**TREE TEST**

//cac phep toan co ban tren cay BST - su dung tap test tren ht ELSE de thu

#include<stdio.h>

#include<malloc.h>

struct Node{

int Key;

struct Node \*Left, \*Right;

};

typedef struct Node\* Tree;

//cau 1

Tree initTree()

{

Tree T=NULL;

return T;

}

//cau 2

int isEmpty(struct Node \*T)

{

if (T==NULL)

return 1;

else return 0;

}

//cau 3

void insertNode(int x,Tree \*T)

{

if(\*T==NULL)

{

struct Node\* Newnode=(struct Node\*)realloc(0,sizeof(struct Node));

Newnode->Key=x;

Newnode->Left=NULL;

Newnode->Right=NULL;

\*T=Newnode;

}

else

{

if((\*T)->Key<x)

{

insertNode(x,&(\*T)->Right);

}

else if((\*T)->Key>x)

{

insertNode(x,&(\*T)->Left);

}

}

}

//cau 4

void preOrder(Tree T)

{

if(T != NULL)

{

printf("%d ",T->Key);

preOrder(T->Left);

preOrder(T->Right);

}

}

//cau 5

void inOrder(Tree T)

{

if(T != NULL)

{

inOrder(T->Left);

printf("%d ",T->Key);

inOrder(T->Right);

}

}

//cau 6

void posOrder(Tree T)

{

if(T != NULL)

{

posOrder(T->Left);

posOrder(T->Right);

printf("%d ",T->Key);

}

}

//cau 7

Tree searchNode(int X, Tree T) {

if(T!=NULL) {

if(T->Key==X)

return T;

if(X<T->Key)

return searchNode(X, T->Left);

else return searchNode(X, T->Right);

}

return NULL;

}

//cau 8

Tree getParent(int X, Tree T) {

if(searchNode(X, T)!=NULL) {

if(X==T->Key) {

return NULL;

}

if(T->Left->Key==X || T->Right->Key==X) {

return T;

}

if(X<T->Key) {

return getParent(X, T->Left);

}

if(X>T->Key) {

return getParent(X, T->Right);

}

}

return NULL;

}

//cau 9

Tree rightSibling(int X, Tree T) {

if(searchNode(X, T)!=NULL) {

if(X==T->Key) {

return NULL;

}

if(T->Left->Key==X) {

return T->Right;

}

if(X<T->Key) return rightSibling(X, T->Left);

else return rightSibling(X, T->Right);

}

return NULL;

}

//cau 10

void printPath(int X, Tree T) {

if(T!=NULL) {

if(T->Key==X) {

printf("%d -> Tim thay", T->Key);

} else {

printf("%d ", T->Key);

if(X<T->Key) {

printPath(X, T->Left);

} else printPath(X, T->Right);

}

} else {

printf("-> Khong thay");

}

}

//cau 11

Tree maxNode(Tree T){

Tree p=T;

while (p->Right!=NULL)

p=p->Right;

return p;

}

Tree getPrevious(int x,Tree T){

Tree p=T,lleft=NULL;

while (p!=NULL){

if (p->Key==x){

if (p->Left==NULL) return lleft;

else return maxNode(p->Left);

}

else

if (x<p->Key) {

//lleft=p;

p=p->Left;}

else {lleft=p;

p=p->Right;

}

}

return NULL;

}

//cau 12

Tree getMinNode(Tree T) {

while(T->Left!=NULL) {

T=T->Left;

}

return T;

}

Tree getNext(int X, Tree T) {

Tree N=searchNode(X, T);

if(N->Right!=NULL) return getMinNode(N->Right);

else {

Tree P=getParent(X, T);

while(P!=NULL && X==P->Right->Key) {

X=P->Key;

P=getParent(P->Key, T);

}

return P;

}

return NULL;

}

//cau 13

int deleteMin(Tree \*T) {

int k;

if((\*T)->Left==NULL) {

k=(\*T)->Key;

(\*T)=NULL; //gan lai null

return k;

}

else return deleteMin(&(\*T)->Left);

}

void deleteNode(int x, Tree \*T) {

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

if(x<(\*T)->Key) deleteNode(x, &(\*T)->Left);

else if(x>(\*T)->Key) deleteNode(x, &(\*T)->Right);

else if((\*T)->Left==NULL && (\*T)->Right==NULL) (\*T)=NULL;

else if((\*T)->Left==NULL) (\*T)=(\*T)->Right;

else if((\*T)->Right==NULL) (\*T)=(\*T)->Left;

else (\*T)->Key=deleteMin(&(\*T)->Right);

}

}

int max(int a, int b) {

return a>b?a:b;

}

int getHeight(Tree T) {

if(T==NULL)

return -1;

return max(getHeight(T->Left), getHeight(T->Right)) + 1;

}

int hNode(int x, Tree T) {

if(searchNode(x, T)==NULL || T==NULL) return -1;

if(T->Key==x) return getHeight(T);

if(T->Key>x) return hNode(x, T->Left);

return hNode(x, T->Right);

}

int main()

{

Tree T = initTree();

if (T==NULL) {

printf("Da khoi tao thanh cong");

}

else{

printf("Loi");

}

return 0;

}