#define \_CRT\_SECURE\_NO\_WARNINGS

#define MAXSIZE 100

#include<stdio.h>

#include<Windows.h>

#include<stdlib.h>

//二叉链表

typedef struct Node {

int data;////结点元素

struct Node \*lchild, \*rchild; //左右孩子指针

}\*Bitree, BiTNode;////二叉树结点 ,指向二叉树结点的指针

void CreateBiTree(Bitree \*T);

Bitree Point(Bitree T, char e);

char LeftChild(Bitree T, char e);

char RightChild(Bitree T, char e);

void PreOrderTraverse(Bitree T);

//主函数开始

int main() {

Bitree T;

char ch=0;

//初始化二叉树

//InitBiTree(T);

//生成一颗二叉树

CreateBiTree(&T);

//处理特殊字符

//ch = getchar();

////打印二叉树结点

//printf("请输入要返回的字符名称\n");

//scanf("%c", &ch);

// //空指针尚未处理

//printf("查找的字符的结点的地址为%d\n", Point(T, ch));

////处理特殊字符

//ch = getchar();

//printf("请输入要返回的字符的结点为\n");

//scanf("%c", &ch);

//printf("该节点左孩子的值为%c\n", LeftChild(T, ch));

//printf("该节点右孩子的值为%c\n", RightChild(T, ch));

//先序遍历

printf("先序遍历的顺序是\n");

PreOrderTraverse(T);

while (1)

{

getchar();

}

return 0;

}

//初始化二叉树

void InitBiTree(Bitree \*T)

{

\*T= NULL;

}

//创建一个新的二叉树

void CreateBiTree(Bitree \*T)

{

char ch=0;

scanf("%c",&ch);

////读取到了回车,树建立结束

//if (ch == 13 || ch == 10)

// return;

if (ch =='#')//#表示空树

\*T = NULL;

else {

\*T = (Bitree)malloc(sizeof(BiTNode));//生成一个新的二叉树

if (!(\*T))

{

exit(-1);

}

(\*T)->data = ch;//给二叉树的数据域赋值

CreateBiTree(&((\*T)->lchild));//创建左子树

CreateBiTree(&((\*T)->rchild));//创建右子树

}

}

//销毁二叉树

void DestroyBitTree(Bitree \*T) {

if (\*T) {

if ((\*T)->lchild) {

DestroyBitTree(&((\*T)->lchild));

}

if ((\*T)->rchild) {

DestroyBitTree(&((\*T)->rchild));

}

free(\*T);

\*T = NULL;

}

}

//返回二叉树结点的指针操作

Bitree Point(Bitree T, char e) {

Bitree Q[MAXSIZE];

int font, rear ;

font = rear = 0;

BiTNode \*p;

if (T) {

Q[rear] = T;

rear++;

while (font != rear) {

p = Q[font];

font++;

if (p->data == e)

return p;

if (p->lchild) {

Q[rear] = p->lchild;

rear++;

}

if (p->rchild) {

Q[rear] = p->rchild;

rear++;

}

}

}

return NULL;

}

//返回二叉树的结点的左孩子操作值操作

char LeftChild(Bitree T, char e)

{

Bitree p;

if (T) {

p = Point(T, e);

if (p&&p->lchild)

return p->lchild->data;

}

return 0;

}

//返回二叉树的结点的右孩子操作值操作

char RightChild(Bitree T, char e)

{

Bitree p;

if (T) {

p = Point(T, e);

if (p&&p->rchild)

return p->rchild->data;

}

return 0;

}

/\*

说明:二叉树的左插入操作

参数:指针p指向二叉树T的某个节点

返回值:1,插入成功

0,未插入

\*/

int InsertLeftChild(Bitree p, Bitree c) {

if (p) {

c->rchild = p->lchild;

p->lchild = c;

return 1;

}

return 0;

}

int InsertRightChild(Bitree p, Bitree c) {

if (p) {

c->rchild = p->lchild;

p->rchild = c;

return 1;

}

return 0;

}

//二叉树的左删除操作

int DeleteLeftChild(Bitree p) {

if (p) {

DestroyBitTree(&(p->lchild));

return 1;

}

return 0;

}

//右删除操作

int DeleteRightChild(Bitree p) {

if (p) {

DestroyBitTree(&(p->rchild));

return 1;

}

return 0;

}

//先序遍历二叉树:访问根节点,先序遍历左子树,先序遍历右子数

void PreOrderTraverse(Bitree T) {

if (T) {

printf("%2c", T->data);

PreOrderTraverse(T->lchild);

PreOrderTraverse(T->rchild);

}

}

//中序遍历二叉树:中序遍历左子树,访问根节点,中序遍历右子数

void InOrderTraverse(Bitree T) {

if (T) {

InOrderTraverse(T->lchild);

printf("%2c", T->data);

InOrderTraverse(T->rchild);

}

}

//后序遍历二叉树:后序遍历左子树,右序遍历右子数,访问根节点

void PostTraverse(Bitree T) {

if (T) {

PostTraverse(T->lchild);

PostTraverse(T->rchild);

printf("%2c", T->data);

}

}