**Assignment-2**

* ***Binary Search Tree***

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

int data;

struct node \*left,\*right;

}node;

typedef struct

{

node \*root;

}Tree;

node\* fatherOf(Tree \*t,node \*p)

{

node \*q;

//if p is root node no father

if(p==t->root)

return NULL;

q=t->root;

while(q!=NULL)

{

if(q->left==p|| q->right==p)

return q;

if(p->data<=q->data)

q=q->left;

else

q=q->right;

}

return NULL;

}

node\* findMax(node \*r)

{

if(r->right==NULL)

return r;

else

return findMax(r->right);

}

int search(Tree \*t,int ele)

{

node \*p=t->root;

while(p!=NULL)

{

if(ele==p->data)

return 1;

else if(ele<p->data)

p=p->left;

else

p=p->right;

}

return 0;

}

void insertNode(Tree \*t,int ele)

{

node \*newrec,\*p;

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

newrec->data=ele;

newrec->left=newrec->right=NULL;

//agar root null hai

if(t->root==NULL)

{

t->root=newrec;

return;

}

p=t->root;

while(p!=NULL)

{

if(ele<=p->data)

{

if(p->left==NULL)

{

p->left=newrec;

return;

}

else

p=p->left;

}

else if(p->right==NULL)

{

p->right=newrec;

return;

}

else

p=p->right;

}//while

}

void deleteNode(Tree \*t,int ele)

{

node \*p,\*f;

if(t->root==NULL)

{printf("BST IS EMPTY\n");

return;

}

//search for ele

p=t->root;

while(p!=NULL)

{

if(ele==p->data)

break;

if(ele<p->data)

p=p->left;

else

p=p->right;

}

if(p==NULL)

{

printf("node %d is not present\n",ele);

return;

}

//if p is leaf node

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

{

if(p==t->root)

{

t->root=NULL;

return;

}

f=fatherOf(t,p);

if(f->left==p)

f->left=NULL;

else

f->right=NULL;

return;

}

//p has left child

if(p->left!=NULL)

{

int val;

node \*max= findMax(p->left);

val=max->data;

deleteNode(t,max->data);

p->data=val;

return;

}

// p has right child

if(p==t->root)

{

t->root=t->root->right;

return;

}

f=fatherOf(t,p);

if(f->left==p)

f->left=p->right;

else

f->right=p->right;

}

void inorder(node \*x)

{

if(x!=NULL)

{

inorder(x->left);

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

inorder(x->right);

}

}

void preorder(node \*x)

{

if(x!=NULL)

{

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

preorder(x->left);

preorder(x->right);

}

}

void postorder(node \*x)

{

if(x!=NULL)

{

postorder(x->left);

postorder(x->right);

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

}

}

int main()

{

Tree t;

int ch,ele;

t.root=NULL;

while(1)

{

printf("\n1:InsertNode\t2:DeleteNode\t3:Search\t4:Inorder\t5:Preorder\n");

printf("6:PostOrder\t7:Exit\n");

printf("enter choice\n");

scanf("%d",&ch);

if(ch==7)

break;

switch(ch)

{

case 1:printf(" enter element to be inserted\n");

scanf("%d",&ele);

insertNode(&t,ele);

printf(" tree nodes are\n");

inorder(t.root);

break;

case 2:printf(" enter element to be deleted\n");

scanf("%d",&ele);

deleteNode(&t,ele);

printf(" tree nodes are\n");

inorder(t.root);

break;

case 3:printf(" enter element to be search\n");

scanf("%d",&ele);

printf(" tree nodes are\n");

inorder(t.root);

if(search(&t,ele))

printf(" node %d is present\n",ele);

else

printf(" node %d is not present\n",ele);

break;

case 4:printf(" tree nodes in inorder sequence\n");

inorder(t.root);

break;

case 5:printf(" tree nodes in preorder sequence\n");

preorder(t.root);

break;

case 6:printf(" tree nodes in postorder sequence\n");

postorder(t.root);

break;

default:printf(" invalid choice\n");

}

}

return 0;

}