

**มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ**

**KING MONGKUT’S UNIVERSITY OF TECHNOLOGY NORTH BANGKOK**

**ASSIGNMENT 2 BINARY SEARCH TREE**

**เสนอ**

**อาจารย์ประดิษฐ์ พิทักษ์เสถียรกุล**

**จัดทำโดย**

**นายวรศิษฏ์ ภู่สุวรรณ์**

**ITI-2RB รหัส 6206021421237**

**รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithm**

**(รหัสรายวิชา 060223119)**

**ภาคการศึกษาที่ 1 ปีการศึกษา 2563**

**สาขา เทคโนโลยีสารสนเทศและการจัดการอุตสาหกรรม ภาควิชา เทคโนโลยีสารสนเทศ**

**คณะเทคโนโลยีและการจัดการอุตสาหกรรม**

**มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ ปราจีนบุรี**

**Code ไฟล์ BinarySearchTree.h**

#ifndef BINARY\_SEARCH\_TREE\_H\_

#define BINARY\_SEARCH\_TREE\_H\_

#include <iostream>

template <class Comparable>

class BinarySearchTree;

template <class Comparable>

class BinaryNode

{

Comparable element;

BinaryNode \*left;

BinaryNode \*right;

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

: element( theElement ), left( lt ), right( rt ) { }

friend class BinarySearchTree<Comparable>;

};

template <class Comparable>

class BinarySearchTree

{

public:

explicit BinarySearchTree( const Comparable & notFound );

BinarySearchTree( const BinarySearchTree & rhs );

~BinarySearchTree( );

const Comparable & findMin( ) const;

const Comparable & findMax( ) const;

const Comparable & find( const Comparable & x ) const;

bool isEmpty( ) const;

void printTree( ) const;

void printPre( ) const;

void printIn( ) const;

void printPost( ) const;

void makeEmpty( );

void insert( const Comparable & x );

void remove( const Comparable & x );

const BinarySearchTree & operator=( const BinarySearchTree & rhs );

private:

BinaryNode<Comparable> \*root;

const Comparable ITEM\_NOT\_FOUND;

const Comparable & elementAt( BinaryNode<Comparable> \*t ) const;

void insert( const Comparable & x, BinaryNode<Comparable> \* & t ) const;

void remove( const Comparable & x, BinaryNode<Comparable> \* & t ) const;

BinaryNode<Comparable> \* findMin( BinaryNode<Comparable> \*t ) const;

BinaryNode<Comparable> \* findMax( BinaryNode<Comparable> \*t ) const;

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

void makeEmpty( BinaryNode<Comparable> \* & t ) const;

void printTree( BinaryNode<Comparable> \*t ) const;

void printPre( BinaryNode<Comparable> \*t ) const;

void printIn( BinaryNode<Comparable> \*t ) const;

void printPost( BinaryNode<Comparable> \*t ) const;

BinaryNode<Comparable> \* clone( BinaryNode<Comparable> \*t ) const;

};

#endif

**Code ไฟล์ dsexceptions.h**

#ifndef DSEXCEPTIONS\_H\_

#define DSEXCEPTIONS\_H\_

class Underflow { };

class Overflow { };

class OutOfMemory { };

class BadIterator { };

#endif

**Code ไฟล์ BinarySearchTree.cpp**

#include "BinarySearchTree.h"

#include <iostream>

template <class Comparable>

BinarySearchTree<Comparable>::BinarySearchTree( const Comparable & notFound ) :

root( NULL ), ITEM\_NOT\_FOUND( notFound )

{

}

template <class Comparable>

BinarySearchTree<Comparable>::

BinarySearchTree( const BinarySearchTree<Comparable> & rhs ) :

root( NULL ), ITEM\_NOT\_FOUND( rhs.ITEM\_NOT\_FOUND )

{

\*this = rhs;

}

template <class Comparable>

BinarySearchTree<Comparable>::~BinarySearchTree( )

{

makeEmpty( );

}

/\*\*

\* Insert x into the tree; duplicates are ignored.

\*/

template <class Comparable>

void BinarySearchTree<Comparable>::insert( const Comparable & x )

{

insert( x, root );

}

/\*\*

\* Remove x from the tree. Nothing is done if x is not found.

\*/

template <class Comparable>

void BinarySearchTree<Comparable>::remove( const Comparable & x )

{

remove( x, root );

}

/\*\*

\* Find the smallest item in the tree.

\* Return smallest item or ITEM\_NOT\_FOUND if empty.

\*/

template <class Comparable>

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

{

return elementAt( findMin( root ) );

}

/\*\*

\* Find the largest item in the tree.

\* Return the largest item of ITEM\_NOT\_FOUND if empty.

\*/

template <class Comparable>

const Comparable & BinarySearchTree<Comparable>::findMax( ) const

{

return elementAt( findMax( root ) );

}

/\*\*

\* Find item x in the tree.

\* Return the matching item or ITEM\_NOT\_FOUND if not found.

\*/

template <class Comparable>

const Comparable & BinarySearchTree<Comparable>::

find( const Comparable & x ) const

{

return elementAt( find( x, root ) );

}

/\*\*

\* Make the tree logically empty.

\*/

template <class Comparable>

void BinarySearchTree<Comparable>::makeEmpty( )

{

makeEmpty( root );

}

/\*\*

\* Test if the tree is logically empty.

\* Return true if empty, false otherwise.

\*/

template <class Comparable>

bool BinarySearchTree<Comparable>::isEmpty( ) const

{

return root == NULL;

}

/\*\*

\* Print the tree contents in sorted order.

\*/

template <class Comparable>

void BinarySearchTree<Comparable>::printTree( ) const

{

if( isEmpty( ) )

cout << "Empty tree" << endl;

else

printTree( root );

}

template <class Comparable>

void BinarySearchTree<Comparable>::printPre( ) const

{

if( isEmpty( ) )

cout << "Empty tree" << endl;

else

printPre( root );

}

template <class Comparable>

void BinarySearchTree<Comparable>::printIn( ) const

{

if( isEmpty( ) )

cout << "Empty tree" << endl;

else

printIn( root );

}

template <class Comparable>

void BinarySearchTree<Comparable>::printPost( ) const

{

if( isEmpty( ) )

cout << "Empty tree" << endl;

else

printPost( root );

}

template <class Comparable>

const BinarySearchTree<Comparable> &

BinarySearchTree<Comparable>::

operator=( const BinarySearchTree<Comparable> & rhs )

{

if( this != &rhs )

{

makeEmpty( );

root = clone( rhs.root );

}

return \*this;

}

template <class Comparable>

const Comparable & BinarySearchTree<Comparable>::

elementAt( BinaryNode<Comparable> \*t ) const

{

if( t == NULL )

return ITEM\_NOT\_FOUND;

else

return t->element;

}

template <class Comparable>

void BinarySearchTree<Comparable>::

insert( const Comparable & x, BinaryNode<Comparable> \* & t ) const

{

if( t == NULL )

t = new BinaryNode<Comparable>( x, NULL, NULL );

else if( x < t->element )

insert( x, t->left );

else if( t->element < x )

insert( x, t->right );

else

;

}

template <class Comparable>

void BinarySearchTree<Comparable>::

remove( const Comparable & x, BinaryNode<Comparable> \* & t ) const

{

if( t == NULL )

return; // Item not found; do nothing

if( x < t->element )

remove( x, t->left );

else if( t->element < x )

remove( x, t->right );

else if( t->left != NULL && t->right != NULL ) // Two children

{

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

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

}

else

{

BinaryNode<Comparable> \*oldNode = t;

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

delete oldNode;

}

}

template <class Comparable>

BinaryNode<Comparable> \*

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

{

if( t == NULL )

return NULL;

if( t->left == NULL )

return t;

return findMin( t->left );

}

template <class Comparable>

BinaryNode<Comparable> \*

BinarySearchTree<Comparable>::findMax( BinaryNode<Comparable> \*t ) const

{

if( t != NULL )

while( t->right != NULL )

t = t->right;

return t;

}

template <class Comparable>

BinaryNode<Comparable> \*

BinarySearchTree<Comparable>::

find( const Comparable & x, BinaryNode<Comparable> \*t ) const

{

if( t == NULL )

return NULL;

else if( x < t->element )

return find( x, t->left );

else if( t->element < x )

return find( x, t->right );

else

return t;

}

template <class Comparable>

void BinarySearchTree<Comparable>::

makeEmpty( BinaryNode<Comparable> \* & t ) const

{

if( t != NULL )

{

makeEmpty( t->left );

makeEmpty( t->right );

delete t;

}

t = NULL;

}

template <class Comparable>

void BinarySearchTree<Comparable>::printTree( BinaryNode<Comparable> \*t ) const

{

if( t != NULL )

{

printTree( t->left );

cout << t->element << " ";

printTree( t->right );

}

}

template <class Comparable>

void BinarySearchTree<Comparable>::printPre( BinaryNode<Comparable> \*t ) const

{

if( t != NULL )

{

cout << t->element << " ";

printPre( t->left );

printPre( t->right );

}

}

template <class Comparable>

void BinarySearchTree<Comparable>::printIn( BinaryNode<Comparable> \*t ) const

{

if( t != NULL )

{

printIn( t->left );

cout << t->element << " ";

printIn( t->right );

}

}

template <class Comparable>

void BinarySearchTree<Comparable>::printPost( BinaryNode<Comparable> \*t ) const

{

if( t != NULL )

{

printPost( t->left );

printPost( t->right );

cout << t->element << " ";

}

}

template <class Comparable>

BinaryNode<Comparable> \*

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

{

if( t == NULL )

return NULL;

else

return new BinaryNode<Comparable>( t->element, clone( t->left ), clone( t->right ) );

}

**Code ไฟล์ mainProgram.cpp**

#include <iostream>

#include "BinarySearchTree.h"

#include "dsexceptions.h"

#include "BinarySearchTree.cpp"

using namespace std;

void insert();

void print();

void remove();

void find();

void mainMenu();

const int ITEM\_NOT\_FOUND = -9999;

BinarySearchTree<int> a(ITEM\_NOT\_FOUND);

void main( )

{

system("cls");

int menu;

mainMenu();

do{

cout << "Select Menu : ";

cin >> menu;

}while(menu < 0 && menu >= 5);

switch(menu){

case 1 : insert(); break;

case 2 : print(); break;

case 3 : remove(); break;

case 4 : find(); break;

case 5 : exit(0);

}

system("pause");

}

void insert(){

int t,ne;

cout << "Enter Number Of Element : ";

cin>>ne;

for(int i=0;i<ne;i++){

cout << "Enter Element : ";

cin>>t;

a.insert(t);

}

system("pause");

main();

}

void print(){

system("cls");

cout << "-------------------------\n";

cout << "Print Tree\n";

cout << "\n-------------------------\n";

cout << "\n PreOrder : ";

a.printPre();

cout << "\n InOrder : ";

a.printIn();

cout << "\n PostOrder : ";

a.printPost();

cout << "\n";

system("pause");

main();

}

void remove(){

int t;

cout << "-------------------------\n";

cout << "Remove Tree\n";

cout << "\n-------------------------\n";

cout << "Enter Element : ";

cin >> t;

if(a.find(t) == t){

a.remove(t);

cout << "All Done!! to remove "<<t<<"\n";

}else{

cout << "Data Not Found "<<t<<" Please Try again\n";

}

system("pause");

main();

}

void find(){

system("cls");

int t;

cout << "-------------------------\n";

cout << "Find Tree\n";

cout << "\n-------------------------\n";

cout << "Min Element : " << a.findMin() << "\n";

cout << "Max Element : " << a.findMax() << "\n";

cout << "Enter Element : ";

cin >> t;

if(a.find(t) == t){

a.find(t);

cout << "Found "<<t<<" on Memory\n\n";

}else{

cout << "Data Not Found "<<t<<" Please Try again\n\n";

}

system("pause");

main();

}

void mainMenu(){

cout << "-------------------------\n";

cout << "1.Insert to Tree\n";

cout << "2.Print Tree\n";

cout << "3.Remove form Tree\n";

cout << "4.Find in Tree\n";

cout << "5.Close Program\n";

cout << "\n-------------------------\n";

}