#include<stdio.h>

#include<stdlib.h>

#include<string.h>

/\* BST node \*/

struct node

{

int data;

struct node \*left\_child;

struct node \*right\_child;

};

typedef struct node\* NODE;

/\* Function to create a newnode \*/

NODE getnode()

{

NODE p;

p = (NODE) malloc(sizeof(struct node));

if(p== NULL)

{

printf("Insufficient Memory");

exit(0);

}

p->left\_child= NULL;

p->right\_child = NULL;

return p;

}

NODE Create()

{

NODE q;

int x;

printf( "\nEnter the data to be inserted ");

scanf("%d",&x);

if( x == -1)

return NULL;

q= getnode();

q->data =x;

printf("\nEnter the Left child of %d",x);

q->left\_child=Create();

printf("\nEnter the Right child of %d",x);

q->right\_child=Create();

return q;

}

void InOrder( NODE p)

{

if( p ) // p != NULL

{

InOrder(p->left\_child);

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

InOrder(p->right\_child);

}

}

void PreOrder( NODE p)

{

if( p != NULL)

{

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

PreOrder(p->left\_child);

PreOrder(p->right\_child);

}

}

void PostOrder( NODE p)

{

if( p)

{

PostOrder(p->left\_child);

PostOrder(p->right\_child);

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

}

}

int Count( NODE p)

{

int c1,c2,c;

if( p== NULL)

return 0;

c1 = Count(p->left\_child);

c2 = Count(p->right\_child);

c = 1+ c1 + c2;

return c;

}

int CountLeaf( NODE p)

{

int c1,c2,c;

if( p== NULL)

return 0;

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

return 1;

c1 = CountLeaf(p->left\_child);

c2 = CountLeaf(p->right\_child);

c = c1 + c2;

return c;

}

int CountNonLeaf( NODE p)

{

int c1,c2,c;

if( p== NULL)

return 0;

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

return 0;

c1 = CountNonLeaf(p->left\_child);

c2 = CountNonLeaf(p->right\_child);

c = 1+ c1 + c2;

return c;

}

int Height( NODE p)

{

int h1,h2,h;

if( p== NULL)

return -1;

h1 = Height(p->left\_child);

h2 = Height(p->right\_child);

if(h1 > h2 )

h=h1+1;

else

h=h2+1;

return h;

}

int Max( NODE p)

{

int m1,m2;

if( p== NULL)

return -1;

m1 = Max(p->left\_child);

m2 = Height(p->right\_child);

if(m1 > m2 )

return m1;

else

return m2;

}

int SearchBT( NODE p,int x)

{

int found =0;

if( p== NULL)

return 0;

if( p->data==x)

return 1;

found = SearchBT(p->left\_child,x);

if(!found)

found = Height(p->right\_child,x);

return found;

}

int StrictlyBT( NODE p)

{

int flag;

if( p== NULL)

return 1;

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

return 1;

if( p->left\_child==NULL || p->right\_child==NULL)

return 0;

flag = StrictlyBT(p->left\_child);

if(flag==1)

flag = StrictlyBT(p->right\_child);

return flag;

}

//first call Level(root,x,-1)

int Level( NODE p,int x, int cl)

{

int found =0;

if( p== NULL)

return -1;

if( p->data==x)

return cl+1;

v = Level(p->left\_child,x,cl+1);

if(v==-1)

v = Level(p->right\_child,x,cl+1);

return v;

}

int SearchBST( NODE p,int x)

{

int found =0;

if( p== NULL)

return 0;

if( p->data==x)

return 1;

if(x < p->data)

found = SearchBT(p->left\_child,x);

else

found = Height(p->right\_child,x);

return found;

}

int MaxBST( NODE p)

{

int found =0;

if( p== NULL)

{

printf(" Empty Tree");

return;

}

while( p->right\_child!=NULL)

p=p->right\_child;

return p->data;

}

int InorderSuccessor(NODE p)

{

if( p== NULL)

{

printf(" Empty Tree");

return;

}

p= p->right;

if( p== NULL)

{

printf(" There is No Successor");

return;

}

while( p->left\_child!=NULL)

p=p->left\_child;

return p->data;

}

int InorderPredecessor(NODE p)

{

if( p== NULL)

{

printf(" Empty Tree");

return;

}

p= p->left\_child;

if( p== NULL)

{

printf(" There is No Predecessor");

return;

}

while( p->right\_child!=NULL)

p=p->right\_child;

return p->data;

}

int main()

{

int choice,c;

NODE root=NULL;

printf("Demonstration of BST Insert ,Delete, Display");

while(1)

{

printf("\n1:Create\n2:Preorder Display\n3:Postorder Display \n4:Inorder Display\n5:Number of Nodes\n6:Number of leaf nodes\n Any other number:exit ");

printf("\nEnter the Choice :");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("\nEnter the Root node data");

root=Create();

break;

case 2 :printf("\nTraversal of BT : Preorder");

PreOrder(root);

break;

case 3 :printf("\nTraversal of BT : Postorder");

PostOrder(root);

break;

case 4 :printf("\nTraversal of BT : Postorder");

InOrder(root);

break;

case 5 :c=Count(root);

printf("Number of Nodes in the tree is %d",c);

break;

case 6 :c=CountLeaf(root);

printf("Number of Leaf Nodes in the tree is %d",c);

break;

deafult:exit(0);

} //switch

}//while

return 0;

}//main