

temp->right->parent = temp;

return temp;

}

}

//Move constructor

template <typename Comparable>

Set<Comparable>::Set(Set && rhs): root{rhs.root}, theSize{rhs.theSize}

{

rhs.root = nullptr;

rhs.theSize = 0;;

}

//deletes every element in the tree starting with leaves, and working there way back to root. root

//is left null

template <typename Comparable>

void Set<Comparable>::clear(BinaryNode \* & t)

{

if(t != nullptr)

{

clear(t->left);

clear(t->right);

delete t;

}

t = nullptr;

theSize = 0;

}

//Copies contents of source Set to current Set

template <typename Comparable>

Set<Comparable> & Set<Comparable>::operator=(const Set & rhs)

{

Set copy = rhs;

swap(\*this, copy);

return \*this;

}

//Moves contents of source Set to current Set

template <typename Comparable>

Set<Comparable> & Set<Comparable>::operator=(Set && rhs)

{

root = rhs.root;

rhs.root = nullptr;

return \*this;

}

//helper function which tells findMin to start at root

template <typename Comparable>

const Comparable & Set<Comparable>::findMin() const

{

BinaryNode \* t = findMin(root);

if(t == nullptr)

{

BinaryNode \* temp = new BinaryNode(Comparable(),nullptr,nullptr,nullptr);

return temp->element;

}

return t->element;

}

//Recursivly calls until reaches bottom left leaf.

template <typename Comparable>

typename Set<Comparable>::BinaryNode \* Set<Comparable>::findMin(BinaryNode \* t) const

{

if (t == nullptr)

return nullptr;

if (t->left == nullptr)

return t;

return findMin(t->left);

}

//helper functions for printing contents of tree. Tells to start at root

template <typename Comparable>

void Set<Comparable>::printTree(ostream & out) const

{

cout << "Print Tree\n";

printTree(out,root, "", "");

}

//Recursivly goes through elements of tree to print to console. Each deeper call increases indent

template <typename Comparable>

void Set<Comparable>::printTree(ostream & out, BinaryNode \* t, string indent, const string & tag) const

{

if (t == nullptr)

return;

out << indent << tag << t->element << endl;

indent += " ";

printTree(out, t->left, indent, "L ");

printTree(out, t->right, indent, "R ");

}

//recursivily compares values to find correct place to insert in tree

template <typename Comparable>

typename Set<Comparable>::iterator Set<Comparable>::insert(const Comparable & x, BinaryNode \* p, BinaryNode \* & t)

{

if(t == nullptr)

{

t = new BinaryNode(x,p, nullptr, nullptr);

assert(t);

theSize++;

return t;

}

else if (x < t->element)

insert(x, t, t->left);

else if (x > t->element)

insert(x, t, t->right);

else

return t;

}

//recursivily compares value to find correct place to insert in tree

template <typename Comparable>

typename Set<Comparable>::iterator Set<Comparable>::insert(Comparable && x, BinaryNode \* p, BinaryNode \* & t)

{

if(t == nullptr)

{

t = new BinaryNode(move(x), p, nullptr,nullptr);

assert(t);

theSize++;

return t;

}

else if(x < t->element)

insert(move(x),t, t->left);

else if(x > t->element)

insert(move(x),t, t->right);

else

return t;

}

//helper function which tells contain to start at root

template <typename Comparable>

unsigned int Set<Comparable>::count(const Comparable & x) const

{

return count(x,root);

}

//recursively calls to one side of tree till found or at nullptr

template <typename Comparable>

unsigned int Set<Comparable>::count(const Comparable & x, BinaryNode \* t) const

{

if(t == nullptr)

return false;

else if(x < t->element)

return count(x,t->left);

else if(x > t->element)

return count(x,t->right);

else

return true;

}

//first part searches tree bases on size of element. If element has two children, find

//the min value to the right of that node and assing to that position. then find that

//node and erase. If one child move child to spot being deleted. If no children

//delete node.

template <typename Comparable>

void Set<Comparable>::erase(const Comparable & x, BinaryNode \* & t)

{

if(t == nullptr)

return;

if(x < t->element)

erase(x,t->left);

else if(x > t->element)

erase(x,t->right);

else if(t->left != nullptr && t->right != nullptr)

{

t->element = findMin(t->right)->element;

erase(t->element,t->right);

}

else

{

BinaryNode \*oldNode = t;

BinaryNode \*p = t->parent;

t = (t->left != nullptr) ? t->left : t->right;

if(t != nullptr)

t->parent = p;

delete oldNode;

theSize--;

}

}

//Returns an iterator to the begining of the Set which is a left slide

template <typename Comparable>

typename Set<Comparable>::iterator Set<Comparable>::begin() const

{

BinaryNode \*temp = root;

while(temp && temp->left)

temp = temp->left;

return iterator(temp);

}

//Return an iterator to the end of Set which is a nullptr

template <typename Comparable>

typename Set<Comparable>::iterator Set<Comparable>::end() const

{

return iterator(nullptr);

}

//helper function to find a value x in Set. Tells to start at root

template <typename Comparable>

typename Set<Comparable>::iterator Set<Comparable>::find(const Comparable & x) const

{

return find(x,root);

}

//returns an iterator of were the value of x is found in the Set.

template <typename Comparable>

typename Set<Comparable>::iterator Set<Comparable>::find(const Comparable & x, BinaryNode \* t) const

{

if(t == nullptr)

return iterator{};

else if(x < t->element)

return find(x,t->left);

else if(x > t->element)

return find(x,t->right);

else

return iterator{t};

}

//uses iterator to find value to be deleted. if source node has children, redirect

//the next node to point back to the source nodes parent, then point the parent at

//the next node

template <typename Comparable>

void Set<Comparable>::erase(Set<Comparable>::iterator itr)

{

if(itr == end())

return;

BinaryNode\* temp;

if(itr.current->left != nullptr && itr.current->right != nullptr) //two children

{

temp = itr.current->right;

while(temp->left != nullptr)

{

temp = temp->left;

}

itr.current->element = temp->element;

}

else

temp = itr.current;

//adjust child pointer

BinaryNode \* p = temp->parent;

if(p != nullptr)

{

if(p->left == temp)

p->left = (temp->left != nullptr) ? temp->left : temp->right;

else

p->right = (temp->right != nullptr) ? temp->left : temp->right;

}

//adjust parent pointer

if(temp->left != nullptr)

temp->left->parent = p;

else if(temp->right != nullptr)

temp->right->parent = p;

delete temp;

theSize--;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*class Set\_iterator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename Comparable>

class Set\_iterator

{

public:

typedef typename Set<Comparable>:: BinaryNode BinaryNode;

Set\_iterator() = default;

Set\_iterator(const Set\_iterator & it) = default;

Set\_iterator(Set\_iterator && it) = default;

~Set\_iterator() = default;

Set\_iterator & operator=(const Set\_iterator & itr) = default;

Set\_iterator & operator=(Set\_iterator && itr) = default;

Set\_iterator(BinaryNode \* t): current{t} {}

bool operator==(Set\_iterator itr) const {return current == itr.current;}

bool operator!=(Set\_iterator itr) const {return current != itr.current;}

Set\_iterator & operator++();

Set\_iterator operator++(int);

Comparable & operator\*() {return current->element;}

protected:

BinaryNode \* current;

friend class Set<Comparable>;

};

//inorder traversal, pre-increment

template <typename Comparable>

Set\_iterator<Comparable> & Set\_iterator<Comparable>::operator++()

{

if(current->right)

{

current = current->right;

while(current->left)

current = current->left;

}

else

{

BinaryNode \* child = current;

current = current->parent;

while(current && current->right == child)

{

child = current;

current = current->parent;

}

}

return \*this;

}

//inorder traversal, post-increment

template <typename Comparable>

Set\_iterator<Comparable> Set\_iterator<Comparable>::operator++(int)

{

Set\_iterator<Comparable> clone(\*this);

++(\*this);

return clone;

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Set\_test1.cpp

\* 03/07/2018

\* This program test the methods of the class Set.h. The program will demonstraight the

\* insert, findMin, findMax, remove, and print methods. PrintTree will print elements according to the

\* depth that they are in the tree

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "Set.h"

using namespace std;

//Main function to test methods of Set.h

int main()

{

Set<int> tree;

tree.insert(6);

tree.insert(8);

tree.insert(2);

tree.insert(1);

tree.insert(5);

tree.insert(3);

tree.insert(4);

tree.insert(1);

tree.printTree();

cout << "Min = " << tree.findMin() << endl;

cout << "Max = " << tree.findMax() << endl;

cout << "Remove 2\n";

tree.remove(2);

tree.printTree();

cout << "Contains 2? " << tree.contains(2) << endl;

cout << "Contains 4? " << tree.contains(4) << endl;

cout << "Copy Constructor\n";

Set<int> copy(tree);

cout << "Remove 6\n";

copy.remove(6);

copy.printTree();

cout << "Inorder Traversal\n";

copy.inorder();

cout << endl;

cout << "Clear tree\n";

copy.makeEmpty();

copy.printTree();

cout << copy.findMin();

cout << copy.findMax();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Set\_test2.cpp

\* 03/07/2018

\* This program test the different traversal options available int the Set.h

\* class. The following traversals follow the pattersn: inorder(left,root,right),

\* postorder(left,right,root), and preorder(root,left,right). later altercations to Set.h

\* will only have inorder traversal

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "Set.h"

using namespace std;

//Main function that test methods of Set.h

int main()

{

Set<int> complete\_tree;

complete\_tree.insert(20);

complete\_tree.insert(10);

complete\_tree.insert(30);

complete\_tree.insert(35);

complete\_tree.insert(25);

complete\_tree.insert(5);

complete\_tree.insert(15);

complete\_tree.printTree();

cout << "Inorder Traversal\n";

complete\_tree.inorder();

cout << endl;

cout << "Postorder Traversal\n";

complete\_tree.postorder();

cout << endl;

cout << "Preorder Traversal\n";

complete\_tree.preorder();

cout << endl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Set\_test3.cpp

\* 03/07/2018

\* This program test the size(),empty(),insert() and erase() methods of Set.h. After

\* performing some operations, results will be printed to console.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "Set.h"

using namespace std;

//Main function to test methods in Set.h

int main()

{

Set<int> tree;

cout << "Size = " << tree.size() << endl;

cout << "Empty? " << tree.empty() << endl;

tree.insert(6);

tree.insert(8);

tree.insert(2);

tree.insert(1);

tree.insert(5);

tree.insert(3);

tree.insert(4);

tree.insert(5); // duplicate, will be ignored

tree.printTree();

cout << "Count 5 = " << tree.count(5) << endl;

cout << "Size = " << tree.size() << endl;

cout << "Empty? " << tree.empty() << endl;

cout << "Erase 2\n";

tree.erase(2);

tree.printTree();

cout << "Size = " << tree.size() << endl;

cout << "Empty? " << tree.empty() << endl;

cout << "Count 2 = " << tree.count(2) << endl;

cout << "Erase 6\n";

tree.erase(6);

tree.printTree();

cout << "Copy Constructor\n";

Set<int> copy(tree);

copy.printTree();

cout << "Copy Size = " << copy.size() << endl;

cout << "Clear Copy\n";

copy.clear();

cout << "Size = " << copy.size() << endl;

cout << "Empty? " << copy.empty() << endl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Set\_test4.cpp

\* 03/07/2018

\* This program test the size(),empty(),insert() and erase() as well as the iterator

\* methods for find(), post and pre increments and erase() from Set.h. After

\* performing some operations, results will be printed to console.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "Set.h"

using namespace std;

//Main function to test methods in Set.h

int main()

{

Set<int> tree;

cout << "Size = " << tree.size() << endl;

cout << "Empty? " << tree.empty() << endl;

tree.insert(6);

tree.insert(8);

tree.insert(2);

tree.insert(1);

tree.insert(5);

tree.insert(3);

tree.insert(4);

tree.insert(5); // duplicate, will be ignored

tree.printTree();

cout << "Count 5 = " << tree.count(5) << endl;

cout << "Size = " << tree.size() << endl;

cout << "Empty? " << tree.empty() << endl;

cout << "Erase 2\n";

tree.erase(2);

tree.printTree();

cout << "Size = " << tree.size() << endl;

cout << "Empty? " << tree.empty() << endl;

cout << "Count 2 = " << tree.count(2) << endl;

cout << "Erase 6\n";

tree.erase(6);

tree.printTree();

cout << "Copy Constructor\n";

Set<int> copy(tree);

copy.printTree();

cout << "Copy Size = " << copy.size() << endl;

cout << "Clear Copy\n";

copy.clear();

cout << "Size = " << copy.size() << endl;

cout << "Empty? " << copy.empty() << endl;

Set<int>::iterator i;

cout << "Iterate Over and Print Elements\n";

for (i = tree.begin(); i != tree.end(); i++)

cout << \*i << " ";

cout << endl;

cout << "Find, return iterator, advance it\n";

i = tree.find(5);

if (i != tree.end()) {

cout << "Found 5\n";

++i;

cout << "Next value = " << \*i << endl;

} else

cout << "Didn't find 5\n";

i = tree.find(6);

if (i != tree.end())

cout << "Found 6\n";

else

cout << "Didn't find 6\n";

cout << "Insert 2, advance iter\n";

i = tree.insert(2);

cout << \*++i << endl;

tree.printTree();

cout << "Size = " << tree.size() << endl;

cout << "Insert 2 again\n";

i = tree.insert(2);

cout << \*i << endl;

cout << "Size = " << tree.size() << endl;

cout << "Iterate Over and Print Elements\n";

for (i = tree.begin(); i != tree.end(); i++)

cout << \*i << " ";

cout << endl;

cout << "Erase 2 with iter\n";

i = tree.find(2);

tree.erase(i);

tree.printTree();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Set\_test5.cpp

\* 03/07/2018

\* This program test the size(),empty(),insert() and erase() as well as the iterator

\* methods for find(), post and pre increments and erase() from Set.h. After

\* performing some operations, contents will be displayed to console were user can

\* enter value to erase from Set

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "Set.h"

using namespace std;

//Main function to test methods in Set.h

int main()

{

Set<int> tree;

Set<int>::iterator i;

int n;

tree.insert(100);

tree.insert(50);

tree.insert(110);

tree.insert(40);

tree.insert(75);

tree.insert(30);

tree.insert(60);

tree.insert(80);

tree.insert(55);

tree.insert(90);

tree.insert(59);

tree.insert(57);

tree.insert(56);

tree.insert(58);

tree.printTree();

cout << "Iterate Over and Print Elements\n";

for (i = tree.begin(); i != tree.end(); i++)

cout << \*i << " ";

cout << endl;

//Set<int> copy(tree); // save a copy

cin >> n;

while (n >= 0) {

i = tree.find(n);

tree.erase(i);

//tree.erase(n); // test the other erase()

tree.printTree();

cout << "Iterate Over and Print Elements\n";

for (i = tree.begin(); i != tree.end(); i++)

cout << \*i << " ";

cout << endl;

//tree = copy; // restore original tree

cin >> n;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Eric Blasko

\* Set\_test5.cpp

\* 03/07/2018

\* This program test all methods from class Set.h. Each test will include an assert which

\* will verify that the each method works as intended

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <cassert>

#include "Set.h"

using namespace std;

//Main function to test methods in Set.h

int main()

{

Set<int> s;

assert(s.size() == 0);

assert(s.empty());

s.insert(10);

Set<int>::iterator iter = s.begin();

assert(\*iter == 10);

s.insert(6);

s.insert(6);

assert(s.count(6) == 1);

assert(s.count(10) == 1);

assert(s.count(12) == 0);

iter = s.begin();

assert(\*iter == 6);

++iter;

assert(\*iter == 10);

++iter;

assert(iter == s.end());

s.insert(7);

s.insert(9);

s.insert(9);

s.insert(8);

s.insert(11);

iter = s.begin();

assert(\*iter == 6);

++iter;

assert(\*iter == 7);

++iter;

assert(\*iter == 8);

++iter;

assert(\*iter == 9);

++iter;

assert(\*iter == 10);

++iter;

assert(\*iter == 11);

Set<int> s2;

s2.insert(3);

s2.insert(7);

s2.insert(-1);

s2.insert(16);

s2.insert(11);

s2.insert(4);

iter = s2.find(3);

assert(\*iter == 3);

iter = s2.find(888);

assert(iter == s2.end());

s2.erase(7);

iter = s2.begin();

assert(\*iter == -1);

++iter;

assert(\*iter == 3);

++iter;

assert(\*iter == 4);

++iter;

assert(\*iter == 11);

++iter;

assert(\*iter == 16);

++iter;

assert(iter == s2.end());

s2.erase(16);

iter = s2.begin();

assert(\*iter == -1);

++iter;

assert(\*iter == 3);

++iter;

assert(\*iter == 4);

++iter;

assert(\*iter == 11);

++iter;

assert(iter == s2.end());

s2.erase(3);

iter = s2.begin();

assert(\*iter == -1);

++iter;

assert(\*iter == 4);

++iter;

assert(\*iter == 11);

++iter;

assert(iter == s2.end());

s2.erase(11);

iter = s2.begin();

assert(\*iter == -1);

++iter;

assert(\*iter == 4);

++iter;

assert(iter == s2.end());

s2.erase(-1);

iter = s2.begin();

assert(\*iter == 4);

++iter;

assert(iter == s2.end());

s2.erase(4);

iter = s2.begin();

assert(iter == s2.end());

cout << "All tests passed." << endl;

}

* **Sample Runs**

**Test 1**

Script started on 2018-03-06 20:32:47-0800

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ -c Set\_test1.cpp

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ Set\_test1.o

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ ./a.out

Print Tree

6

L 2

L 1

R 5

L 3

R 4

R 8

Min = 1

Max = 8

Remove 2

Print Tree

6

L 3

L 1

R 5

L 4

R 8

Contains 2? 0

Contains 4? 1

Copy Constructor

Remove 6

Print Tree

8

L 3

L 1

R 5

L 4

Inorder Traversal

1 3 4 5 8

Clear tree

Print Tree

Segmentation fault (core dumped)

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ exit

Script done on 2018-03-06 20:33:10-0800

**Test 2**

Script started on 2018-03-06 20:33:45-0800

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ -c Set\_test2.cpp

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ Set\_test2.o

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ ./a.out

Print Tree

20

L 10

L 5

R 15

R 30

L 25

R 35

Inorder Traversal

5 10 15 20 25 30 35

Postorder Traversal

5 15 10 25 35 30 20

Preorder Traversal

20 10 5 15 30 25 35

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ exit

Script done on 2018-03-06 20:34:06-0800

**Test 3**

Script started on 2018-03-06 20:38:16-0800

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g[Kg++ -c Set\_tets[K[Kst3.cpp

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ Set\_test2[K3.p[Ko

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ ./a.out

Size = 0

Empty? 1

Print Tree

6

L 2

L 1

R 5

L 3

R 4

R 8

Count 5 = 1

Size = 7

Empty? 0

Erase 2

Print Tree

6

L 3

L 1

R 5

L 4

R 8

Size = 6

Empty? 0

Count 2 = 0

Erase 6

Print Tree

8

L 3

L 1

R 5

L 4

Copy Constructor

Print Tree

8

L 3

L 1

R 5

L 4

Copy Size = 5

Clear Copy

Size = 0

Empty? 1

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ exit

Script done on 2018-03-06 20:38:44-0800

**Test 4**

Script started on 2018-03-06 20:39:02-0800

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ -c Set\_test4.cpp

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ Set\_test4.o

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ ./a.out

Size = 0

Empty? 1

Print Tree

6

L 2

L 1

R 5

L 3

R 4

R 8

Count 5 = 1

Size = 7

Empty? 0

Erase 2

Print Tree

6

L 3

L 1

R 5

L 4

R 8

Size = 6

Empty? 0

Count 2 = 0

Erase 6

Print Tree

8

L 3

L 1

R 5

L 4

Copy Constructor

Print Tree

8

L 3

L 1

R 5

L 4

Copy Size = 5

Clear Copy

Size = 0

Empty? 1

Iterate Over and Print Elements

1 3 4 5 8

Find, return iterator, advance it

Found 5

Next value = 8

Didn't find 6

Insert 2, advance iter

3

Print Tree

8

L 3

L 1

R 2

R 5

L 4

Size = 6

Insert 2 again

2

Size = 6

Iterate Over and Print Elements

1 2 3 4 5 8

Erase 2 with iter

Print Tree

8

L 3

L 1

R 5

L 4

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ exit

Script done on 2018-03-06 20:39:34-0800

**Test 5**

Script started on 2018-03-06 20:40:37-0800

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ -c Set\_test5.cpp

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ Set)[K\_test4[K5.o

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ ./a.out

Print Tree

100

L 50

L 40

L 30

R 75

L 60

L 55

R 59

L 57

L 56

R 58

R 80

R 90

R 110

Iterate Over and Print Elements

30 40 50 55 56 57 58 59 60 75 80 90 100 110

20

Print Tree

100

L 50

L 40

L 30

R 75

L 60

L 55

R 59

L 57

L 56

R 58

R 80

R 90

R 110

Iterate Over and Print Elements

30 40 50 55 56 57 58 59 60 75 80 90 100 110

40

Print Tree

100

L 50

L 30

R 75

L 60

L 55

R 59

L 57

L 56

R 58

R 80

R 90

R 110

Iterate Over and Print Elements

30 50 55 56 57 58 59 60 75 80 90 100 110

56

Print Tree

100

L 50

L 30

R 75

L 60

L 55

R 59

L 57

R 58

R 80

R 90

R 110

Iterate Over and Print Elements

30 50 55 57 58 59 60 75 80 90 100 110

100

Print Tree

110

L 50

L 30

R 75

L 60

L 55

R 59

L 57

R 58

R 80

R 90

Iterate Over and Print Elements

30 50 55 57 58 59 60 75 80 90 110

^C

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ exit

Script done on 2018-03-06 20:41:17-0800

**Test 6**

Script started on 2018-03-06 20:41:25-0800

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ -c Set\_test6.cpp

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ g++ Set\_test6.o

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ ./a.out

All tests passed.

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ ^C

]0;005670557@csusb.edu@csevnc:~/cse330/lab8[005670557@csusb.edu@csevnc lab8]$ exit

Script done on 2018-03-06 20:42:08-0800