// bai3\_BinarySearchTree\_songuyen.cpp : Defines the entry point for the console application.

//

#include "stdio.h"

#include "conio.h"

#include "stdlib.h"

#include "math.h"

typedef struct cautruccay

{

int info;

struct cautruccay \*left,\*right;

}TNODE;

typedef TNODE \*TREE;

TNODE \*taophantu(int x);

int them(TREE &T, int x);

void nhapcay(TREE &T);

void xuatNLR(TREE T);

void xuatNRL(TREE T);

void xuatLNR(TREE T);

void xuatRNL(TREE T);

void docao(TREE T, int &max, int i);

void InmucM(TREE T, int m, int i);

void DemNodeM(TREE T, int m, int &dem, int i);

void DemNode(TREE T, int &dem);

void timqthayp(TREE &p, TREE &q);

int DemnhanhTrai(TREE T, int x);

void DemNodebac0(TREE T, int &dem);

void DemNodebac1(TREE T, int &dem);

void DemNodebac2(TREE T, int &dem);

TNODE \*timnode(TREE T, int x);

int xoanode(TREE &T, int x);

void xoaSoNgto(TREE &T);

void xoacay(TREE &T);

void menu()

{

printf("\n0: Thoat");

printf("\n1: Nhap cay");

printf("\n2: Duyet cay tang dan");

printf("\n3: Duyet cay giam dan");

printf("\n4: Duyet Node - left - right");

printf("\n5: Duyet Node - right - left");

printf("\n6: Tinh do cao cay");

printf("\n7: Duyet cay theo muc");

printf("\n8: Dem node muc thu m");

printf("\n9: Dem so node o cay con trai");

printf("\n10: Dem node bac 0");

printf("\n11: Dem node bac 1");

printf("\n12: Dem node bac 2");

printf("\n13: Tim node");

printf("\n14: Xoa 1 node");

printf("\n15: Xoa tat ca so nguyen to tren cay");

printf("\n");

printf("\nChon thao tac: ");

}

void main()

{

char ch;

int x,m,kq=0,h=0;

TREE T;

T=NULL;

do

{

menu();

scanf("%d",&ch);

switch(ch)

{

case 0:

return;

case 1:

T=NULL;

nhapcay(T);

break;

case 2:

xuatLNR(T);

break;

case 3:

xuatRNL(T);

break;

case 4:

xuatNLR(T);

break;

case 5:

xuatNRL(T);

break;

case 6:

h=0;

docao(T,h,0);

h=h+1;

printf("\nChieu cao cay la: %d",h);

break;

case 7:

printf("\nNhap muc: ");

scanf("%d",&m);

InmucM(T,m,0);

break;

case 8:

kq=0;

printf("\nNhap muc: ");

scanf("%d",&m);

DemNodeM(T,m,kq,0);

printf("\nMuc %d co %d Node",m,kq);

break;

case 9:

printf("\nNhap x:");

scanf("%d",&x);

if(timnode(T,x)==NULL)

printf("\nKhong co Node %d",x);

else

printf("\nCay con trai co %d Node",DemnhanhTrai(T,x));

break;

case 10:

kq=0;

DemNodebac0(T,kq);

printf("\nCo %d node bac 0",kq);

break;

case 11:

kq=0;

DemNodebac1(T,kq);

printf("\nCo %d node bac 1",kq);

break;

case 12:

kq=0;

DemNodebac2(T,kq);

printf("\nCo %d node bac 2",kq);

break;

case 13:

printf("\nNhap phan tu x: ");

scanf("%d",&x);

if(timnode(T,x))

printf("\nTim thay %d",x);

else

printf("\nKhong co phan tu %d",x);

break;

case 14:

printf("\nNhap node can xoa: ");

scanf("%d",&x);

xoanode(T,x);

break;

case 15:

xoaSoNgto(T);

break;

default:

printf("\nSai chuc nang!");

break;

}

}

while(1);

}

TNODE \*taophantu(int x)

{

TNODE \*p;

p=new TNODE;

if(p==NULL)

{

printf("\nKhong du bo nho!");

exit(1);

}

p->info=x;

p->left=p->right=NULL;

return p;

}

int them(TREE &T, int x)

{

if(T==NULL)

{

T=new TNODE;

if(T==NULL)

return -1;

T->info=x;

T->left=T->right=NULL;

return 1;

}

if(T->info>x)

return them(T->left,x);

if(T->info<x)

return them(T->right,x);

return 0;

}

void nhapcay(TREE &T)

{

int kq, x, i=0;

T=NULL;

printf("\nNhap cac phan tu khong am:");

do

{

printf("\nNhap phan tu thu %d: ",i+1);

scanf("%d",&x);

if(x>=0)

{

kq=them(T,x);

if(kq==0)

printf("\nNhap gia tri khac!");

if(kq==-1)

{

printf("\nKhong du bo nho!");

exit(1);

}

}

i++;

}

while(x>=0);

}

void xuatLNR(TREE T)

{

if(T!=NULL)

{

xuatLNR(T->left);

printf("%3d",T->info);

xuatLNR(T->right);

}

}

void xuatRNL(TREE T)

{

if(T!=NULL)

{

xuatRNL(T->right);

printf("%3d",T->info);

xuatRNL(T->left);

}

}

void xuatNLR(TREE T)

{

if(T!=NULL)

{

printf("%3d",T->info);

xuatNLR(T->left);

xuatNLR(T->right);

}

}

void xuatNRL(TREE T)

{

if(T!=NULL)

{

printf("%3d",T->info);

xuatNRL(T->right);

xuatNRL(T->left);

}

}

void docao(TREE T, int &max, int i)

{

if(T)

{

if(i>max)

max=i;

docao(T->left,max, i+1);

docao(T->right,max, i+1);

}

}

void InmucM(TREE T, int m, int i)

{

if(T)

{

if(i==m)

printf("%3d",T->info);

InmucM(T->left,m,i+1);

InmucM(T->right,m,i+1);

}

}

void DemNodeM(TREE T, int m, int &dem, int i)

{

if(T)

{

if(i==m)

dem++;

DemNodeM(T->left,m,dem,i+1);

DemNodeM(T->right,m,dem,i+1);

}

}

TNODE \*timnode(TREE T, int x)

{

if(T)

{

TNODE \*p;

p=T;

if(p->info==x)

return p;

else

{

if(p->info>x)

return timnode(T->left,x);

else

return timnode(T->right,x);

}

}

return NULL;

}

void DemNode(TREE T, int &dem)

{

if(T)

{

dem++;

DemNode(T->left,dem);

DemNode(T->right,dem);

}

}

int DemnhanhTrai(TREE T, int x)

{

int dem=0;

TNODE \*p;

if(T)

{

p=timnode(T,x);

DemNode(p->left,dem);

}

return dem;

}

void DemNodebac0(TREE T, int &dem)

{

if(T)

{

if(T->left==NULL && T->right==NULL)

dem++;

DemNodebac0(T->left,dem);

DemNodebac0(T->right,dem);

}

}

void DemNodebac1(TREE T, int &dem)

{

if(T)

{

if(T->left)

if(T->right==NULL)

dem++;

else

if(T->right)

dem++;

DemNodebac1(T->left,dem);

DemNodebac2(T->right,dem);

}

}

void DemNodebac2(TREE T, int &dem)

{

if(T)

{

if(T->left && T->right)

dem++;

DemNodebac2(T->left,dem);

DemNodebac2(T->right,dem);

}

}

void timqthayp(TREE &p, TREE &q)

{

if(q->left)

timqthayp(p,q->left);

else

{

p->info=q->info;

p=q;

q=q->right;

}

}

int xoanode(TREE &T, int x)

{

TNODE \*p;

if(T==NULL)

return 0;

if(x<T->info)

return xoanode(T->left,x);

else

if(x>T->info)

return xoanode(T->right,x);

else

{

p=T;

if(T->left==NULL)

T=T->right;

else

if(T->right==NULL)

T=T->left;

else

timqthayp(p,T->right);

delete p;

return 1;

}

}

int soNgto(int x)

{

if(x<2)

return 0;

for(int i=2;i<=sqrt(x);i++)

if(x%i==0)

return 0;

return 1;

}

void xoaSoNgto(TREE &T)

{

int x;

if(T!=NULL)

{

xoaSoNgto(T->left);

xoaSoNgto(T->right);

if(soNgto(T->info))

{

x=T->info;

xoanode(T,x);

}

}

}

void xoacay(TREE &T)

{

if(T)

{

xoacay(T->left);

xoacay(T->right);

delete T;

}

}

//dem nut la

// in cay theo tung muc

// dem cai nut chi co cay con trai

#include "iostream"

#include "stdio.h"

#include "conio.h"

#include "math.h"

using namespace std;

typedef struct tagTNode

{

int key;

struct tagTNode \*pLeft;

struct tagTNode \*pRight;

}TNode;

typedef TNode \*Tree;

void CreateTree (Tree &T)

{

T=NULL;

}

TNode \*CreateTNode(int x)

{

TNode \*p;

p=new TNode;

if(p==NULL)exit(1);

else

{

p->key=x;

p->pLeft=NULL;

p->pRight=NULL;

}

return p;

}

int InsertNode(Tree &T, int x)

{

if(T)

{

if(T->key==x)return 0;

if(T->key<x) return InsertNode(T->pRight, x);

else if(T->key>x) return InsertNode(T->pLeft,x);

}

T=new TNode;

if(T==NULL) return -1;

T->key=x;

T->pLeft=T->pRight=NULL;

return 1;

}

void nhap(Tree &T)

{

int x;

do

{

cout<<"Nhap vao du lieu, nhap 0 de thoat: ";

cin>>x;

if(x==0) break;

InsertNode(T,x);

}while(x!=0);

}

void LNR(Tree T)

{

if(T!=NULL)

{

LNR(T->pLeft);

cout<<T->key<<" ";

LNR(T->pRight);

}

}

void NLR(Tree T)

{

if(T!=NULL)

{

cout<<T->key<<" ";

NLR(T->pLeft);

NLR(T->pRight);

}

}

void RNL(Tree T)

{

if(T!=NULL)

{

RNL(T->pRight);

cout<<T->key<<" ";

RNL(T->pLeft);

}

}

TNode \*SearchNode(Tree T, int x)

{

if(T!=NULL)

{

if(T->key==x)return T;

else if(T->key>x) return SearchNode(T->pLeft,x);

else return SearchNode(T->pRight,x);

}return NULL;

}

void demla(Tree T, int &la)

{

if(T)

{

if((T->pLeft==NULL)&&(T->pRight==NULL))la++;

demla(T->pLeft,la);

demla(T->pRight,la);

}

}

void demtrai(Tree T, int &left)

{

if(T)

{

if((T->pLeft!=NULL)&&(T->pRight==NULL))left++;

demtrai(T->pLeft,left);

demtrai(T->pRight,left);

}

}

int nguyento(int n)

{

if(n==1) return 0;

float t= sqrt(float(n));

for(int i=2;i<=t;i++) if(n%i==0)return 0;

return 1;

}

void demsnt(Tree T, int &snt)

{

if(T)

{

if(nguyento(T->key)==1)snt++;

demsnt(T->pLeft,snt);

demsnt(T->pRight,snt);

}

}

void docao(Tree T, int h, int &max)

{

if(T)

{

if(h>max)max=h;

docao(T->pLeft,h+1,max);

docao(T->pRight,h+1,max);

}

}

void InMuc(Tree T, int h, int muc)

{

if(T)

{

if(muc==h) cout<<T->key<<" ";

else

{

InMuc(T->pLeft,h+1,muc);

InMuc(T->pRight,h+1,muc);

}

}

}

void thaythe(Tree &p, Tree&T)

{

if(T->pLeft!=NULL) thaythe(p,T->pLeft);

else

{

p->key=T->key;

p=T;

T=T->pRight;

}

}

void deleteNode(Tree &T, int x)

{

if(T!=NULL)

{

if(T->key<x)deleteNode(T->pRight,x);

else

{

if(T->key>x)deleteNode(T->pLeft,x);

else

{

TNode \*p;

p=T;

if(T->pLeft==NULL)T=T->pRight;

else

{

if(T->pRight==NULL)T=T->pLeft;

else thaythe(p,T->pRight);

}

delete p;

}

}

}

else cout<<"Khong tim thay node x de xoa";

}

void xoasnt(Tree &T)

{

if(T)

{

xoasnt(T->pLeft);

xoasnt(T->pRight);

if(nguyento(T->key)==1)deleteNode(T,T->key);

}

}

void docaotrai(Tree T, int h, int &max)

{

if(T)

{

if(h>max)max=h;

docaotrai(T->pLeft,h+1,max);

}

}

void docaophai(Tree T, int h, int &max)

{

if(T)

{

if(h>max)max=h;

docaophai(T->pRight,h+1,max);

}

}

void menu()

{

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

cout<<"\n0: Exit";

cout<<"\n1: Duyet NLR";

cout<<"\n2: Duyet LNR";

cout<<"\n3: Duyet RNL";

cout<<"\n4: Tim phan tu co khoa x tren cay";

cout<<"\n5: Tinh do cao cua cay, cua cay con trai, cay con phai";

cout<<"\n6: Dem cac nut la";

cout<<"\n7: In cay theo tung muc";

cout<<"\n8: Dem cac so nguyen to trong cay";

cout<<"\n9: Xoa not x trong cay";

cout<<"\n10: Dem so node chi co cay con trai";

cout<<"\n11: Xoa so nguyen to trong cay";

}

void main()

{

Tree T;

int x;

CreateTree(T);

nhap(T);

do

{

int c;

menu();

cout<<"\n\nNhap lua chon: ";

cin>>c;

switch(c)

{

case 0: exit(1);

case 1: {NLR(T);break;}

case 2: {LNR(T);break;}

case 3: {RNL(T);break;}

case 4:

{

cout<<"Nhap x can tim: ";

cin>>x;

if(SearchNode(T,x)!=NULL)cout<<"Co phan tu x trong cay";

else cout<<"Khong ton tai phan tu x trong cay";

break;

}

case 5:

{

int h=0,max=0,left=0,right=0;

docao(T,h,max);

docaotrai(T,h,left);

docaophai(T,h,right);

cout<<"Do cao cua cay la: "<<max;

cout<<"\nDo cao cua cay con trai la: "<<left;

cout<<"\nDo cao cua cay con phai la: "<<right;

break;

}

case 6:

{

int la=0;

demla(T,la);

cout<<"So nut la trong cay la: "<<la;break;

}

case 7:

{

int h=0,muc=0;

docao(T,h,muc);

for(int i=0;i<muc;i++)

{InMuc(T,h,i);cout<<"\n";}

break;

}

case 8:

{

int snt=0;

demsnt(T,snt);

cout<<"So node la so nguyen to trong cay la: "<<snt;

break;

}

case 9:

{

int x;

cout<<"Nhap gia tri node x can xoa: ";

cin>>x;

deleteNode(T,x);

LNR(T);

break;

}

case 10:

{

int left=0;

demtrai(T,left);

cout<<"So nut la chi co cay con trai trong cay la: "<<left;break;

}

case 11:

{

xoasnt(T);

LNR(T);

}

}

}while(1);

getch();

}

#include<iostream>

using namespace std;

class TreeNode{

private:

int data;

TreeNode \*left;

TreeNode \*right;

public:

TreeNode(){

data=-1;

left=right=NULL;

}

TreeNode(int n){

data=n;

left=right=NULL;

}

int getData(){

return data;

}

void setData(int n){

data=n;

}

void setLeft(TreeNode \*left){

this->left=left;

}

void setRight(TreeNode \*right){

this->right=right;

}

TreeNode \*getLeft(){

return left;

}

TreeNode \*getRight(){

return right;

}

void print(){

cout<<data<<"";

}

};

class Stack{

private:

TreeNode \*nodes;

public:

int size;

int top;

Stack(int size){

top=0;

this->size=size;

nodes = new Treenode[size];

}

TreeNode pop(){

if (top == 0){

cout<<"Stack is empty"<<endl;

return -1;

}

top--;

return nodes[top];

}

TreeNode push(TreeNode \*node){

if (top==size){

cout<<"Stack is full"<<endl;

return 0;

nodes[top]= \*node;

top++;

}

}

};

class BST{

private:

TreeNode \* root;

public:

BST(){

root=NULL;

}

void preorder(){

preorder(root);

cout<<endl;

}

void preorder(TreeNode \*root){

if(root !=NULL){

root->print();

preorder(root->getLeft());

preorder(root->getRight());

}

}

void insert(int n){

TreeNode \* newNode = new TreeNode(n);

if(root == NULL)

{

root = newNode;

}

else{

TreeNode \*temp = root;

TreeNode \*prev = NULL;

while(temp != NULL){

prev=temp;

if (n < temp->getData()){

temp=temp->getLeft();

}

else{

temp=temp->getRight();

}

if (n > prev->getData()){

prev->setRight( newNode );

}

else{

prev->setLeft( newNode );

}

}

}

}

void print(){

Stack \*s= new Stack(50);

s->push(root);

while( s->size > 0 ){

TreeNode t = s->pop();

cout<<t.getData()<<"";

if(t.getLeft() != NULL){

s->push(t.getLeft());

}

if(t.getRight() != NULL){

s->push(t.getRight());

}

}

cout<<endl;

}

};

int main(){

int n;

BST \*bst = new BST();

do{

cin>>n;

if( n == -1)

break;

bst->insert(n);

}while( true );

bst->print();

bst->preorder();

return 0;

}

#include "stdafx.h"

#include <cstdlib>

#include <iostream>

#define SIZE 10

using namespace std;

struct Node {

int value;

Node \*left;

Node \*right;

Node \*parent;

};

struct BST {

Node \*root;

};

void insert(int value, BST \*tree) {

Node \*x = tree->root;

Node \*y = NULL;

Node \*z = (Node \*) malloc(sizeof(Node));

z->left = NULL;

z->right = NULL;

z->value = value;

// Add your code here

while (x!=NULL){

y=x;

if (z->value < x->value)

x= x->left;

else x = x->right;

}

z->parent=y;

if (y==NULL)

tree->root=z;

else if (z->value <y->value)

y->left =z;

else y->right =z;

}

Node \*search(int key, Node \*n) {

if (n== NULL || key == n->value)

return n;

if (key < n->value)

search(key, n->left);

else

search(key, n->right);

}

Node \*min(Node \*n) {

if (n == NULL || n->left == NULL)

return n;

else

return min(n->left);

}

Node \*max(Node \*n) {

if (n == NULL || n->right == NULL)

return n;

else

return max(n->right);

}

Node \*successor(int value, Node \*n) {

Node \*y = NULL;

Node \*x = search(value, n);

if (x == NULL)

return NULL;

if (x->right != NULL)

return min(x->right);

y = x->parent;

while (y != NULL && x == y->right) {

x = y;

y = y->parent;

}

return y;

}

Node \*predecessor(int value, Node \*n) {

Node \*x = search(value, n);

Node \*y = NULL;

if (x == NULL)

return NULL;

if (x->left != NULL)

return max(x->left);

y = x->parent;

while (y != NULL && x == y->left) {

x = y;

y = y->parent;

}

return y;

}

Node \*remove(int value, BST \*tree) {

Node \*z = search(value, tree->root);

Node \*y = NULL, \*x = NULL;

if (z == NULL) return NULL;

if (z->left == NULL || z->right == NULL)

y = z;

else

y = successor(value, z);

if (y->left != NULL)

x = y->left;

else

x = y->right;

if (x != NULL)

x->parent = y->parent;

if (y->parent == NULL)

tree->root = x;

else if (y == y->parent->left)

y->parent->left = x;

else

y->parent->right = x;

if (y != z) {

int tmp = z->value;

z->value = y->value;

y->value = tmp;

}

return y;

}

// ascending sort function

void sortAsc(Node \*node) {

//Add your code here

//inorder

if (node->left!=NULL)

sortAsc(node->left);

cout<<node->value<<" ";

if (node->right!=NULL)

sortAsc(node->right);

}

// descending sort function

void sortDes(Node \*node) {

// Add your code here

//inorder

if (node->right!=NULL)

sortDes(node->right);

cout<<node->value<<" ";

if (node->left!=NULL)

sortDes(node->left);

}

void clear(BST \*tree) {

Node \*n = NULL;

while (tree->root != NULL) {

n = remove(tree->root->value, tree);

free(n);

}

}

int main() {

int A[] = {3, 5, 10, 4, 8, 9, 1, 4, 7, 6};

Node \*node = NULL;

BST \*tree = (BST \*) malloc(sizeof(BST));

tree->root = NULL;

// build BST tree

cout << "Input data:\n\t";

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

cout << A[i] << " "; // by the way, print it to the console

insert(A[i], tree); // You need to complete TASK 1, so that it can work

}

// sort values in ascending order

cout << "\n\nAscending order:\n\t";

sortAsc(tree->root); // You need to complete TASK 2. Otherwise you see nothing in the console

// sort values in descending order

cout << "\n\nDescending order:\n\t";

sortDes(tree->root); // TASK 2 also!

// Find minimum value

if (tree->root != NULL)

cout << "\n\nMin: " << min(tree->root)->value;

// Find maximum value

if (tree->root != NULL)

cout << "\n\nMax: " << max(tree->root)->value;

// delete 4

cout << "\n\nDelete 4 and add 2";

//free(remove(4, tree)); // You need to complete TASK 3, so that remove(int, BST \*) function works properly

// we also need to release the resource!!!

// insert 2

insert(2, tree); // It belongs to TASK 1 too.

cout << "\n\nAscending order:\n\t";

sortAsc(tree->root); // TASK 2!!

// Find the successor of 5, -1 means no successor

node = search(5, tree->root);

cout << "\n\nSearch of 5 is: " << (node != NULL?node->value:-1);

// Find the successor of 5, -1 means no successor

node = successor(5, tree->root);

cout << "\n\nSuccessor of 5 is: " << (node != NULL?node->value:-1);

// Find the predecessor of 5. -1 means no predecessor

node = predecessor(5, tree->root);

cout << "\n\nPredecessor of 5 is: " << (node != NULL?node->value:-1);

cout << "\n\n";

// clear all elements

clear(tree); // delete all nodes and release resource

free(tree); // delte the tree too

system("Pause");

}

#include <iostream.h>

#include <process.h> //for exit(1)

#include <conio.h>

struct node{

int data;

struct node \*left;

struct node \*right;

};

class BST{

public:

node \*tree;

BST(){

tree=NULL;

}

void createTree(node \*\*,int item); //For Building Treevoid preOrder(node \*); //For Tree Traversalvoid inOrder(node \*);

void postOrder(node \*);

void determineHeight(node \*,int \*);

int totalNodes(node \*);

int internalNodes(node \*); //no. of non-leaf nodesint externalNodes(node \*); //no. of leaf nodes.void removeTree(node \*\*); //Remove tree from memory.

node \*\*searchElement(node \*\*,int);

void findSmallestNode(node \*);

void findLargestNode(node \*);

void deleteNode(int);

};

//it is used for inseting an single element in//a tree, but if calls more than once will create tree.void BST :: createTree(node \*\*tree,int item){

if(\*tree == NULL){

\*tree = new node;

(\*tree)->data = item;

(\*tree)->left = NULL;

(\*tree)->right = NULL;

}

else{

if( (\*tree)->data > item)

createTree( &((\*tree)->left),item);

else

createTree( &((\*tree)->right),item);

}

}

void BST :: preOrder(node \*tree){

if( tree!=NULL){

cout<<" "<< tree->data;

preOrder(tree->left);

preOrder(tree->right);

}

}

void BST :: inOrder(node \*tree){

if( tree!=NULL){

inOrder( tree->left);

cout<<" "<< tree->data;

inOrder(tree->right);

}

}

void BST :: postOrder(node \*tree){

if( tree!=NULL){

postOrder( tree->left);

postOrder( tree->right);

cout<<" "<<tree->data;

}

}

void BST :: determineHeight(node \*tree, int \*height){

int left\_height, right\_height;

if( tree == NULL)

\*height = 0;

else{

determineHeight(tree->left, &left\_height);

determineHeight(tree->right, &right\_height);

if( left\_height > right\_height)

\*height = left\_height + 1;

else

\*height = right\_height + 1;

}

}

int BST :: totalNodes(node \*tree){

if( tree == NULL)

return 0;

elsereturn( totalNodes(tree->left) + totalNodes(tree->right) + 1 );

}

int BST :: internalNodes(node \*tree){

if( (tree==NULL) || (tree->left==NULL && tree->right==NULL))

return 0;

elsereturn( internalNodes(tree->left) + internalNodes(tree->right) + 1 );

}

int BST :: externalNodes(node \*tree){

if( tree==NULL )

return 0;

elseif( tree->left==NULL && tree->right==NULL)

return 1;

elsereturn( externalNodes(tree->left) + externalNodes(tree->right));

}

void BST :: removeTree(node \*\*tree){

if( (\*tree) != NULL){

removeTree( &(\*tree)->left );

removeTree( &(\*tree)->right );

delete( \*tree );

}

}

node \*\* BST :: searchElement(node \*\*tree, int item){

if( ((\*tree)->data == item) || ( (\*tree) == NULL) )

return tree;

elseif( item < (\*tree)->data)

return searchElement( &(\*tree)->left, item);

elsereturn searchElement( &(\*tree)->right, item);

}

void BST :: findSmallestNode(node \*tree){

if( tree==NULL || tree->left==NULL)

cout<< tree->data;

else

findSmallestNode( tree->left);

}

//Finding In\_order Successor of given node..//for Delete Algo.

node \* find\_Insucc(node \*curr)

{

node \*succ=curr->right; //Move to the right sub-tree.if(succ!=NULL){

while(succ->left!=NULL) //If right sub-tree is not empty.

succ=succ->left; //move to the left-most end.

}

return(succ);

}

void BST :: findLargestNode(node \*tree){

if( tree==NULL || tree->right==NULL)

cout<<tree->data;

else

findLargestNode(tree->right);

}

void BST :: deleteNode(int item){

node \*curr=tree,\*succ,\*pred;

int flag=0,delcase;

//step to find location of nodewhile(curr!=NULL && flag!=1)

{

if(item < curr->data){

pred = curr;

curr = curr->left;

}

elseif(item > curr->data){

pred = curr;

curr = curr->right;

}

else{ //curr->data = item

flag=1;

}

}

if(flag==0){

cout<<"\nItem does not exist : No deletion\n";

getch();

goto end;

}

//Decide the case of deletionif(curr->left==NULL && curr->right==NULL)

delcase=1; //Node has no childelseif(curr->left!=NULL && curr->right!=NULL)

delcase=3; //Node contains both the childelse

delcase=2; //Node contains only one child//Deletion Case 1if(delcase==1){

if(pred->left == curr) //if the node is a left child

pred->left=NULL; //set pointer of its parentelse

pred->right=NULL;

delete(curr); //Return deleted node to the memory bank.

}

//Deletion Case 2if(delcase==2){

if(pred->left==curr){ //if the node is a left childif(curr->left==NULL)

pred->left=curr->right;

else

pred->left=curr->left;

}

else{ //pred->right=currif(curr->left==NULL)

pred->right=curr->right;

else

pred->right=curr->left;

}

delete(curr);

}

//Deletion case 3if(delcase==3){

succ = find\_Insucc(curr); //Find the in\_order successor//of the node.int item1 = succ->data;

deleteNode(item1); //Delete the inorder successor

curr->data = item1; //Replace the data with the data of//in order successor.

}

end:

}

void main(){

BST obj;

int choice;

int height=0,total=0,n,item;

node \*\*tmp;

while(1){

clrscr();

cout<<"\*\*\*\*\*BINARY SEARCH TREE OPERATIONS\*\*\*\*\*\n\n";

cout<<"--Binary Tree and Binary Search Tree common operations--\n";

cout<<"1) Create Tree\n";

cout<<"2) Traversal\n";

cout<<"3) Height of Tree\n";

cout<<"4) Total Nodes\n";

cout<<"5) Internal Nodes \n";

cout<<"6) External Nodes \n";

cout<<"7) Remove Tree\n";

cout<<"\n--Only Binary Search Tree Operations--\n";

cout<<"8) Insert Node\n";

cout<<"9) Search Node\n";

cout<<"10) Find Smallest Node\n";

cout<<"11) Find Largest Node\n";

cout<<"12) Delete Node\n";

cout<<"13) Exit\n";

cout<<"Enter your choice : ";

cin>>choice;

switch(choice){

case 1 : //Create Tree

cout<<"\n\n--Creating Tree--";

cout<<"\nHow many nodes u want to enter : ";

cin>>n;

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

cout<<"Enter value : ";

cin>>item;

obj.createTree(&obj.tree,item);

}

break;

case 2 : //All Traversals

cout<<"\n\nInorder Traversal : ";

obj.inOrder(obj.tree);

cout<<"\n\nPre-order Traversal : ";

obj.preOrder(obj.tree);

cout<<"\n\nPost-order Traversal : ";

obj.postOrder(obj.tree);

getch();

break;

case 3 : //Determining Height of Tree

obj.determineHeight(obj.tree,&height);

cout<<"\n\nHeight of Tree : "<<height;

getch();

break;

case 4 : //Total nodes in a tree

total=obj.totalNodes(obj.tree);

cout<<"\n\nTotal Nodes : "<<total;

getch();

break;

case 5 : //Internal nodes in a tree

total=obj.internalNodes(obj.tree);

cout<<"\n\nInternal Nodes : "<<total;

getch();

break;

case 6 : //External nodes in a tree

total=obj.externalNodes(obj.tree);

cout<<"\n\nExternal Nodes : "<<total;

getch();

break;

case 7 : //Remove Tree from memory

obj.removeTree(&obj.tree);

cout<<"\n\nTree is removed from Memory";

getch();

break;

case 8 : //Inserting a node in a tree

cout<<"\n\n--Inserting Node in a tree--\n";

cout<<"Enter value : ";

cin>>item;

obj.createTree(&obj.tree,item);

cout<<"\nItem is inserted\n";

getch();

break;

case 9 : //Search element

cout<<"\n\n--Search Element--\n";

cout<<"Enter item to searched : ";

cin>>item;

&(\*tmp) = obj.searchElement(&obj.tree,item);

if( (\*tmp) == NULL)

cout<<"\nSearch Element Not Found";

else

cout<<"\nSearch Element was Found";

getch();

break;

case 10 : //Find Smallest Node

cout<<"\n\nSmallest Node is : ";

obj.findSmallestNode(obj.tree);

getch();

break;

case 11 : //Find Largest Node

cout<<"\n\nLargest Node is : ";

obj.findLargestNode(obj.tree);

getch();

break;

case 12 : //Deleting a node from a tree

cout<<"\n\n--Deleting a Node from a tree--\n";

cout<<"Enter value : ";

cin>>item;

obj.deleteNode(item);

break;

case 13 : exit(1);

}//end of switch

}

}

Data Structure: Implementing Tree Sort in C++

#include<iostream>

using namespace std;

struct tree{

int info;

tree \*Left, \*Right;

};

tree \*root;

class TreeSort{

public:

int no\_of\_elements;

int elements[10];

public:

void getarray();

void sortit();

void insert1(int);

tree \*insert2(tree \*, tree \*);

void display(tree \*);

};

void TreeSort::getarray(){

cout<<"How many elements? ";

cin>>no\_of\_elements;

cout<<"Insert array of element to sort: ";

for(int i=0;i<no\_of\_elements;i++){

cin>>elements[i];

}

}

void TreeSort::sortit(){

for(int i = 0; i < no\_of\_elements; i++){

insert1(elements[i]);

}

}

tree\* TreeSort::insert2(tree \*temp,tree \*newnode){

if(temp==NULL){

temp=newnode;

}

else if(temp->info < newnode->info){

insert2(temp->Right,newnode);

if(temp->Right==NULL)

temp->Right=newnode;

}

else{

insert2(temp->Left,newnode);

if(temp->Left==NULL)

temp->Left=newnode;

}

return temp;

}

void TreeSort::insert1(int n){

tree \*temp=root,\*newnode;

newnode=new tree;

newnode->Left=NULL;

newnode->Right=NULL;

newnode->info=n;

root=insert2(temp,newnode);

}

/\* Inorder traversal \*/

void TreeSort::display(tree \*t = root){

if(root==NULL){

cout<<"Nothing to display";

}else

if(t!=NULL){

display(t->Left);

cout<<t->info<<" ";

display(t->Right);

}

}

int main(){

TreeSort TS;

TS.getarray();

TS.sortit();

TS.display();

return 0;

}