作业3

二叉树存储结构的建立、遍历和应用

树型结构的遍历是树型结构算法的基础，本实验要求编写程序演示二叉树的存储结构的建立方法、遍历过程以及应用。

存储结构

```
struct TreeNode
{
    char data;
    bool vis;
    TreeNode * LCh;
    TreeNode * RCh;
    TreeNode()
    {
        this->LCh = NULL;
        this->RCh = NULL;
    }
};
typedef TreeNode * Tree;
typedef TreeNode * Node;
```

函数说明

1. 编写建立二叉树的二叉链表存储结构（左右链表示）的程序，并以适当的形式显示和保存二叉树；

`Tree InputTree()` 按照**先序序列**建立二叉树。

`void PrintTree(Tree &Root)` 打印根为 `Root` 的树，共 n 行，**每行输出该节点、该节点左儿子、该节点右儿子，若无儿子则输出** `#` 。

2. 采用二叉树的二叉链表存储结构，编写程序实现二叉树的先序、中序和后序遍历的递归和非递归算法以及层序遍历算法，并以适当的形式显示和保存二叉树及其相应的遍历序列；

`void Get_PreOrder_WithRecursion(Tree &Root)` 用递归实现输出**先**序遍历序列。

`void Get_InOrder_WithRecursion(Tree &Root)` 用递归实现输出**中**序遍历序列。

`void Get_PostOrder_WithRecursion(Tree &Root)` 用递归实现输出**后**序遍历序列。

`void Get_PreOrder_WithoutRecursion(Tree &Root)` 用非递归实现输出**先**序遍历序列。

`void Get_InOrder_WithoutRecursion(Tree &Root)` 用非递归实现输出**中**序遍历序列。

`void Get_PostOrder_WithoutRecursion(Tree &Root)` 用非递归实现输出**后**序遍历序列。

`void Get_LeverOrder(Tree &Root)` 输出层序遍历序列。

3. 设计并实现判断任意一棵二叉树是否为完全二叉树的算法。

`bool Is_CompleteTree(Tree &Root)` 判断根为 `Root` 的树是否为完全二叉树。

4. 设计并实现计算任意一棵二叉树的宽度的（递归或非递归）算法。二叉树的宽度是指其各层结点数的最大值。

`int Get_MaxWidth(Tree &Root)` 输出根为 `Root` 的树的宽度。

自测

测试说明

由于本程序未加入`system("pause")`，建议在CMD/Terminal中测试。

本程序采用标准输入输出。

为方便测试，源程序 `Homework3.cpp` 中**将测试内容写入主函数**。

另提供样例测试数据输入 `Homework3_In_1.txt` ，和期望输出 `Homework3_Out_1.txt` 。

另提供样例测试数据输入 `Homework3_In_2.txt` ，和期望输出 `Homework3_Out_2.txt` 。

输入格式说明

输入 1 行，二叉树的先序序列。

输出格式说明

1. 输出二叉树，共 n 行，每行输出该节点、该节点左儿子、该节点右儿子，若无儿子则输出 `#` 。
2. 分别用递归和非递归实现输出**先**序遍历序列，共 2 个序列，期望这 2 个序列是相同的。
3. 分别用递归和非递归实现输出**中**序遍历序列，共 2 个序列，期望这 2 个序列是相同的。
4. 分别用递归和非递归实现输出**后**序遍历序列，共 2 个序列，期望这 2 个序列是相同的。
5. 输出层序遍历序列，只有 1 个序列。
6. 判断树是否为完全二叉树，若是，则输出 `YES` ；否则输出 `NO` 。
7. 输出树的宽度， 1 个整数。

数据说明

#1

Homework3_In_1.txt

```
ABDH##I##E##CF#J##G##
```

Homework3_Out_1.txt

```
Your Tree:                              二叉树，每行输出该节点、该节点左儿子、该节点右儿子，若无儿子则输出#
ABC
BDE
DHI
H##
I##
E##
```

```
CFG
F#J
J##
G##
PreOrder_WithRecursion:                用递归输出先序遍历序列
A B D H I E C F J G
PreOrder_WithoutRecursion:             用非递归输出先序遍历序列
A B D H I E C F J G

InOrder_WithRecursion:                 用递归输出中序遍历序列
H D I B E A F J C G
InOrder_WithoutRecursion:              用非递归输出中序遍历序列
H D I B E A F J C G

PostOrder_WithRecursion:               用递归输出后序遍历序列
H I D E B J F G C A
PostOrder_WithoutRecursion:            用非递归输出后序遍历序列
H I D E B J F G C A

LeverOrder:                            输出层序遍历序列
A B C D E F G H I J

Complete_Tree?                         是否是完全二叉树
NO
Max_Width:                             树的宽度
4
```

#2

Homework3_In_2.txt

```
ABCD##E##FG##H##IJK##L##MN##O##
```

Homework3_Out_2.txt

```
Your Tree:                             二叉树，每行输出该节点、该节点左儿子、该节点右儿子，若无儿子则输出#
ABI
BCF
CDE
D##
E##
FGH
G##
H##
IJM
JKL
K##
L##
MNO
N##
O##
PreOrder_WithRecursion:                用递归输出先序遍历序列
A B C D E F G H I J K L M N O
PreOrder_WithoutRecursion:             用非递归输出先序遍历序列
```

```
A B C D E F G H I J K L M N 0

InOrder_WithRecursion:                  用递归输出中序遍历序列
D C E B G F H A K J L I N M 0
InOrder_WithoutRecursion:               用非递归输出中序遍历序列
D C E B G F H A K J L I N M 0

PostOrder_WithRecursion:                用递归输出后序遍历序列
D E C G H F B K L J N 0 M I A
PostOrder_WithoutRecursion:             用非递归输出后序遍历序列
D E C G H F B K L J N 0 M I A

LeverOrder:                             输出层序遍历序列
A B I C F J M D E G H K L N 0

Complete_Tree?                          是否是完全二叉树
YES
Max_Width:                              树的宽度
8
```