

# 第一题：二叉树的基础算法

先根据二叉树的先序、中序遍历序列还原该二叉树，求出这棵二叉树的高度和宽度，以及其叶节点的个数。再将这棵二叉树在内存当中镜像克隆一份，输出克隆的这棵树的层次遍历序列，最后再释放掉这两棵树所占的内存空间。

你需要实现以下几个函数：

- `Node* createTree(int preOrder[], int inOrder[], int N);`
    - 二叉树的先序、中序遍历序列分别存放在`preOrder`、`inOrder`数组当中，这两个数组存放元素的个数都为`N`。
    - 函数返回创建好的二叉树的根节点指针
  - `int getTreeHeight(Node* root);`
    - 约定：空树的高度为0，一棵树如果只有1个节点（根节点）其高度为1
  - `int getTreeWidth(Node* root);`
    - 树的宽度是指：拥有节点数最多的那一层的节点数。
  - `int countLeafNode(Node* root);`
    - 返回二叉树中叶节点的个数，这个没什么好解释的。
  - `Node* mirrorCloneTree(Node* root);`
    - 镜像克隆的含义是你需要在内存中新创建一颗二叉树（不修改原来的树），新创建的树与原来的树为镜像关系。例如：  

1 2 3 4 5 6		原来的树 1 / 2      \ 5 / 3      \ 4	镜像克隆的树 1 / 5      \ 2 / 4      \ 3
----------------------------	--	---	---
  - 函数返回新创建树的根节点指针
- `void levelOrderTraversal(Node* root);`
  - 输出其层次遍历序列，以空格分割，行末应输出一个换行符`\n`
- `void destroyTree(Node* &root);`
  - 释放树所占的内存空间
  - 函数执行完后`root`指针应置为`NULL`

main函数如下：

允许你对main函数进行适量修改，比如你想使用全局数组。

```
1 int main() {
2     int N;
3     scanf("%d", &N);
4     int* preOrder = (int*)malloc(N * sizeof(int));
5     int* inOrder = (int*)malloc(N * sizeof(int));
6     for (int i = 0; i < N; i++) {
7         scanf("%d", &preOrder[i]);
8     }
9     for (int i = 0; i < N; i++) {
10        scanf("%d", &inOrder[i]);
11    }
12    Node* root = createTree(preOrder, inOrder, N);
13    printf("该树的高度是: %d\n", getTreeHeight(root));
```

```

14     printf("该树的宽度是: %d\n", getTreeWidth(root));
15     printf("该树的叶节点个数是: %d\n", countLeafNode(root));
16     Node* newTreeRoot = mirrorCloneTree(root);
17     printf("该树的镜像树的层次遍历序列为:\n");
18     lavelOrderTraversal(newTreeRoot);
19     destoryTree(root);
20     destoryTree(newTreeRoot);
21     return 0;
22 }

```

样例输入：

```

1 7
2 4 1 3 2 6 5 7
3 1 2 3 4 5 6 7

```

其对应的二叉树为:

```

1      4
2     / \
3    1   6
4     \ / \
5      3 5 7
6     /
7    2

```

样例输出：

```

1 该树的高度是：4
2 该树的宽度是：3
3 该树的叶节点个数是：3
4 该树的镜像树的层次遍历序列为：
5 4 6 1 7 5 3 2

```

温馨提示：

1. 函数的接口不允许修改，如果你觉得实现起来不方便，你可以自己重新写一个函数，然后直接调用你自己写的函数。比如createTree 函数你想使用递归来实现，但我给你的函数接口很难写成递归函数，那么你可以自己另外写一个递归函数，然后在createTree函数中直接调用你写的递归函数。

## 第二题：后序遍历的第k个节点

(这道题是清华大学的考研真题)

假设二叉树的节点定义如下：

其中节点的size成员表示：当前节点以及其孩子节点的总数，也可以理解为以该节点为根的子树当中，所有的节点个数。

```

1 struct BinNode{
2     int size; //当前节点以及其孩子节点的总数
3     BinNode *lchild,*rchild;
4 };

```

编写一个算法：返回后序遍历的第 K 个 (K从1开始计数) 节点(该节点记为x)，要求时间复杂度不超过该节点的深度，即O(depth(x))

```

1  BinNode *rank(BinNode* t,int k){
2      //有效代码行数不超过12行
3      //不要尝试模拟后序遍历，时间复杂度会超时（按照清华的评分标准这题直接得0分）
4  }

```

这道题的代码的框架如下：

你只需要实现rankInPostorder 和generateEachNodeSize这两个函数。

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      int size;
7      Node *lchild, *rchild;
8
9      Node(int x = 0) { data = x; lchild = rchild = NULL; }
10
11     void setChildNode(Node* l, Node* r) {
12         lchild = l;
13         rchild = r;
14     }
15 };
16
17 Node* rankInPostorder(Node* t, int k) {
18
19 }
20
21 //计算每个节点的size的值
22 void generateEachNodeSize(Node* root) { //需要的话你可以把该函数的返回值改为int类型
23
24 }
25
26 int main() {
27     Node* root = new Node(1);
28     Node* node_2 = new Node(2);
29     Node* node_3 = new Node(3);
30     Node* node_4 = new Node(4);
31     Node* node_5 = new Node(5);
32     Node* node_6 = new Node(6);
33     Node* node_7 = new Node(7);
34     Node* node_8 = new Node(8);
35
36     root->setChildNode(node_2, node_3);
37     node_2->setChildNode(node_4, node_5);
38     node_3->setChildNode(NULL, node_7);
39     node_5->setChildNode(node_6, NULL);
40     node_4->setChildNode(NULL, node_8);
41
42     generateEachNodeSize(root);
43
44     for (int i = 1; i <= 8; i++) {
45         Node* p = rankInPostorder(root, i);
46         printf("%d ", p->data);
47     }
48     printf("\n");
49     return 0;
50 }

```

如果你写的算法正确，程序应当输出：

1 | 8 4 6 5 2 7 3 1

该二叉树的结构为：

