实验三　树 和 二 叉 树

一、实验目的

1. 掌握二叉树的结构特征，以及各种存储结构的特点及适用范围。
2. 掌握用指针类型描述、访问和处理二叉树的运算。

二、实验要求

1. 认真阅读和掌握本实验的程序。
2. 上机运行本程序。
3. 保存和打印出程序的运行结果，并结合程序进行分析。
4. 按照二叉树的操作需要，重新改写主程序并运行，打印出文件清单和运行结果。

三、实验内容

1. 输入字符序列，建立二叉链表。
2. 按先序、中序和后序遍历二叉树（递归算法）。
3. 按某种形式输出整棵二叉树。
4. 求二叉树的高度。
5. 求二叉树的叶节点个数。
6. 交换二叉树的左右子树。
7. 借助队列实现二叉树的层次遍历。
8. 在主函数中设计一个简单的菜单，分别调试上述算法。

为了实现对二叉树的有关操作，首先要在计算机中建立所需的二叉树。建立二叉树有各种不同的方法。一种方法是利用二叉树的性质 5 来建立二叉树，输入数据时要将节点的序号（按满二叉树编号）和数据同时给出：（序号，数据元素 0）。另一种方法是主教材中介绍的方法，这是一个递归方法，与先序遍历有点相似。数据的组织是先序的顺序，但是另有特点，当某结点的某孩子为空时以字符“#”来充当，也要输入。若当前数据不为“#”，则申请一个结点存入当前数据。递归调用建立函数，建立当前结点的左右子树。

四、解题思路

1、先序遍历：①访问根结点，②先序遍历左子树，③先序遍历右子树

2、中序遍历：①中序遍历左子树，②访问根结点，③中序遍历右子树

3、后序遍历：①后序遍历左子树，②后序遍历右子树，③访问根结点

4、层次遍历算法：采用一个队列 q，先将二叉树根结点入队列，然后退队列，输出该结点；若它有左子树，便将左子树根结点入队列；若它有右子树，便将右子树根结点入队列，直到队列空为止。因为队列的特点是先进后出，所以能够达到按层次遍历二叉树的目的。

五、程序清单

```
#include<stdio.h>
#include<stdlib.h>
#define M 100
typedef char Etype;                          //定义二叉树结点值的类型为字符型
typedef struct BiTNode                       //树结点结构
```

```
{
    Etype data;
    struct BiTNode *lch,*rch;
}BiTNode,*BiTree;
BiTree que[M];
int front=0,rear=0;
//函数原型声明
BiTNode *creat_bt1();
BiTNode *creat_bt2();
void preorder(BiTNode *p);
void inorder(BiTNode *p);
void postorder(BiTNode *p);
void enqueue(BiTree);
BiTree delqueue( );
void levorder(BiTree);
int treedepth(BiTree);
void prtbtree(BiTree,int);
void exchange(BiTree);
int leafcount(BiTree);
void paintleaf(BiTree);
BiTNode *t;
int count=0;
//主函数
void main()
{
    char ch;
    int k;
    do{
        printf("\n\n\n");
        printf("\n==================主菜单==================");
        printf("\n\n     1.建立二叉树方法 1");
        printf("\n\n     2.建立二叉树方法 2");
        printf("\n\n     3.先序递归遍历二叉树");
        printf("\n\n     4.中序递归遍历二叉树");
        printf("\n\n     5.后序递归遍历二叉树");
        printf("\n\n     6.层次遍历二叉树");
        printf("\n\n     7.计算二叉树的高度");
        printf("\n\n     8.计算二叉树中叶结点个数");
        printf("\n\n     9.交换二叉树的左右子树");
        printf("\n\n     10.打印二叉树");
        printf("\n\n     0.结束程序运行");
        printf("\n==========================================");
        printf("\n     请输入您的选择(0,1,2,3,4,5,6,7,8,9,10)");
        scanf("%d",&k);
```

```
switch(k)
{
    case 1:t=creat_bt1( );break;                    //调用性质 5 建立二叉树算法
    case 2:printf("\n 请输入二叉树各结点值:");fflush(stdin);
    t=creat_bt2();break;                            //调用递归建立二叉树算法
    case 3:if(t)
            {printf("先序遍历二叉树:");
            preorder(t);
            printf("\n");
            }
            else printf("二叉树为空!\n");
            break;
    case 4:if(t)
            {printf("中序遍历二叉树:");
            inorder(t);
            printf("\n");
            }
            else printf("二叉树为空!\n");
            break;
    case 5:if(t)
            {printf("后序遍历二叉树:");
            postorder(t);
            printf("\n");
            }
            else printf("二叉树为空!\n");
            break;
    case 6:if(t)
            {printf("层次遍历二叉树：");
             levorder(t);
             printf("\n");
            }
            else printf("二叉树为空！\n");
            break;
    case 7:if(t)
            {printf("二叉树的高度为：%d",treedepth(t));
             printf("\n");
            }
            else printf("二叉树为空！\n");
            break;
    case 8:if(t)
            {printf("二叉树的叶子结点数为：%d\n",leafcount(t));
             printf("二叉树的叶结点为：");paintleaf(t);
             printf("\n");
            }
```

```
                    else printf("二叉树为空！\n");
                    break;
            case 9:if(t)
                    {printf("交换二叉树的左右子树：\n");
                     exchange(t);
                     prtbtree(t,0);
                     printf("\n");
                    }
                    else printf("二叉树为空！\n");
                    break;
            case 10:if(t)
                     {printf("逆时针旋转 90 度输出的二叉树：\n");
                      prtbtree(t,0);
                      printf("\n");
                     }
                     else printf("二叉树为空！\n");
                     break;
            case 0:exit(0);
            }     //switch
    }while(k>=1&&k<=10);
        printf("\n 再见！按回车键，返回…\n");
        ch=getchar();
}    //main

//利用二叉树性质 5，借助一维数组 V 建立二叉树
BiTNode *creat_bt1()
{ BiTNode *t,*p,*v[20];int i,j;Etype e;
/*输入结点的序号 i、结点的数据 e*/
printf("\n 请输入二叉树各结点的编号和对应的值（如 1，a）：");
scanf("%d,%c",&i,&e);
while(i!=0&&e!='#')                                    //当 i 为 0，e 为'#'时，结束循环
{
    p=(BiTNode*)malloc(sizeof(BiTNode));
    p->data=e;
    p->lch=NULL;
    p->rch=NULL;
    v[i]=p;
    if(i==1)
    t=p;    //序号为 1 的结点是根
    else
    {
        j=i/2;
        if(i%2==0)v[j]->lch=p;                         //序号为偶数，作为左孩子
        else   v[j]->rch=p;                            //序号为奇数，作为右孩子
```

```
        }
        printf("\n 请继续输入二叉树各结点的编号和对应的值：");
        scanf("%d,%c",&i,&e);
}
return(t);
}//creat_bt1;
//模仿先序递归遍历方法，建立二叉树
BiTNode *creat_bt2()
{
        BiTNode *t;
        Etype e;
        scanf("%c",&e);
        if(e=='#')t=NULL;     //对于'#'值，不分配新结点
        else{
                t=(BiTNode *)malloc(sizeof(BiTNode));
                t->data=e;
                t->lch=creat_bt2();                                    //左孩子获得新指针值
                t->rch=creat_bt2();                                    //右孩子获得新指针值
              }
return(t);
}     //creat_bt2
//先序递归遍历二叉树
void preorder(BiTNode *p)
{if(p){
            printf("%3c",p->data);
            preorder(p->lch);
            preorder(p->rch);
        }
}       //preorder
//中序递归遍历二叉树
void inorder(BiTNode *p)
{if(p){
            inorder(p->lch);
            printf("%3c",p->data);
            inorder(p->rch);
        }
} //inorder
//后序递归遍历二叉树
void postorder(BiTNode *p)
{ if(p){ postorder(p->lch);
             postorder(p->rch);
             printf("%3c",p->data);
         }
} //postorder
```

```
void enqueue(BiTree T)
{
      if(front!=(rear+1)%M)
      {rear=(rear+1)%M;
      que[rear]=T;}
}
BiTree delqueue( )
{
      if(front==rear)return NULL;
      front=(front+1)%M;
      return(que[front]);
}
void levorder(BiTree T)                                          //层次遍历二叉树
{
      BiTree p;
      if(T)
      {enqueue(T);
          while(front!=rear){
            p=delqueue(    );
            printf("%3d",p->data);
            if(p->lch!=NULL)enqueue(p->lch);
            if(p->rch!=NULL)enqueue(p->rch);
            }
      }
}
int treedepth(BiTree bt)                                         //计算二叉树的高度
{
      int hl,hr,max;
      if(bt!=NULL)
      {    hl=treedepth(bt->lch);
          hr=treedepth(bt->rch);
          max=(hl>hr)?hl:hr;
          return (max+1);
      }
      else return (0);
}

void prtbtree(BiTree bt,int level)                               //逆时针旋转 90 度输出二叉树树形
{int j;
if(bt)
{prtbtree(bt->rch,level+1);
for(j=0;j<=6*level;j++)printf(" ");
printf("%c\n",bt->data);
```

```
prtbtree(bt->lch,level+1);
}
}

void exchange(BiTree bt)    //交换二叉树左右子树
{BiTree p;
if(bt)
{p=bt->lch;bt->lch=bt->rch;bt->rch=p;
exchange(bt->lch);exchange(bt->rch);
}
}

int leafcount(BiTree bt)    //计算叶结点数
{
if(bt!=NULL)
{leafcount(bt->lch);
leafcount(bt->rch);
if((bt->lch==NULL)&&(bt->rch==NULL))
      count++;
}
return(count);
}

void paintleaf(BiTree bt)     //输出叶结点
{if(bt!=NULL)
      {if(bt->lch==NULL&&bt->rch==NULL)
            printf("%3c",bt->data);
      paintleaf(bt->lch);

 paintleaf(bt->rch);
      }
}
```

图 11.2 所示二叉树的输入数据顺序应该是：abd#g###ce#h##f##。

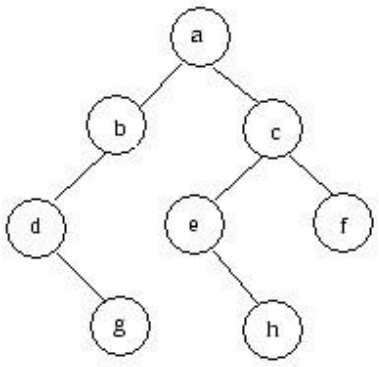

图 11.2　二叉树示意图

运行结果：

==========主菜单==========

1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行

==========

请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 1
请输入二叉树各结点的编号和对应的值（如 1，a）：1，a
请继续输入二叉树各结点的编号和对应的值：2，b
请继续输入二叉树各结点的编号和对应的值：3，c
请继续输入二叉树各结点的编号和对应的值：4，d
请继续输入二叉树各结点的编号和对应的值：6，e
请继续输入二叉树各结点的编号和对应的值：7，f
请继续输入二叉树各结点的编号和对应的值：9，g
请继续输入二叉树各结点的编号和对应的值：13，h
请继续输入二叉树各结点的编号和对应的值：0，#

==========主菜单==========");

1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行

==========

请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 3
先序遍历二叉树：a b d g c e h f

==========主菜单==========");
1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行
==========
请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 4
中序遍历二叉树：d g b a e h c f

==========主菜单==========");
1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行
==========
请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 5
后序遍历二叉树：g d b h e f c a

==========主菜单==========");
1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树

10.打印二叉树
0.结束程序运行
==========
请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 6
层次遍历二叉树： 97 98 99100101102103104

==========主菜单==========");
1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行
==========
请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 7
二叉树的高度为：4

==========主菜单==========");
1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行
==========
请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 8
二叉树的叶子结点数为：3
二叉树的叶结点为： g h f

==========主菜单==========");
1.建立二叉树方法 1
2.建立二叉树方法 2

3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行
==========
请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 9
交换二叉树的左右子树：
d
g
b
a
e
h
c
f

==========主菜单==========");
1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行
==========
请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 10
逆时针旋转 90 度输出的二叉树：
d
g
b
a
e
h
c
f

==========主菜单==========");
1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行
==========
请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 2
请输入二叉树各结点值：abd#g###ce#h##f##

==========主菜单==========");
1.建立二叉树方法 1
2.建立二叉树方法 2
3.先序递归遍历二叉树
4.中序递归遍历二叉树
5.后序递归遍历二叉树
6.层次遍历二叉树
7.计算二叉树的高度
8.计算二叉树中叶结点个数
9.交换二叉树的左右子树
10.打印二叉树
0.结束程序运行
==========
请输入您的选择(0,1,2,3,4,5,6,7,8,9,10) 0
请按任意键继续. . .

六、调试心得及收获