/\*

#include<stdio.h>

#include<stdlib.h>

struct node{

int value;

struct node \*left;

struct node \*right;

}\*root=NULL,\*temp=NULL,\*t2,\*t1,\*tp;

int key;

int s;

int n;

int count;

void search(struct node \*t);

void insert()

{

int data;

printf("Enter data to be inserted-");

scanf("%d",&data);

temp=(struct node\*)malloc(sizeof(struct node));

temp->value=data;

temp->left=temp->right=NULL;

if(root==NULL)

root=temp;

else

search(root);

}

void search(struct node\*t)

{

if((temp->value>t->value)&&(t->right!=NULL))

search(t->right);

else if((temp->value>t->value)&&(t->right==NULL))

t->right=temp;

else if ((temp->value<t->value)&&(t->left!=NULL))

search(t->left);

else if((temp->value<t->value)&&(t->left==NULL))

t->left=temp;

}

void inorder(struct node \*t)

{

if(root==NULL)

{

printf("No elements in the tree\n");

return;

}

if(t->left!=NULL)

inorder(t->left);

printf("%d->",t->value);

if(t->right!=NULL)

inorder(t->right);

}

void preorder(struct node \*t)

{

if(root==NULL)

{

printf("No elements in the tree\n");

return;

}

printf("%d->",t->value);

if(t->left!=NULL)

preorder(t->left);

if(t->right!=NULL)

preorder(t->right);

}

void postorder(struct node \*t)

{

if(root==NULL)

{

printf("No elements in the tree\n");

return;

}

if(t->left!=NULL)

postorder(t->left);

if(t->right!=NULL)

postorder(t->right);

printf("%d->",t->value);

}

void searching(struct node \*t, int item)

{

if(t==NULL)

{

printf("Tree is empty");

return;

}

while(t!=NULL)

{

if(item==t->value)

break;

if(item<t->value)

t=t->left;

else

t=t->right;

}

if(t==NULL)

{

printf("Item not found");

return;

}

printf("Item found");

}

int count\_node\_tree(struct node \*t)

{

if(t!=NULL)

{

count\_node\_tree(t->left);

count++;

count\_node\_tree(t->right);

}

return count;

}

int leaf(struct node \*t)

{

if(t!=NULL)

{

leaf(t->left);

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

count++;

leaf(t->right);

}

return count;

}

struct node\* minValueNode(struct node\* t)

{

struct node\* current = t;

while (current && current->left != NULL)

current = current->left;

return current;

}

struct node\* maxvaluenode(struct node\* t)

{

struct node\* current = t;

while (current && current->right != NULL)

current = current->right;

return current;

}

struct node\* deleteNode(struct node\* t, int key)

{

if (t == NULL)

return root;

// If the key to be deleted

// is smaller than the root's

// key, then it lies in left subtree

if (key < t->value)

t->left = deleteNode(t->left, key);

// If the key to be deleted

// is greater than the root's

// key, then it lies in right subtree

else if (key > t->value)

t->right = deleteNode(t->right, key);

// if key is same as root's key,

// then This is the node

// to be deleted

else {

// node with only one child or no child

if (t->left == NULL) {

struct node\* temp = t->right;

free(t);

return temp;

}

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

struct node\* temp = t->left;

free(t);

return temp;

}

// node with two children:

// Get the inorder successor

// (smallest in the right subtree)

struct node\* temp = minValueNode(t->right);

// Copy the inorder

// successor's content to this node

t->value = temp->value;

// Delete the inorder successor

t->right = deleteNode(t->right, temp->value);

}

return t;

}

int main()

{

int choice;

while(1)

{

printf("\n\*\*\*menu\*\*\n");

printf("1. insert an element into tree\n");

printf("2. inorder traversal\n");

printf("3. preorder traversal\n");

printf("4. postorder traversal\n");

printf("5. for deletion\n");

printf("6. for searching an element\n");

printf("7. count the no of nodes\n");

printf("8.count the no of leaf nodes\n");

printf("9. to get the min value in the tree\n");

printf("10. to get the max value in the tree\n");

printf("11. to exit\n");

printf("Enter your choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

insert();

break;

case 2:printf("Inorder traversal\n");

inorder(root);

break;

case 3: printf("preprder traversal\n");

preorder(root);

break;

case 4: printf("postorder traversal\n");

postorder(root);

break;

case 5:printf("Enter the value to be deleted\n");

scanf("%d",&n);

root=deleteNode(root,n);

printf("inorder traversal of the modified tree\n");

inorder(root);

break;

case 6:printf("Enter the value to be searched\n");

scanf("%d",&s);

searching(root,s);

break;

case 7:printf("NO of nodes=%d",count\_node\_tree(root));

break;

case 8:printf("no of leaf nodes=%d",leaf(root));

break;

case 9:tp=minValueNode(root);

printf("MIn value node=%d",tp->value);

break;

case 10:tp=maxvaluenode(root);

printf("Max value node =%d",tp->value);

break;

case 11:exit(0);

default:printf("Invalid choice");

break;

}

}

return 0;

}

\*/

#include<stdio.h>

#include<stdlib.h>

struct node{

int value;

struct node \*left;

struct node \*right;

}\*root=NULL,\*temp=NULL,\*t2,\*t1,\*tp;

int key;

int s;

int n;

int count;

void search(struct node \*t);

void insert()

{

int data;

printf("Enter data to be inserted-");

scanf("%d",&data);

temp=(struct node\*)malloc(sizeof(struct node));

temp->value=data;

temp->left=temp->right=NULL;

if(root==NULL)

root=temp;

else

search(root);

}

void search(struct node\*t)

{

if((temp->value>t->value)&&(t->right!=NULL))

search(t->right);

else if((temp->value>t->value)&&(t->right==NULL))

t->right=temp;

else if ((temp->value<t->value)&&(t->left!=NULL))

search(t->left);

else if((temp->value<t->value)&&(t->left==NULL))

t->left=temp;

}

void inorder(struct node \*t)

{

if(root==NULL)

{

printf("No elements in the tree\n");

return;

}

if(t->left!=NULL)

inorder(t->left);

printf("%d->",t->value);

if(t->right!=NULL)

inorder(t->right);

}

void preorder(struct node \*t)

{

if(root==NULL)

{

printf("No elements in the tree\n");

return;

}

printf("%d->",t->value);

if(t->left!=NULL)

preorder(t->left);

if(t->right!=NULL)

preorder(t->right);

}

void postorder(struct node \*t)

{

if(root==NULL)

{

printf("No elements in the tree\n");

return;

}

if(t->left!=NULL)

postorder(t->left);

if(t->right!=NULL)

postorder(t->right);

printf("%d->",t->value);

}

struct node\* maxvaluenode(struct node\* t)

{

struct node\* current = t;

while (current && current->right != NULL)

current = current->right;

return current;

}

int main()

{

int choice;

while(1)

{

printf("\n\*\*\*menu\*\*\n");

printf("1. insert an element into tree\n");

printf("2. to get the max value in the tree\n");

printf("3. to print the tree elements in inorder traversal\n");

printf("4. to print the tree elements in preorder traversal\n");

printf("5. to print the tree elements in postorder traversal\n");

printf("6. to exit\n");

printf("Enter your choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1: insert();

break;

case 2:tp=maxvaluenode(root);

printf("\*\*\*\*\*MAX VALUE NODE\*\*\*\*");

printf("Node with Maximum value =%d",tp->value);

break;

case 3:printf("\*\*\*inorder traversal\*\*\*\n");

inorder(root);

break;

case 4:printf("\*\*\*preorder traversal\*\*\*\n");

preorder(root);

break;

case 5:printf("\*\*\*postorder traversal\*\*\*\n");

postorder(root);

break;

case 6:exit(0);

default:printf("Invalid choice");

break;

}

}

return 0;

}