**Exercise 6: Binary Search Tree and its Applications**

**ADT.h:**

#include<stdio.h>

struct tree

{

int data;

struct tree \*left,\*right;

};

struct tree \* insert(struct tree\*,int);

//void inorder(struct tree \*);

int \* inorder(struct tree \*,int\*);

//int \* inorder(struct tree \*,int\*,int);

int height(struct tree\* T);

void printCurLvl(struct tree\* T, int lvl);

void printLvlOrd(struct tree\* T);

struct tree \* find(struct tree \*,int);

struct tree \* findMin(struct tree \*);

struct tree \* del(struct tree \*,int);

int isComplete(struct tree \*T);

**impl.h:**

#include "adt.h"

#include<stdlib.h>

struct tree \* insert(struct tree \*T,int ele)

{

if(T==NULL)

{

T=(struct tree \*)malloc(sizeof(struct tree \*));

T->data=ele;

T->left=NULL;

T->right=NULL;

}

else

{

if(T->data>ele)

T->left=insert(T->left,ele);

if(T->data<ele)

T->right=insert(T->right,ele);

}

return T;

}

/\*void inorder(struct tree \*T)

{

if(T)

{

inorder(T->left);

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

inorder(T->right);

}

}\*/

int \* inorder(struct tree \*T,int \*A)

{

//static int \*A=NULL;

if(T)

{

A=inorder(T->left,A);

//printf("%d ",T->data);

if(A==NULL)

{

A=(int \*)malloc(sizeof(int));

}

\*A++=T->data;

//printf("\n%d\n",\*A);

A=inorder(T->right,A);

}

return A;

}

/\*static int \*A=NULL;

static int i=1;

if(T)

{

inorder(T->left);

//printf("%d ",T->data);

if(A==NULL)

{

A=(int \*)malloc(sizeof(int));

A[0]=0;

}

A[i]=T->data;

A[0]=A[0]+1;

i++;

//printf("\n%d\n",\*A);

inorder(T->right);

}

return A;

}

int \* inorder(struct tree \*T,int \*A,int i)

{

//static int A[5],i=0;

if(T)

{

inorder(T->left,A,i);

//printf("%d ",T->data);

A[i++]=T->data;

//printf("\n%d\n",\*A);

inorder(T->right,A,i);

}

return A;

}\*/

int height(struct tree\* T)

{

if(T==NULL)

return 0;

else

{

int lheight = height(T->left);

int rheight = height(T->right);

if (lheight>rheight)

return lheight+1;

else

return rheight+1;

}

}

void printCurLvl(struct tree\* T, int lvl)

{

if(T==NULL)

return;

if(lvl==1)

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

else if(lvl>1) {

printCurLvl(T->left,lvl-1);

printCurLvl(T->right,lvl-1);

}

}

void printLvlOrd(struct tree\* T)

{

int h=height(T);

for(int i=1;i<=h;i++)

printCurLvl(T,i);

printf("\n");

}

struct tree\* find(struct tree \*T,int x)

{

if(!T||x==T->data)

return T;

else if(x<T->data)

return find(T->left,x);

return find(T->right,x);

}

struct tree \* findMin(struct tree \*T)

{

if(!T)

return T;

while(T->left!=NULL)

return findMin(T->left);

return T;

}

struct tree \* del(struct tree \*T,int x)

{

struct tree \*temp;

if(T==NULL)

return T;

if(x<T->data)

T->left=del(T->left,x);

else if(x>T->data)

T->right=del(T->right,x);

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

{

temp=findMin(T->right);

T->data=temp->data;

T->right=del(T->right,T->data);

}

else

{

temp=T;

if(!T->right)

T=T->left;

else if(!T->left)

T=T->right;

free(temp);

/\* if (T->left == NULL) {

temp = T->right;

free(T);

return temp;

}

else if (T->right == NULL) {

temp = T->left;

free(T);

return temp;

}\*/

}

return T;

}

int isComplete(struct tree \*T)

{

struct tree \*ptr = T;

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

{

if (isComplete(ptr->left) == 1 && isComplete(ptr->right) == 1)

{

return 1;

}

else

{

return 0;

}

}

else if (ptr->left == NULL && ptr->right == NULL)

{

return 1;

}

else

{

return 0;

}

}

**appl.c:**

#include "impl.h"

int main()

{

struct tree \*T=NULL;

int ele,n3=0;

/\*T=insert(T,29);

T=insert(T,23);

T=insert(T,4);

T=insert(T,13);

T=insert(T,39);

T=insert(T,31);

T=insert(T,45);

T=insert(T,56);

T=insert(T,49);

//insert(T,29);

inorder(T);

printf("\n");\*/

printf("\nMENU:\n1.Insert\n2.Delete\n3.Inorder Display\n4.Level Order traversal\n5.Find an element\n6.Find the minimum element\n7.Check whether 2 trees have same set of elements\n8.Check whether BST is complete or not\n9.Count number of nodes in a given range\n10.Exit\n");

int n1=0,n2=0;

int i;

struct tree \*T1=NULL;

struct tree \*T2=NULL;

/\*int A[]={10,5,50,1,40,100};

for(int i=0;i<6;i++)

{

T1=insert(T1,A[i]);

n1++;

T2=insert(T2,A[i]);

n2++;

}\*/

/\*T1=insert(T1,10);

T1=insert(T1,5);

T1=insert(T1,50);

T1=insert(T1,1);

T1=insert(T1,40);

T1=insert(T1,100);

T2=insert(T2,10);

T2=insert(T2,5);

T2=insert(T2,50);

T2=insert(T2,1);

T2=insert(T2,40);

T2=insert(T2,100);\*/

//inorder(T1);

int \*A1=(int \*)malloc(sizeof(int));;

int \*A2=(int\*)malloc(sizeof(int));

int \*A3;

//A1--;

//A2--;

//A1++;

//printf("\n%d\n",A2[0]);

//A2--;

/\*for(int i=100;i>0;i--)

{

printf("%d ",A2[i]);

}\*/

//A1=A1--;

//A2=inorder(T2);

for(i=0;i<n1;i++)

{

//printf("%d ",\*A2--);

A1--;

}

for(i=0;i<n2;i++)

{

//printf("%d ",\*A2--);

A2--;

}

/\*for(int i=0;i<n2;i++)

{

printf("%d %d ",A1[i],A2[i]);

}\*/

/\*T=insert(T,45);

/\*T=insert(T,45);

T=insert(T,56);

T=insert(T,49);\*/

//int ch=1;

int ch;

do

{

printf("\nEnter your choice: ");

scanf("%d",&ch);

switch(ch)

{

case 7://printf("\nIdentical Trees:\n");

printf("\nEnter total number of elements for T1: ");

scanf("%d",&n1);

for(i=0;i<n1;i++)

{printf("Enter the element: ");

scanf("%d",&ele);

T1=insert(T1,ele);}

printf("\nEnter total number of elements for T2: ");

scanf("%d",&n2);

for(i=0;i<n2;i++)

{printf("Enter the element: ");

scanf("%d",&ele);

T2=insert(T2,ele);}

A1=inorder(T1,A1);

A2=inorder(T2,A2);

for(i=0;i<n1;i++)

{

//printf("%d ",\*A2--);

A1--;

}

for(i=0;i<n2;i++)

{

//printf("%d ",\*A2--);

A2--;

}

printf("\nTree 1: ");

printLvlOrd(T1);

printf("Tree 2: ");

printLvlOrd(T2);

if(n1!=n2)

printf("\nThey dont have same elements!\n");

else

{

for(i=0;i<n1;i++)

{

if(A1[i]!=A2[i])

break;

}

if(i==n1)

printf("\nThey have same elements!\n");

else

printf("\nThey don't have same elements!\n");

}

break;

case 8://printf("\nComplete Tree:\n");

printf("\nEnter total number of elements for T1: ");

scanf("%d",&n1);

for(i=0;i<n1;i++)

{printf("Enter the element: ");

scanf("%d",&ele);

T1=insert(T1,ele);}

for(i=0;i<n1;i++)

{

//printf("%d ",\*A2--);

A1--;

}

if(isComplete(T1) == 1)

printf("TREE1 IS COMPLETE");

else

printf("\nTREE1 IS NOT COMPLETE");

if(isComplete(T2) == 1)

printf("TREE2 IS COMPLETE");

else

printf("\nTREE2 IS NOT COMPLETE");

break;

case 9: //printf("\nNo of elements btw a & b:\n");

printf("\nEnter a & b for [a,b]: ");

int a,b,c1=0,c2=0;

scanf("%d %d",&a,&b);

printf("\nTree 1: ");

for(i=0;i<n1;i++)

{

if(A1[i]>=a && A1[i]<=b)

{

//printf("%d ",A1[i]);

c1++;

}

}

printf("%d",c1);

printf("\nTree 2: ");

for(i=0;i<n2;i++)

{

if(A2[i]>=a && A2[i]<=b)

{

//printf("%d ",A2[i]);

c2++;

}

}

printf("%d\n",c2);

break;

case 4: printf("\nLevel Order:\n");

/\*printf("\nTree 1: ");

printLvlOrd(T1);

printf("\nTree 2: ");

printLvlOrd(T2);\*/

printLvlOrd(T);

break;

case 5://printf("\nFind an element:\n");

printf("Enter the element to find: ");

scanf("%d",&ele);

struct tree \*temp;//=NULL;//(struct tree\*)malloc(sizeof(struct tree \*));

//printf("\nTree 1: ");

temp=find(T,ele);

if(temp)

printf("\nElement %d found! \n",temp->data);

else

printf("\nElement is not found!\n");

/\*printf("\nTree 2: ");

temp=find(T2,100);

if(temp)

printf("\nElement found! %d\n",temp->data);

else

printf("\nElement not found!\n");\*/

break;

case 6: //printf("\nFind the minimum element:\n");

//temp=(struct tree\*)malloc(sizeof(struct tree \*));

//printf("\nTree 1: ");

temp=findMin(T);

if(temp)

printf("\nMinimum element: %d\n",temp->data);

else

printf("\nNo element is there in the tree\n");

/\*printf("\nTree 2: ");

temp=findMin(T2);

if(temp)

printf("\nMinimum element: %d\n",temp->data);

else

printf("\nNo element is there in the tree\n");\*/

break;

case 2: //printf("\nDeletion:\n");

/\*printf("\nTree 1: ");

T1=del(T1,1);

//int \*A3=inorder(T1);

//printLvlOrd(T1);

printf("\nTree 2: ");

T2=del(T2,100);

printLvlOrd(T2);\*/

printf("Enter an element to delete: ");

scanf("%d",&ele);

//for(i=0;i<n3;i++)

if(find(T,ele))

{

T=del(T,ele);

n3--;

}

break;

case 1: printf("Enter the element: ");

scanf("%d",&ele);

T=insert(T,ele);

n3++;

break;

case 3:printf("Inorder Traversal: ");

A3=(int\*)malloc(sizeof(int));

A3=inorder(T,A3);

for(i=0;i<n3;i++)

{

//printf("%d ",\*A2--);

A3--;

}

for(i=0;i<n3;i++)

{

printf("%d ",A3[i]);

}

printf("\n");

break;

case 10: printf("\nExiting...");

break;

default:printf("\nInvalid Input!\n");

break;

}

//ch++;

}while(ch!=10);

}

**Sample I/O:**





