一、 **选择题**

1、 一棵完全二叉树上有 1001 个结点，其中叶子结点的个数是（C）。

A．250　　B． 500　　C．254　　D．501

2、 一个具有 1025 个结点的二叉树的高 h 为（C）。

A．11　　B．10

C．11 至 1025 之间　　D．10 至 1024 之间

3、 某二叉树 T 有 n 个结点，设按某种顺序对 T 中的每个结点进行编号，编号值为 1,2……n。如果有如下性质：T 中任意结点 v，其编号等于左子树上的最小编号减 1，而 v 的右子树的结点中，其最小编号等于 v 左子树上结点的最大编号加 1，则这是按（B）编号的。

A. 中序遍历序列　　B. 先序遍历序列

C. 后序遍历序列　　D. 层次遍历序列

4、 二叉树的叶子结点在先序、中序、后序遍历序列中的相对次序（A）

A. 不发生改变　　B. 发生改变　　C. 不确定

5、 若二叉树采用二叉链表存储结构，要交换其所有分支结点左、右子树的位置，利用（D）遍历方法最合适。

A．前序　　B．中序　　C．后序

二、 **简答题：**

试找出满足下列条件的二叉树。

1、先序序列与后序序列相同。

2、中序序列与后序序列相同

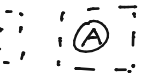
3、先序序列与中序序列相同

4、中序序列与层次遍历序列相同。

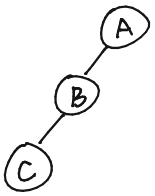
5、先序序列与后序序列正好相反

提交要求：除了空树及只有一个根结点的二叉树外，找找层数大于 1 的满足条件的二叉树，每个小题画出一个满足条件的二叉树，并写出两个遍历序列验证。

1、先序=后序：由于先序遍历第一个结点一定是根，而后序的最后一个结点才是根，故只有空树或单结点树满足。



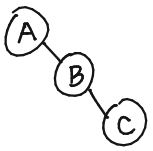

2．中序=后序：即每一个结点都没有右子树.




中：CBA

后：CBA

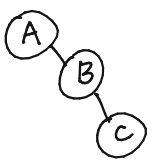
3．先序=中序：每一个结点都没有左子树




先：ABC

中：ABC

4、中序=层次：同3．没有右子树.

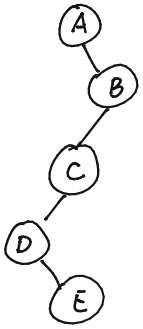



中：ABC

层：ABC

5、先序刚好与后序相反：

每层只有一个结点.




先序：ABCDE

后序：EDCBA

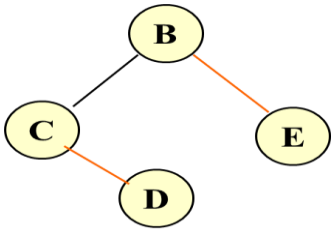
## 三、 编程实现的二叉链式存储方式的二叉树的数据结构方法

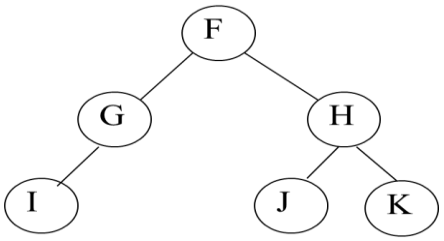
```
//2. 销毁二叉链式存储的二叉树，释放每个结点所分配的内存
//    注意是没有头结点的二叉树
bool DestroyBinTree(BiTree bt);

//3.1 从先序遍历的字符串中创建二叉树，将创建的二叉树用返回值返回到调用者
//     先序遍历的字符串中用字符'!'表示空结点
//参考教材 P131 实现，这里的参数 nStringStart，表示创建树时
//，从 preordString 的那个字符开始读入
//使用引用形式的参数传递方式进行参数传递，这样可以在递归调用时，
// 每读入一个字符后，
// nStringStart 向后移动一个位置，并将这个值的改变带回到调用者
BiTree CreateBinTree(char * preordString, int & nStringStart);

//4.1 先序遍历二叉树，将结果输出到控制台（stdc::out）
//    返回值：空树返回 false，非空树返回 true
bool PreOrderTraverse(BiTree bt);
//4.2 中序遍历二叉树，将结果输出到控制台（stdc::out）
//    返回值：空树返回 false，非空树返回 true
bool InOrderTraverse(BiTree bt);
//4.3 后序遍历二叉树，将结果输出到控制台（stdc::out）
//    返回值：空树返回 false，非空树返回 true
bool PostOrderTraverse(BiTree bt);
//4.4 层序遍历二叉树，将结果输出到控制台（stdc::out）
//    返回值：空树返回 false，非空树返回 true
bool LevelOrderTraverse(BiTree bt);

//5 计算树的结点数目
int BiTreeNodeCount(BiTree bt);
//6 计算树的叶子结点数目
int BiTreeLeavesCount(BiTree bt);
//7 计算树的层数
int BiTreeLevelCount(BiTree bt);
```

在 06BinTree.cpp 中补充所缺的代码来完成作业：

i. 声明了所有的函数；给出了所需实现函数的空函数框架，补充每个函数的实现代码。

ii. 在 main 函数里面有两个字符串变量 sTree1 和 sTree2，预先保存了两棵树的先序遍历结果供读入使用，两个字符串对应下图的两棵树




测试时，可用其他树的遍历结果替换字符串换不同的树进行测试。

iii. 请仔细阅读 main 函数，理解 main 函数调用的各个基本操作。

iv. 将程序调试正确

v. 程序运行结果参考

```
从先序遍历结果  BC!D!!E!!  创建第一棵二叉树

第一棵树的先序遍历：
BCDE
第一棵树的中序遍历：
CDBE
第一棵树的后序遍历：
DCEB
第一棵树的层序遍历：
BCED
第一棵树的结点数为：4

第一棵树的叶子结点数为：2

第一棵树的层数为：3

从先序遍历结果  FGI!!!HJ!!K!!  创建第二棵二叉树

第二棵树的先序遍历：
FGIHJK
第二棵树的中序遍历：
IGFJHK
第二棵树的后序遍历：
IGJKHF
第二棵树的层序遍历：
FGHIJK
第二棵树的结点数为：6

第二棵树的叶子结点数为：3

第二棵树的层数为：3

将第一棵树销毁

将第二棵树销毁
```

完成了题目但做了一些改动：

将原来的 main 函数拆到 BinTreeDemo.cpp 文件中

函数原型和函数实现拆分开

最后编译得到 BinTreeDemo，运行结果存在 logfile 中的 q3result.txt 中

四、 编程题（均可以在 06BinTree.cpp 中实现，自行设计函数参数和返回值并调试通过）

a) 已知在二叉链表表示的二叉树中，root 为根结点，p 和 q 分别指向二叉树中两个结点，试编写算法求距离它们最近的共同祖先。

```
ElemType nearestAncestor(BiTree bt, ElemType x, ElemType y) {
    if (!bt || !(findInTree( bt: bt->left,  key: x) && findInTree( bt: bt->left,  key: y))) {
        cout << "They have no common ancestor.\n";
        return -1;
    }

    ElemType nearAnc = bt->data;

    if (findInTree( bt: bt->left,  key: x) && findInTree( bt: bt->left,  key: y))
        nearAnc = nearestAncestor( bt: bt->left, x, y);
    else if (findInTree( bt: bt->right,  key: x) && findInTree( bt: bt->right,  key: y))
        nearAnc = nearestAncestor( bt: bt->right, x, y);

    return nearAnc;
}
```

```
bool findInTree(BiTree bt, ElemType key) {
    if (bt) {
        if (key == bt->data) return true;
        return findInTree( bt: bt->left, key) || findInTree( bt: bt->right, key);
    } return false;
}
```

由两个函数实现，思路为不断寻找当前节点是否为共同祖先，如果子树结点中没有更近的，则返回并更新 nearAnc 值。

借用上一题的树，两个结果大致如下：

q4.1result.txt | q4.2result.txt | q5result.txt | root2leaf.cpp

```
1  The nearest ancestor of J and K is: H
2  The nearest ancestor of D and E is: B
3
```

b) 试给出算法求出一条从根到叶子结点的序列，其结点数为层数。

```
void root2leaf(BiTree bt) {
    int maxLength = BiTreeLevelCount(bt);
    ElemType str[maxLength];
    for (int i = 0; i < maxLength; i++)
        str[i] = 0;
    if (!bt) return;
    root2leaf(bt, str,  pos: 0, maxLength);
}
```

```
void root2leaf(BiTree bt, ElemType* str, int pos, int maxLength) {
    if (!bt) return;

    str[pos] = bt->data;
    if (!bt->left && !bt->right) // this is a leaf
        cout.write( s: str,  n: pos + 1) << endl;

    if (bt->left) {
        ElemType str1[maxLength];
        strcpy( Dest: str1,  Source: str);
        root2leaf( bt: bt->left,  str: str1,  pos: pos + 1, maxLength);
    }
    if (bt->right) {
        ElemType str2[maxLength];
        strcpy( Dest: str2,  Source: str);
        root2leaf( bt: bt->right,  str: str2,  pos: pos + 1, maxLength);
    }
```

用同名函数，保存最大层数和当前是第几层以及字符串，如果遇到叶子则打印，否则继续向下加长字符串。

借用上一题的树，所有的路径如下：

q4.1result.txt | q4.2result.txt | q5result.txt

```
1  ABCD
2  ABE
3  AFGI
4  AFHJ
5  AFHK
6
```

## 五、 编程题（选做，上交代码源文件或者手写代码均可以，自行设计函数参数和返回值并调试通过）

a) 将后缀表达式读入，构建一棵表达式计算的二叉树，再将二叉树转换为中缀表达式，记得要在合适的位置加入括号维持计算顺序不变。

见 src 下 rpn2infix.cpp 和 cmake-build-debug 中可执行文件以及 logfile 下对于一个源文件中示例后缀表达式的转换结果。