一、【实验目的】  
1、掌握二叉树的建立方法

2、掌握二叉树遍历的基本方法（前序、中序、后序）

3、掌握递归二叉树遍历算法的应用

二、【实验内容】

1.构造一棵二叉树,树的形态如下图(亦见附件)所示，打印出先序遍历、中序遍历、后序遍历的遍历序列。

https://www.scholat.com/resources/p_picture/flyi_1683079613018.pnghttps://www.scholat.com/resources/p_picture/flyi_1683079613067.png

                                             A

https://www.scholat.com/resources/p_picture/flyi_1683079613115.pnghttps://www.scholat.com/resources/p_picture/flyi_1683079613147.png

                       B                                             F

https://www.scholat.com/resources/p_picture/flyi_1683079613174.pnghttps://www.scholat.com/resources/p_picture/flyi_1683079613214.png

             C                                         G

       D            E

2.选择一种遍历方式计算该树中叶子结点的个数，并打印出叶子结点。

3.编写一个层序遍历算法，利用队列结构按层次（同一层自左至右）输出二叉树中所有的结点。

三、【实验源代码】

Tree.h

#define OK 1

#define ERROR 0

//tree结点

typedef struct treecode{

    int data;

    struct treecode \*right,\*left;

}treecode;

#define MAXSIZE 50  //定义队列中元素的最大个数

typedef struct{

    int data[MAXSIZE];  //存放队列元素

    int front,rear;

}SqQueue;//队列实现层次

/\*初始化一个空队列Q\*/

int InitQueue(SqQueue \*Q){

    Q->front = 0;

    Q->rear = 0;

    return OK;

}

/\*判队空\*/

int isEmpty(SqQueue Q){

    if(Q.rear == Q.front){

        return 1;

    }else{

        return 0;

    }

}

/\*若队列不空，则删除Q中队头元素，用e返回其值\*/

int DeQueue(SqQueue \*Q, ElemType \*e){

    if(isEmpty(Q)){

        return REEOR;   //队列空的判断

    }

    \*e = Q->data[Q->front]; //将队头元素赋值给e

    Q->front = (Q->front + 1) % MAXSIZE;    //front指针向后移一位置，若到最后则转到数组头部

}

/\*若队列未满，则插入元素e为Q新的队尾元素\*/

int EnQueue(SqQueue \*Q, ElemType e){

    if((Q->rear + 1) % MAXSIZE == Q->front){

        return ERROR;   //队满

    }

    Q->data[Q->rear] = e;   //将元素e赋值给队尾

    Q->rear = (Q->rear + 1) % MAXSIZE;  //rear指针向后移一位置，若到最后则转到数组头部

    return OK;

}

/\*若队列不空，则删除Q中队头元素，用e返回其值\*/

int DeQueue(SqQueue \*Q, ElemType \*e){

    if(isEmpty(Q)){

        return ERROR;   //队列空的判断

    }

    \*e = Q->data[Q->front]; //将队头元素赋值给e

    Q->front = (Q->front + 1) % MAXSIZE;    //front指针向后移一位置，若到最后则转到数组头部

}

//建立 ，队列实现 ,参数是创建多少个结点

 treecode \*createtree(void){

    treecode \*root;

    char ch;//输入的值

    scanf("%c",&ch);

    if(ch == '#')root == NULL;

    else

    {

        root = (treecode\*)malloc(sizeof(treecode));

        if(root == NULL)

        {

            printf("out of space\n");

            return root;

        }

        root->data = ch;

        root->left = createtree();

        root->right = createtree();

     }

     return root;

 }

//先序

void preorder(treecode \*a){

    if(a!=NULL){

        printf("%d\n",a->data);

        preorder(a->left);

        preorder(a->right);

    }

}

//中序

void midorder(treecode \*a){

    if(a!=NULL){

        midorder(a->left);

        printf("%d\n",a->data);

        midorder(a->right);

    }

}

//后序

void afterrder(treecode \*a){

    if(a!=NULL){

        afterorder(a->left);

        afterorder(a->right);

        printf("%d\n",a->data);

    }

}

void visit(treecode \*p){

    printf("%d\n",p->data);

}

void LevelOrder(treecode T){

    InitQueue(Q);   //初始化辅助队列

    treecode \*p;

    EnQueue(Q, T);  //将根节点入队

    while(!IsEmpty(Q)){ //队列不空则循环

        DeQueue(Q, p);  //队头结点出队

        visit(p);   //访问出队结点

        if(p->left != NULL){

            EnQueue(Q, p->left);    //左子树不空，则左子树根节点入队

        }

        if(p->right != NULL){

            EnQueue(Q, p->right);   //右子树不空，则右子树根节点入队

        }

    }

}

}

// 1.构造一棵二叉树,树的形态如下图(亦见附件)所示，打印出先序遍历、中序遍历、后序遍历的遍历序列

Tree.c

#include<stdio.h>

#include<malloc.h>

#include"tree.h"

void main(){

    printf("create your tree\n the tree's null tree can replace '#'\n");

    treecode \*a = createtree();

    //输入是abcde#fg#

    printf("先序\n");

    preorder(a);

    printf("中序\n");

    midorder(a);

    printf("后序\n");

    afterorder(a);

    LevelOrder(\*a);

}

四、【实验结果】

无法成功

五、【实验心得