#include<stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#define MaxSize 100

typedef struct node {

char data;

struct node\* lchild, \* rchild;

}BinTNode; //define the data type of the node

typedef BinTNode\* BinTree;

BinTree CreatBinTree() {

BinTree T;

char ch;

if ((ch = getchar()) == '\*')

return(NULL); /\*input is ‘\*’，return null vector\*/

else {

T = (BinTree)malloc(sizeof(BinTNode)); /\*create a node\*/

T->data = ch;

T->lchild = CreatBinTree(); /\*create the left subtree\*/

T->rchild = CreatBinTree(); /\*create the right subtree\*/

return T;

}

}

void deleteBt(BinTree\* b) //传入的是Bt节点的地址，将free后的节点指向地址设为NULL，方便确定已经删除干净

{

if (\*b == NULL) return;

deleteBt(&((\*b)->lchild));

deleteBt(&((\*b)->rchild));

free(\*b);

\*b = NULL;

}

int count(BinTree b) //利用递归来找到node 让num++ 最后计算出node总数

{

static int sum = 0;

if (b) {

count(b->lchild);

count(b->rchild);

sum++;

}

return sum;

}

int countleaf(BinTree b) //与count函数类似 补充一个判断是否是leaf node的条件

{

static int sum1 = 0;

if (b) {

countleaf(b->lchild);

countleaf(b->rchild);

if(b->lchild == NULL && b->rchild == NULL) sum1++;

}

return sum1;

}

void PreOrder(BinTree b)

{

BinTNode\* St[MaxSize];

BinTNode\* p;

int top = -1;

bool flag;

if (b != NULL)

{

do

{

while (b != NULL)

{

printf("%c", b->data); //先root 再left 再right

top++;

St[top] = b;

b = b->lchild;

}

p = NULL;

flag = true;

while (top != -1 && flag)

{

b = St[top];

if (b->rchild == p)

{

top--;

p = b;

}

else

{

b = b->rchild;

flag = false;

}

}

} while (top != -1);

printf("\n");

}

}

void InOrder(BinTree b) { //先left 再root 再right

BinTNode\* St[MaxSize];

BinTNode\* p;

int top = -1;

if (b != NULL) {

do {

while (b != NULL) {

top++;

St[top] = b;

b = b->lchild;

}

if (top != -1) {

p = St[top];

top--;

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

b = p->rchild;

}

} while (top != -1 || b != NULL);

printf("\n");

}

}

void PostOrder(BinTree b)//先left 再right 再root

{

BinTNode\* St[MaxSize];

BinTNode\* p;

int top = -1;

bool flag;

if (b != NULL)

{

do

{

while (b != NULL)

{

top++;

St[top] = b;

b = b->lchild;

}

p = NULL;

flag = true;

while (top != -1 && flag)

{

b = St[top];

if(b->rchild == p)

{

printf("% c", b->data);

top--;

p = b;

}

else

{

b = b->rchild;

flag = false;

}

}

} while (top != -1);

printf("\n");

}

}

void DispBinTree(BinTree b)

{

if (b != NULL)

{

printf("%c", b->data);

if (b->lchild != NULL || b->rchild != NULL)

{

printf("(");

DispBinTree(b->lchild);

if (b->rchild != NULL) printf(",");

DispBinTree(b->rchild);

printf(")");

}

}

}

void judge(BinTree b) //判断是否删除干净

{

if (b == NULL) printf("Empty Binary Tree!!\n");

}