19l-1316 lab 9 code:

ccp:

#include<iostream>

#include "Header.h"

using namespace std;

int main()

{

BinarySearchTree<int> \*BST=new BinarySearchTree<int>();

//following insertions should happen successfully as we are inserting unique values

BST->insert(12);

BST->insert(4);

BST->insert(9);

BST->insert(2);

BST->insert(14);

BST->insert(16);

BST->insert(13);

//this insertion should fail as 12 already exists in the Binary Search tree

BST->insert(12);

//prints data carried by the BST in sorted manner

BST->printSorted();

//the first search would be successful and second would fail

BNode<int>\*n =BST->search(12);

if(n)

{

cout<<"Value exists"<<endl;

}

else

{

cout<<"Value does not exist"<<endl;

}

BNode<int>\*n2 =BST->search(23);

if(n2)

{

cout<<"Value exists"<<endl;

}

else

{

cout<<"Value does not exist"<<endl;

}

system("pause");

return 0;

}

header:

#ifndef BINARY\_SEARCH\_TREE\_H

#define BINARY\_SEARCH\_TREE\_H

#include<iostream>

using namespace std;

template<class DT>

class BNode

{

public:

BNode(){

leftchild = NULL;

rightchild = NULL;

}

void setLeftChild(BNode<DT>\* n){

leftchild = n;

}

BNode<DT>\* getLeftChild(){

return leftchild;

}

void setRightChild(BNode<DT>\* n){

rightchild = n;

}

BNode<DT>\* getRightChild(){

return rightchild;

}

void setData(DT pdate){

data = pdate;

}

DT getData(){

return data;

}

private:

DT data;

BNode\* leftchild;

BNode\* rightchild;

};

template<class DType>

class BinarySearchTree

{

public:

//part1: constructor

BinarySearchTree(){

root=NULL;

}

//part2: Create and insert a BNode carrying data

//in the binary search tree. It return true if

//insertion takes place successfully and false otherwise

bool insert(const DType data){

BNode<DType>\*a=NULL,\*y=root;

while(y)

{

a=y;

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

{

return false;

}

if(data>y->getData())

{

y=y->getRightChild();

}

else

{

y=y->getLeftChild();

}

}

y=new BNode<DType>();

y->setLeftChild(NULL);

y->setRightChild(NULL);

y->setData(data);

if(!root)

{

root=y;

}

else if(data>a->getData())

{

a->setRightChild(y);

}

else

{

a->setLeftChild(y);

}

}

//part3: Search for data in the binary search tree

// and return the pointer of the node carrying data

//return null/0 if data doesn’t exist

BNode<DType> \* search(const DType data){

return searchR(root,data);

}

//part4: prints all the data present in the tree

//sorted in ascending order

void printSorted(){

inorderR(root);

}

//part5: delete the BNode carrying data from the

//binary search tree. It return true if

//deletion takes place successfully and false otherwise

//bool delete(const DType data);

//part6: destructor, delete all nodes

~BinarySearchTree(){

}

private:

BNode<DType> \* root;

BNode<DType>\*searchR(BNode<DType>\* n, DType x){

BNode<DType>\*temp = n;

if (n == NULL)

{

return NULL;

}

else if (x<temp->getData())

{

return searchR(n->getLeftChild(),x);

}

else if (x<temp->getData())

{

return searchR(n->getRightChild(), x);

}

else if (x == temp->getData())

{

return n;

}

else

{

return NULL;

}

}

void inorderR(BNode<DType>\*n){

if (n)

{

inorderR(n->getLeftChild());

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

inorderR(n->getRightChild());

}

}

};

#endif