实验4: 树

确保把类定义写在myTree.h中，类实现写在myTree.cpp中

给出的myTree.h中的函数声明不要改动，myTree.h中的内容可以增但不可以删，测试文件将根据已有的函数声明调用，myTree.cpp中的打印函数不要改动。

将类和函数声明写在书myTree.h中，类和函数的实现写在myTree.cpp中，最终提交两个文件：myTree.h和myTree.cpp

1 实现树的节点类TreeNode（支持线索二叉树）和树类MyTree（支持线索二叉树），采用二叉链表存储，完成以下功能：

* 树节点类的初始化和销毁
* 树节点类的打印，void printNode()
* 树初始化MyTree()，初始化一棵空树
* 树初始化MyTree(const char[])，根据二叉树的先序序列，生成二叉树的二叉链表存储,使用@表示NULL
* 树的复制构造函数MyTree(const MyTree&),复制参数中的树
* 树销毁~MyTree()支持普通二叉树与线索二叉树
* 先序遍历二叉树并打印，void preOrderTraverse()//仅支持普通二叉树，不考虑线索化，该函数不可直接递归调用，可添加必要的函数支持，对节点的访问操作为打印该节点
* 中序遍历二叉树并打印，void inOrderTraverse()//支持普通二叉树与线索二叉树，该函数不可直接递归调用，可添加必要的函数支持，对节点的访问操作为打印该节点
* 后序遍历二叉树并打印，void postOrderTraverse()//仅支持普通二叉树，不考虑线索化，该函数不可直接递归调用，可添加必要的函数支持，对节点的访问操作为打印该节点
* 定位二叉树中的节点，TreeNode& locateNode(const char& v); //在树中找到值为v的节点则返回该节点，否则返回NULL，支持普通二叉树与线索二叉树
* 计算二叉树的叶子结点数，int countLeaf()//仅支持普通二叉树，不考虑线索化
* 计算二叉树的深度，int countHeight()//仅支持普通二叉树，不考虑线索化
* 当前树是否是线索二叉树，bool isThreadedTree()，是线索二叉树返回true，否则false
* 为二叉树生成中序线索二叉树 bool inOrderThreading()
* 寻找中序线索二叉树中某节点的前驱节点TreeNode& preNode(const TreeNode&);//仅支持线索二叉树
* 寻找中序线索二叉树中某节点的后继节点，TreeNode& nextNode(const TreeNode&);//仅支持线索二叉树

1. 实现霍夫曼树HuffmanTree，输出对应的霍夫曼编码

* 树初始化HuffmanTree(const int&, const int[])，根据输入创建一棵霍夫曼树，第一个参数为节点个数，第二个参数为节点数组，节点值为节点重要度，越大代表越重要，要求树构建时偏小的值放入左子树，偏大的值放入右子树
* 树销毁~HuffmanTree()
* 输出霍夫曼编码void printHuffmanCodes()//格式：节点值：编码，节点排序递减
* 其他必要的函数

输入输出示例：

MyTree myTree("ABC@@DE@G@@F@@@");

myTree.preOrderTraverse();//ABCDEGF

cout<<endl;

myTree.inOrderTraverse();//CBEGDFA

cout<<endl;

myTree.postOrderTraverse();//CGEFDBA

cout<<endl;

cout<<myTree.countLeaf()<<endl;//3

cout<<myTree.countHeight()<<endl;//5

MyTree myThreadedTree(myTree);

cout<<myThreadedTree.isThreadedTree()<<endl;//0

myThreadedTree.inOrderThreading();

cout<<myThreadedTree.isThreadedTree()<<endl;//1

myThreadedTree.inOrderTraverse();//CBEGDFA

cout<<endl;

TreeNode n = myThreadedTree.locateNode('G');

myThreadedTree.preNode(n).printNode();//E

cout<<endl;

myThreadedTree.nextNode(n).printNode();//D

cout<<endl;

**int** v[]={2,7,5,4};

HuffmanTree hTree(4,v);

hTree.printHuffmanCodes();

/\*

7:0

5:10

4:111

2:110

\*/