adt.h:

struct BST

{

    int data;

    struct BST \*left,\*right;

};

struct BST \* insert(struct BST \*t,int x);

struct BST \* del(struct BST \*t,int x);

void inorder(struct BST \*t);

void levelorder(struct BST\*t);

struct BST\* find(struct BST\* t,int x);

struct BST\*findmin(struct BST\*t);

impl.h:

#include "adt.h"

#include<stdlib.h>

#include<stdio.h>

struct BST \* insert(struct BST \*t,int x)

{

    if(t==NULL)

    {

        t=(struct BST \*)malloc(sizeof(struct BST));

        t->data=x;

        t->left=NULL;

        t->right=NULL;

    }

    else

    {

        if(x<t->data)

            t->left=insert(t->left,x);

        else

            t->right=insert(t->right,x);

    }

    return t;

}

struct BST\* del(struct BST \*t,int x)

{

    struct BST\* temp;

    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==NULL)

            t=t->left;

        else if(t->left==NULL)

            t=t->right;

        free(temp);

    }

    return t;

}

void inorder(struct BST \*t)

{

    if(t->left!=NULL)

        inorder(t->left);

    printf("%d ",t->data);

    if(t->right!=NULL)

        inorder(t->right);

}

int height(struct BST\*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 printorder(struct BST\*t,int lvl)

{

    if(t==NULL)

        return;

    if(lvl==1)

        printf("%d ",t->data);

    printorder(t->left,lvl-1);

    printorder(t->right,lvl-1);

}

void levelorder(struct BST\*t)

{

    int h=height(t);

    for (int i=1;i<=h;i++)

    {

        printf("\nLevel %d:",i);

        printorder(t,i);

    }

}

struct BST\* find(struct BST\* t,int x)

{

    if(t==NULL)

    {

    return NULL;

    }

    if(t->data<x)

        return find(t->right,x);

    else if(t->data>x)

        return find(t->left,x);

    else

        return t;

}

struct BST\* findmin(struct BST\*t)

{

    if(t->left!=NULL)

        return findmin(t->left);

    return t;

}

Appl.c:

#include "impl.h"

#include<stdio.h>

int main()

{

    struct BST \*t,\*t1=NULL,\*t2=NULL;

    t=NULL;

    t=insert(t,29);

    insert(t,23);

    insert(t,4);

    insert(t,13);

    insert(t,39);

    insert(t,31);

    insert(t,45);

    insert(t,56);

    insert(t,49);

    inorder(t);

    levelorder(t);

    printf("\nAfter deletion:\n");

    del(t,39);

    inorder(t);

    struct BST \*d=findmin(t);

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

    if(d=find(t,13))

        printf("\nAddress of 13:%p\n",d);

    else

        printf("Not found\n");

    if(d=find(t,3))

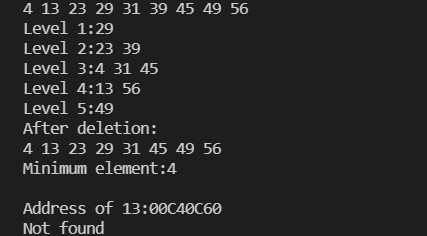
        printf("Address of 3:%p",d);

    else

        printf("Not found\n");

}

o/p:



Applications:

Adt.h:

struct BST

{

    int data;

    struct BST \*left,\*right;

};

struct BST \* insert(struct BST \*t,int x);

struct BST \* del(struct BST \*t,int x);

void inorder(struct BST \*t);

void levelorder(struct BST\*t);

struct BST\* find(struct BST\* t,int x);

struct BST\*findmin(struct BST\*t);

void inorderModify(struct BST\*t,int arr[],int \*n);

impl.h:

#include "adt.h"

#include<stdlib.h>

#include<stdio.h>

struct BST \* insert(struct BST \*t,int x)

{

    if(t==NULL)

    {

        t=(struct BST \*)malloc(sizeof(struct BST));

        t->data=x;

        t->left=NULL;

        t->right=NULL;

    }

    else

    {

        if(x<t->data)

            t->left=insert(t->left,x);

        else

            t->right=insert(t->right,x);

    }

    return t;

}

struct BST\* del(struct BST \*t,int x)

{

    struct BST\* temp;

    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==NULL)

            t=t->left;

        if(t->left==NULL)

            t=t->right;

    }

    return t;

}

void inorder(struct BST \*t)

{

    if(t->left!=NULL)

        inorder(t->left);

    printf("%d ",t->data);

    if(t->right!=NULL)

        inorder(t->right);

}

struct BST\* find(struct BST\* t,int x)

{

    if(t==NULL)

    {

    return NULL;

    }

    if(t->data<x)

        return find(t->right,x);

    else if(t->data>x)

        return find(t->left,x);

    else

        return t;

}

struct BST\* findmin(struct BST\*t)

{

    if(t->left!=NULL)

        return findmin(t->left);

    return t;

}

void inorderModify(struct BST\*t,int arr[],int \*n)

{

    if(t->left!=NULL)

        inorderModify(t->left,arr,n);

    arr[\*n]=t->data;

    (\*n)++;

    if(t->right!=NULL)

        inorderModify(t->right,arr,n);

}

Appl.c:

#include "impl.h"

#include<stdio.h>

void sameBST(struct BST\*t1,struct BST\*t2,int arr1[],int arr2[],int \*n1,int \*n2)

{

    inorderModify(t1,arr1,n1);

    inorderModify(t2,arr2,n2);

    if(\*n1!=\*n2)

        printf("Not same BST\n");

    else

    {

        for(int i=0;i<\*n1;i++)

        {

            if(arr1[i]!=arr2[i])

            {

                printf("Not same BST\n");

                return;

            }

        }

        printf("Same BST\n");

    }

}

int complete(struct BST\* t1,int index,int n)

{

    if(t1==NULL)

        return 1;

    if(index>=n)

        return 0;

    return (complete(t1->left,2\*index+1,n)&&complete(t1->right,2\*index+2,n));

}

int nodesrange(struct BST\*t,int arr[],int n,int l,int r)

{

    int count=0;

    for(int i=0;i<n;i++)

    {

        if(arr[i]>=l && arr[i]<=r)

            count++;

    }

    return count;

}

int main()

{

    struct BST \*t=NULL,\*p=NULL;

    t=insert(t,10);

    insert(t,5);

    insert(t,50);

    insert(t,1);

    insert(t,40);

    insert(t,100);

    p=insert(p,10);

    insert(p,5);

    insert(p,50);

    insert(p,1);

    insert(p,40);

    insert(p,100);

    int arr1[100],arr2[100];

    int n1=0,n2=0;

    sameBST(t,p,arr1,arr2,&n1,&n2);

    if(complete(t,0,n1))

        printf("Complete tree\n");

    else

        printf("Not a complete tree\n");

    int l,r;

    printf("Limits of range:");

    scanf("%d %d",&l,&r);

    printf("No.of nodes in the given range:%d\n",nodesrange(t,arr1,n1,l,r));

}

o/p:

