

《数据结构》上机报告

2018 年 11 月 2 日

姓名： 赵得泽 学号： 1753642 班级： 电子 2 班 得分： _____

实验题目	二叉树	
问题描述	二叉树是 n ($n \geq 0$) 个结点的有限集合，它或为空树，或是由一个称之为根的结点加上两棵分别称为左子树和右子树的互不相交的二叉树组成。	
基本要求	1. 本题练习二叉树的基本操作，包括先序遍历建立二叉树、先序遍历、中序遍历、后序遍历、层次遍历、输出二叉树。 2. 本题练习用递归程序对二叉树进行计算，包括二叉树的叶子结点数，总结点数，高度，复制二叉树（左右互换）。 3. 本题练习用非递归完成二叉树的中序遍历。	
	已完成基本内容（序号）：	1, 2, 3
选做要求		
	已完成选做内容（序号）	
数据结构设计	<pre> typedef struct Node { TElemType Data; Node *lchild; Node *rchild; }*BiTree; typedef struct { BiTree *base; //存放动态申请空间的首地址（栈底） BiTree *top; //栈顶指针 int stacksize; //当前分配的元素个数 } SqStack; struct SqQueue { BiTree *base; int front, rear; }; </pre> <p>本次实验用到的数据结构主要是二叉树，二叉树是 n ($n \geq 0$) 个结点的集合，它或为空树，或是由一个称之为根的结点加上两棵分别称为左子树和右子树的互不相交的二叉树组成。它的指针域包括*lchild和*rchild,这两个指针域分别指向一个结点的左右孩子结点，还有结点存储的数据 data,这是结点要存储的数据。这样其实也就构成了一个二叉链表。在非递归访问二叉树的数据及层次遍历时还用到的数据结构是顺序栈，顺序队列，这两种数据结构是线性结构，和二叉树一起用的好处就是将二叉树上面存储的数据都存到一个有限的连续空间中，使其线性化，也便于对数据访问。</p>	

<p>功能(函数)说明</p>	<pre> /***** 函数功能：二叉树的初始化 说明：二叉树的初始化即将BiTree型的数据T置空，即建立空树。 *****/ void IniBiTree(BiTree &T) { T = NULL; } /***** 函数功能：二叉树的销毁 说明：利用递归将二叉树从根节点开始对每个节点的左右孩子进行置空； *****/ void DestroyBiTree(BiTree &T) { if (T) { if (T->lchild) DestroyBiTree(T->lchild); if (T->rchild) DestroyBiTree(T->rchild); delete T; T = NULL; } } /***** 函数功能：二叉树的建立 说明：此种建立方法是利用先序递归建立二叉树，遇到'#'意味着该结点为空结点， 将其置空，否则T的数据域为输入的字符； *****/ void CreatBiTree(BiTree &T) { TElemType c; cin >> c; if (c == '#') T = NULL; else { T = new Node; if (!T) exit(-1); T->Data = c; CreatBiTree(T->lchild); CreatBiTree(T->rchild); } } </pre>
-----------------	---

```

    }
}

/*****
函数功能：先序遍历二叉树
说明：利用递归，先遍历根节点，然后左孩子、右孩子；即每次都优先访问双亲
节点的数据，然后左、右孩子的数据。
*****/
void PreOrderTraverse(BiTree &T)
{
    if (!T)
        return;
    else
    {
        cout << T->Data;
        PreOrderTraverse(T->lchild);
        PreOrderTraverse(T->rchild);
    }
}

/*****
函数功能：中序遍历二叉树
说明：利用递归，中序遍历左子树；访问根节点；中序遍历右子树；
*****/
void InOrderTraverse(BiTree &T)
{
    if (T)
    {
        InOrderTraverse(T->lchild);
        cout << T->Data;
        InOrderTraverse(T->rchild);
    }
}

/*****
函数功能：后序遍历二叉树
说明：利用递归，后序遍历左子树；后序遍历右子树；访问根节点；
*****/
void PostOrderTraverse(BiTree &T)
{
    if (T)
    {
        PostOrderTraverse(T->lchild);
        PostOrderTraverse(T->rchild);
        cout << T->Data;
    }
}

/*****

```

函数功能：层次遍历二叉树

说明：在此要新增一个队列Q用来存储每次遍历访问的结点，若结点不空，首先将根结点的数据存到队尾（队首），队尾指针后移，再利用循环，先访问该结点数据，队首指针后移，再访问该结点的左右孩子结点每访问一个孩子结点（若孩子结点不空），则将该节点数据入队，队尾指针后移，继续此循环，直到队尾指针和队首指针相遇，结束循环。完成层次遍历。

*****/

```
void LevelOrderTraverse(BiTree &T)
{
    BiTree p;
    SqQueue Q;
    Q.base = new BiTree[MAXSIZE];
    if (!Q.base) exit(-1);
    Q.front = Q.rear = 0;
    if (T)
    {
        Q.base[Q.rear] = T;
        Q.rear = (Q.rear + 1) % MAXSIZE;
        while (Q.front != Q.rear)
        {
            p = Q.base[Q.front];
            cout << p->Data;
            Q.front = (Q.front + 1) % MAXSIZE;
            if (p->lchild)
            {
                Q.base[Q.rear] = (p->lchild);
                Q.rear = (Q.rear + 1) % MAXSIZE;
            }
            if (p->rchild)
            {
                Q.base[Q.rear] = (p->rchild);
                Q.rear = (Q.rear + 1) % MAXSIZE;
            }
        }
    }
}
```

函数功能：输出二叉树树形

说明：由题目给出的树形分析得知，每行的元素正好是中序遍历得到元素的逆序排列由此可以将中序遍历的递归方法中的左右孩子访问次序对换，那么输出的数据次序就是树形输出次序，空格个数是该结点的深度，所以在递归中应该加一个Depth来计算每个结点的深度，实际上递归的层次数就是该结点所在的深度，这样每次的访问不管是进入递归还是退出递归都会对应一个相应的Depth，那样就很容易记录每个结点的深度，从而输出树形。

```

/*****/
void ShapeBiTree(BiTree &T, int Depth)
{
    if (T)
    {
        ShapeBiTree(T->rchild, Depth + 1);
        for (int i = 1; i < Depth; i++)
            cout << "    ";
        cout << T->Data;
        cout << endl;
        ShapeBiTree(T->lchild, Depth + 1);
    }
}

```

/*****/

函数功能：复制二叉树

说明：利用先序遍历的方法递归复制二叉树的每个结点。将其复制到新的树S中。

```

/*****/
void CopyTree(BiTree &T, BiTree &S)
{
    if (T)
    {
        S = new Node;
        if (!S)
            exit(-1);
        S->lchild = S->rchild = NULL;
        S->Data = T->Data;
        CopyTree(T->lchild, S->lchild);
        CopyTree(T->rchild, S->rchild);
    }
}

```

/*****/

函数功能：交换二叉树的左右子树

说明：设置一个临时变量

S = T->lchild;T->lchild = T->rchild;T->rchild = S;将左右孩子节点进行交换，然后在利用先序递归，那么就将所有结点的左右孩子都进行了交换。

```

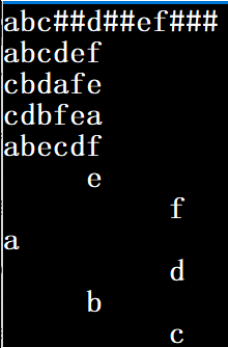
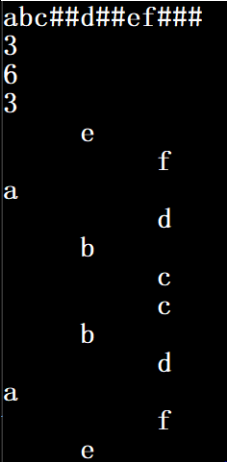
/*****/
void Exchange(BiTree &T)
{
    BiTree S;
    if (T)
    {
        S = T->lchild;
        T->lchild = T->rchild;

```

```

        T->rchild = S;
        Exchange(T->lchild);
        Exchange(T->rchild);
    }
}
/*****
函数功能：计算二叉树
说明：利用先序递归遍历二叉树的方法，首先访问根结点，总结点个数加一，如果该结点的左右孩子都为空，则叶子结点个数加一，并且比较当前叶子结点的深度和树当前深度的大小，将大者赋给树的深度，之后运行递归函数，访问左右孩子，递归结束之后就可以得出总结点个数NumNode，叶子结点个数NumLeave，以及树的深度Tdepth.
(PS：叶子结点的深度计算在上面输出树形时已做过详细说明)
*****/
void CalCulBiTree(BiTree &T, int Depth)
{
    if (T)
    {
        NumNode++;
        if (!T->lchild && !T->rchild)
        {
            NumLeave++;
            Tdepth = max(Tdepth, Depth);
        }
        CalCulBiTree(T->lchild, Depth + 1);
        CalCulBiTree(T->rchild, Depth + 1);
    }
}
/*****
函数功能：非递归中序遍历二叉树
说明：在非递归过程中要用到栈，再次定义一个栈S，从根结点对树进行循环遍历，如果节点不空，将该结点压栈，输出结点数据，再访问该结点左孩子，否则，将栈顶元素出栈，输出结点的数据，再访问结点的右孩子。直到遇到空结点或者栈为空，结束循环。
*****/
void InorderTraverse(BiTree &T)
{
    SqStack S;
    BiTree p = T;
    InitStack(S);
    while (p || !StackEmpty(S))
    {
        if (p)
        {

```

	<pre>Push(S, p); //p相当于某个结点 cout << "push " << p->Data << endl; p = p->lchild; } else { Pop(S, p); cout << "pop" << endl; cout << p->Data << endl; p = p->rchild; } }</pre>
开发环境	Win10, vs2017, C++高级程序设计
调试分析	<p>1.</p>  <p>2.</p>  <p>3.</p>

	<pre> abc##d##ef### push a push b push c pop c pop b push d pop d pop a push e push f pop f pop e </pre>
心得体会	<p>本次实验涉及到的主要是树的操作，包括中、先、后序的递归遍历，创建、复制树，树的深度，总结点、叶子结点个数的计算，还有树形的输出。</p> <p>总的来说，树的操作主要是依靠递归进行解决，但是在构建递归过程中，总免不了会对递归退出条件产生迷惑。所以在用递归解题时，对递归退出条件的准确把握是非常重要的，每层递归所要解决的问题也要清楚明了。还有，在解决与树有关的问题时，我们要对树的结构有一个清楚的认识，对每种遍历方法的特点要准确掌握，这样才能运用自如。</p> <p>比如，在树形输出的时候，每层之间隔 5 个空格，旋转 90 度输出，在接到这个问题的时候，我第一个意识到的就是，它树形输出的顺序，以及输出这个树形的前提。在经过探索之后，发现它和中序输出的顺序正好相反，而且每个结点数据之间空格的个数正好是它所在的深度，接下来的主要目的明确，就是颠倒中序输出顺序，求解每个结点的深度。这样就得出了 ShapeBiTree 函数 (具体方法见上函数功能说明)，依旧是利用递归思想，在求解树的深度的时候，同样也是这样的思路，同叶子结点的深度比较即可得出树的深度。</p> <p>同时还要对递归算法进行演练，即自己制造数据进行推演递归算法，直到完全弄清其中递归的进入及退出，这样也许会对递归加深理解，并且设计算法也相对更有思路。</p>