//program of binary search

#include<stdio.h>

#include"btree.h"

#include<stdlib.h>

void main()

{

    struct bst \*root=NULL,\*temp,\*c,\*mirrorroot;

    int ch,n,i,val,x;

    printf("\n1.create");

    printf("\n2.insert");

    printf("\n3.preorder");

    printf("\n4.inorder");

    printf("\n5.postorder");

    printf("\n6.search");

    printf("\n7.no.of leaf");

    printf("\n8.no.of nonleaf");

    printf("\n9.no.of Node");

    printf("\n10.copy tree");

    printf("\n11.delete");

    printf("\n12.mirror tree");

    printf("\n13.compare");

    printf("\n14.sum of even node");

    printf("\n15.sum of odd node");

    printf("\n16.exit");

    while(1){

        printf("\nenter your choice :");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1:

            root=NULL;

            printf("\nenter number of node:");

            scanf("%d",&n);

            for(i=1;i<=n;i++)

            {

                printf("\nenter data for node %d :",i);

                scanf("%d",&val);

                root=create(root,val);

            }

            break;

            case 2:

            printf("\nenter the data to insert:");

            scanf("%d",&val);

            root=insert(root,val);

            break;

            case 3:preorder(root);

            break;

            case 4:inorder(root);

            break;

            case 5:postorder(root);

            break;

            case 6:printf("\nenter number to search:");

            scanf("%d",&val);

            temp=search(root,val);

            if(temp==NULL)

            printf("\n%d is not found",val);

            else

            printf("\n%d is found",val);

            break;

            case 7:

            val=CountLeaf(root);

            printf("\n leaf node in the trees are:  %d",val);

            break;

            case 8:

            val=countNonLeaf(root);

            printf("\n Non leaf node in the trees are:  %d",val);

            break;

            case 9:

            val=CountNode(root);

            printf("\n all node in the trees are:  %d",val);

            break;

            case 10:

            c=(struct bst \*)malloc(sizeof(struct bst));

            c->lchild=NULL;

            c->rchild=NULL;

            copytree(root,c);

            inorder(c);

            break;

            case 11:

            printf("\nenter thr node to be deleted");

            scanf("%d",&val);

            root=delete(root,val);

            break;

            case 12:

            mirrorroot=(struct bst\*)malloc(sizeof(struct bst));

            mirrorroot->lchild=NULL;

            mirrorroot->rchild=NULL;

            mirror(root,mirrorroot);

            inorder(mirrorroot);

            break;

            case 13:i=compare(root,c);

            if(i==1)

            printf("tree is equal");

            else

            printf("tree is not equal");

            break;

            case 14:x=sumeven(root);

            printf("\nsum of even node: %d",x);

            break;

            case 15:x=sumodd(root);

            printf("\nsum of odd node: %d",x);

            break;

            case 16:exit(0);

            default:

            printf("enter valid choice");

            break;

        }

    }

}

#include<stdio.h>

#include<stdlib.h>

struct bst

{

    int data;

    struct bst \*lchild,\*rchild;

};

int addeven=0,addodd=0;

struct bst \*create(struct bst \*root,int val)

{

    if(root == NULL)

    {

        root=(struct bst \*)malloc(sizeof(struct bst));

        root->lchild=root->rchild=NULL;

        root->data=val;

        return root;

    }

    else

    {

        if(val<root->data)

            root->lchild=create(root->lchild,val);

        else if(val>root->data)

            root->rchild=create(root->rchild,val);

        else

            printf("dublicate number not allow");

        return(root);

    }

}

struct bst \*insert(struct bst \*root,int key)

{

    if(root==NULL)

    {

        root=(struct bst\*)malloc(sizeof(struct bst));

        root->data=key;

        root->lchild=root->rchild=NULL;

        return(root);

    }

    if(key<root->data)

    root->lchild=insert(root->lchild,key);

    else

    root->rchild=insert(root->rchild,key);

    return root;

}

void preorder(struct bst \*root)

{

    if(root != NULL)

    {

        printf("%d ",root->data);

        preorder(root->lchild);

        preorder(root->rchild);

    }

}

void inorder(struct bst \*root)

{

    if(root != NULL)

    {

        inorder(root->lchild);

        printf("%d ",root->data);

        inorder(root->rchild);

    }

}

void postorder(struct bst \*root)

{

    if(root != NULL)

    {

        postorder(root->lchild);

        postorder(root->rchild);

        printf("%d ",root->data);

    }

}

int CountNode(struct bst \*root)

{

    if(root == NULL)

    return 0;

    else

    return(1+CountNode(root->lchild))+(CountNode(root->rchild));

}

int CountLeaf(struct bst \*root)

{

    if(root == NULL)

        return(0);

    else

    if(root->lchild==NULL && root->rchild==NULL)

    return(1);

    else

    return((CountLeaf(root->lchild)+CountLeaf(root->rchild)));

}

int countNonLeaf(struct bst \*root)

{

    int c=0;

    if(root==NULL)

    return 0;

    if(root->lchild==NULL && root->rchild == NULL)

    return 0;

    return(1+countNonLeaf(root->lchild)+countNonLeaf(root->rchild));

}

struct bst \*search(struct bst \*root,int data)

{

    if(root==NULL)

    {

        return(NULL);

    }

    if(root->data==data)

    {

        return(root);

    }

    else if(data<root->data)

    return(search(root->lchild,data));

    else

    return(search(root->rchild,data));

}

void copytree(struct bst \*root,struct bst \*c)

{

    if(root==NULL)

    return;

    c->data=root->data;

    if(root->lchild)

    {

        c->lchild=(struct bst\*)malloc(sizeof(struct bst));

        c->lchild->lchild=NULL;

        c->lchild->rchild=NULL;

        copytree(root->lchild,c->lchild);

    }

    if(root->rchild)

    {

        c->rchild=(struct bst\*)malloc(sizeof(struct bst));

        c->rchild->lchild=NULL;

        c->rchild->rchild=NULL;

        copytree(root->rchild,c->rchild);

    }

}

struct bst \*min1(struct bst \*root)

{

    struct bst \*c=root;

    while(c&&c->lchild!=NULL)

    {

        c=c->lchild;

    }

    return c;

}

struct bst \*delete(struct bst \*root,int data)

{

    if(root==NULL)

    return root;

    if(data<root->data)

    root->lchild=delete(root->lchild,data);

    else if(data>root->data)

    root->rchild=delete(root->rchild,data);

    else

    {

        if(root->lchild==NULL)

        {

            struct bst \*temp=root->rchild;

            free(root);

            return temp;

        }

        else if(root->rchild==NULL)

        {

            struct bst \*temp=root->lchild;

            free(root);

            return temp;

        }

        struct bst \*temp=min1(root->rchild);

        root->data=temp->data;

        root->rchild=delete(root->rchild,data);

    }

    return root;

}

void mirror(struct bst \*r,struct bst \*c)

{

    if(r==NULL)

        return ;

    c->data=r->data;

    if(r->lchild)

    {

        c->rchild=(struct bst \*)malloc(sizeof(struct bst));

        c->rchild->lchild=NULL;

        c->rchild->rchild=NULL;

        mirror(r->lchild,c->rchild);

    }

    if(r->rchild)

    {

        c->lchild=(struct bst\*)malloc(sizeof(struct bst));

        c->lchild->lchild=NULL;

        c->lchild->rchild=NULL;

        mirror(r->rchild,c->lchild);

    }

}

int compare(struct bst \*r,struct bst \*c)

{

    if(r==NULL && c==NULL)

    return 1;

    if(r==NULL || c==NULL)

    return 0;

    if(compare(r->lchild,c->lchild)&&compare(r->rchild,c->rchild)&&r->data==c->data)

    return 1;

    else

    return 0;

}

int sumeven(struct bst \*root)

{

    if(root!=NULL)

    {

        if(root->data%2==0)

        addeven=addeven+root->data;

        sumeven(root->lchild);

        sumeven(root->rchild);

    }

    return addeven;

}

int sumodd(struct bst \*root)

{

    if(root!=NULL)

    {

        if(root->data%2!=0)

        addodd=addodd+root->data;

        sumodd(root->lchild);

        sumodd(root->rchild);

    }

    return addodd;

}

//Create a Binary Search Tree & Display Nodes level wise

#include<stdio.h>

#include<stdlib.h>

#define MAX 20

struct Btree

{

    struct Btree \*lchild;

    int data;

    struct Btree \*rchild;

};

typedef struct Btree BNODE;

struct Queue

{

    struct Btree \*Q[MAX];

    int front,rear;

};

typedef struct Queue QUEUE;

void Addq(QUEUE \*q,BNODE \*t)

{

    q->Q[++q->rear]=t;

}

BNODE \* Delq(QUEUE \*q)

{

    return(q->Q[++q->front]);

}

int isempty(QUEUE \*q)

{

    if(q->front==q->rear)

        return 1;

    else

        return 0;

}

void LevelDisp(BNODE \*h)

{

    QUEUE q;

    BNODE \*temp;

    int total=0,cnt=0,level=0;

    q.front=q.rear=-1;

    temp=h;

    Addq(&q,temp);

    Addq(&q,NULL);

    while(!isempty(&q))

    {

        temp=Delq(&q);

        if(temp==NULL)

        {

            if(!isempty(&q))

            {

                printf(" = %d Level-%d\n",cnt,level);

                cnt=0;

                level++;

                Addq(&q,NULL);

            }

        }

        else

        {

            total++;

            cnt++;

            printf("%d ",temp->data);

            if(temp->lchild!=NULL)

                Addq(&q,temp->lchild);

            if(temp->rchild!=NULL)

                Addq(&q,temp->rchild);

        }

    }

    printf("=%d Level-%d\n",cnt,level);

    printf("\nTotal Nodes:%d",total);

    printf("\nTotal Levels:%d\n",level+1);

}

BNODE\* newNode(int data)

{

    BNODE\* node = (BNODE\*)malloc(sizeof(BNODE));

    node->data = data;

    node->lchild = NULL;

    node->rchild = NULL;

    return(node);

}

void main()

{

    BNODE \*header = newNode(100);

    header->lchild = newNode(50);

    header->rchild = newNode(200);

    header->rchild->lchild=newNode(150);

    header->lchild->lchild = newNode(20);

    header->lchild->rchild = newNode(80);

    header->lchild->rchild->lchild = newNode(70);

    LevelDisp(header);

}

 //C Program to sort an array based on heap sort algorithm(MAX heap)

#include <stdio.h>

void main()

{

    int heap[10], n, i, j, c, root, temp;

    printf("\n Enter no of elements :");

    scanf("%d", &n);

    printf("\n Enter the nos : ");

    for (i = 0; i < n; i++)

        scanf("%d", &heap[i]);

    for (i = 1; i < n; i++)

    {

        c = i;

        do

        {

            root = (c - 1) / 2;

            if (heap[root] < heap[c])   /\* to create MAX heap array \*/

            {

                temp = heap[root];

                heap[root] = heap[c];

                heap[c] = temp;

            }

            c = root;

        } while (c != 0);

    }

    printf("Heap array : ");

    for (i = 0; i < n; i++)

        printf("%d\t ", heap[i]);

    for (j = n - 1; j >= 0; j--)

    {

        temp = heap[0];

        heap[0] = heap[j];    /\* swap max element with rightmost leaf element \*/

        heap[j] = temp;

        root = 0;

        do

        {

            c = 2 \* root + 1;    /\* left node of root element \*/

            if ((heap[c] < heap[c + 1]) && c < j-1)

                c++;

            if (heap[root]<heap[c] && c<j)    /\* again rearrange to max heap array \*/

            {

                temp = heap[root];

                heap[root] = heap[c];

                heap[c] = temp;

            }

            root = c;

        } while (c < j);

    }

    printf("\n The sorted array is : ");

    for (i = 0; i < n; i++)

    printf("\t %d", heap[i]);

}