#ifndef BST\_H\_INCLUDED

#define BST\_H\_INCLUDED

#include<iostream>

using namespace std;

#include "node.h"

template<class T>

class BST

{

private:

Node<T> \*head;

void insert(T, Node<T> \*);//add a new node where appropriate

void preOrder(Node<T>\*);

void inOrder(Node<T>\*);

void postOrder(Node<T>\*);

Node<T>\* deleteNode(Node<T>\*, T);

void makeEmpty(Node<T> \*);

public:

BST();//set head to null

~BST();// ???

void insert(T);//add a new node where appropriate

void preOrder();//print the tree in pre order

void inOrder();

void postOrder();

void deleteNode(T);

bool search(T);

};

#endif

using namespace std;

#include<iostream>

#include "bst.h"

template<class T>

BST<T> :: BST()

{

head = NULL;

}

template<class T>

BST<T> :: ~BST()

{

if(head != NULL)

{

makeEmpty(head->getLeftPtr());

delete head->getLeftPtr();

makeEmpty(head->getRightPtr());

delete head->getRightPtr();

delete head;

}

}

template<class T>

void BST<T> :: makeEmpty(Node<T> \*head)

{

if(head != NULL)

{

makeEmpty(head->getLeftPtr());

delete head->getLeftPtr();

makeEmpty(head->getRightPtr());

delete head->getRightPtr();

}

}

template<class T>

void BST<T> :: insert(T data)

{

if(head == NULL)

{

head = new Node<T>(data);

}

else

{

if(head -> getData() > data)

{

if(head->getLeftPtr() == NULL)

head->setLeftPtr(new Node<T>(data));

else

insert(data, head->getLeftPtr());

}

else

{

if(head->getRightPtr() == NULL)

head->setRightPtr(new Node<T>(data));

else

insert(data, head->getRightPtr());

}

}

}

template<class T>

void BST<T> :: insert(T data, Node<T> \*head)

{

if(head -> getData() > data)

{

if(head->getLeftPtr() == NULL)

head->setLeftPtr(new Node<T>(data));

else

insert(data, head->getLeftPtr());

}

else

{

if(head->getRightPtr() == NULL)

head->setRightPtr(new Node<T>(data));

else

insert(data, head->getRightPtr());

}

}

template<class T>

void BST<T> :: preOrder()

{

if(head != NULL)

{

cout << head->getData() << " ";

preOrder(head->getLeftPtr());

preOrder(head->getRightPtr());

}

}

template<class T>

void BST<T> :: preOrder(Node<T> \*head)

{

if(head != NULL)

{

cout << head->getData() << " ";

preOrder(head->getLeftPtr());

preOrder(head->getRightPtr());

}

}

template<class T>

void BST<T> :: inOrder()

{

if(head != NULL)

{

preOrder(head->getLeftPtr());

cout << head->getData() << " ";

preOrder(head->getRightPtr());

}

}

template<class T>

void BST<T> :: inOrder(Node<T> \*head)

{

if(head != NULL)

{

preOrder(head->getLeftPtr());

cout << head->getData() << " ";

preOrder(head->getRightPtr());

}

}

template<class T>

void BST<T> :: postOrder()

{

if(head != NULL)

{

preOrder(head->getLeftPtr());

preOrder(head->getRightPtr());

cout << head->getData() << " ";

}

}

template<class T>

void BST<T> :: postOrder(Node<T> \*head)

{

if(head != NULL)

{

preOrder(head->getLeftPtr());

preOrder(head->getRightPtr());

cout << head->getData() << " ";

}

}

template<class T>

void BST<T> :: deleteNode(T key)

{

if(head == NULL)

{

return;

}

if(head->getData() > key)

{

head->setLeftPtr(deleteNode(head->getLeftPtr(), key));

}

if(head->getData() < key)

{

head->setRightPtr(deleteNode(head->getRightPtr(), key));

}

if(head->getData() == key)

{

Node<T>\* temp = head;

if((head->getLeftPtr() == NULL) && (head->getRightPtr() == NULL))

{

head = NULL;

delete temp;

}

else if((head->getLeftPtr() == NULL) && (head->getRightPtr() != NULL))

{

head = head->getRightPtr();

delete temp;

}

else if((head->getLeftPtr() != NULL) && (head->getRightPtr() == NULL))

{

head = head->getLeftPtr();

delete temp;

}

else

{

Node<T>\* temp1 = temp;

temp = temp->getRightPtr();

while(temp->getLeftPtr() != NULL)

{

temp = temp->getLeftPtr();

}

head->setData(temp->getData());

temp1->setRightPtr(deleteNode(temp1->getRightPtr(), temp->getData()));

}

}

}

template<class T>

Node<T>\* BST<T> :: deleteNode(Node<T> \*head, T key)

{

if(head == NULL)

{

return NULL;

}

if(head->getData() > key)

{

head->setLeftPtr(deleteNode(head->getLeftPtr(), key));

return head;

}

if(head->getData() < key)

{

head->setRightPtr(deleteNode(head->getRightPtr(), key));

return head;

}

if(head->getData() == key)

{

Node<T>\* temp = head;

if((head->getLeftPtr() == NULL) && (head->getRightPtr() == NULL))

{

delete temp;

return NULL;

}

else if((head->getLeftPtr() == NULL) && (head->getRightPtr() != NULL))

{

Node<T> \*temp1 = head->getRightPtr();

delete temp;

return temp1;

}

else if((head->getLeftPtr() != NULL) && (head->getRightPtr() == NULL))

{

Node<T> \*temp1 = head->getLeftPtr();

delete temp;

return temp1;

}

else

{

Node<T>\* temp1 = temp;

temp = temp->getRightPtr();

while(temp->getLeftPtr() != NULL)

{

temp = temp->getLeftPtr();

}

head->setData(temp->getData());

temp1->setRightPtr(deleteNode(temp1->getRightPtr(), temp->getData()));

return head;

}

}

}

template<class T>

bool BST<T> :: search(T data)

{

Node<T> \*temp = head;

while(temp != NULL)

{

if(temp->getData() == data)

return true;

else

{

if(temp->getData() > data)

temp = temp->getLeftPtr();

else

temp = temp->getRightPtr();

}

}

return false;

}