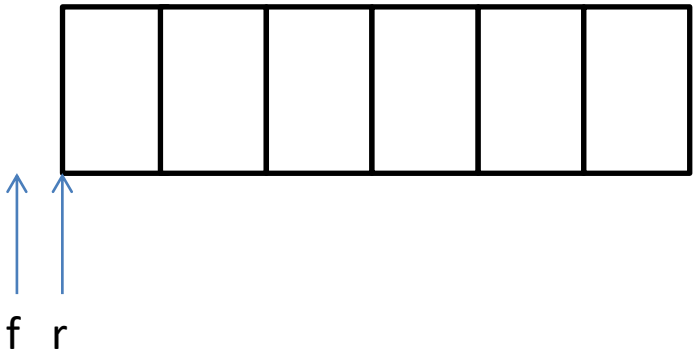
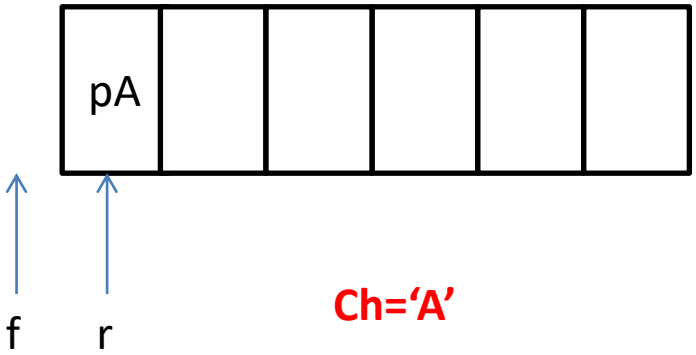
构造函数：根节点创建

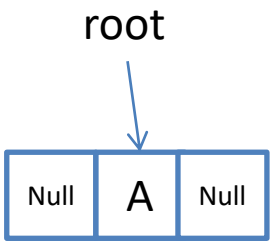



```
void main()
{
    Tree<string> t; //创建一棵树
    /*
       例如对教材P130 图5-3 (a) 可按如下方式输入:
       A
       A B
       C G   //有意输入错误

       B D
       B E
       B F

       C G   //有意输入错误

       A C
       C G
       C H
       E I
     */
```



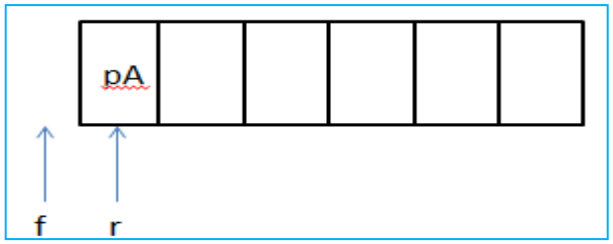
Ch='A'

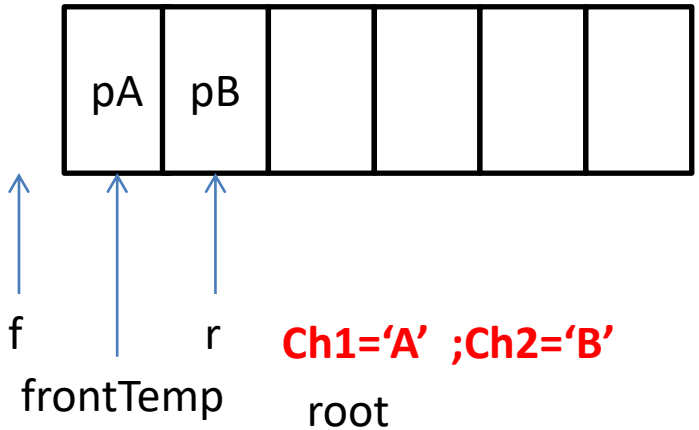
```
const int MaxSize = 100;
int end = 0;
int front = 0;//队列头
int frontTemp=0;
int rear  = 0;  //队列尾，采用顺序队列，并假定不会发生上溢
int j = 0;
TNode<T>* queue[MaxSize]; //声明一个队列
TNode<T>* tempNode;                 //声明指向结点类型的指针
TNode<T>* brotherNode;    //工作指针
```



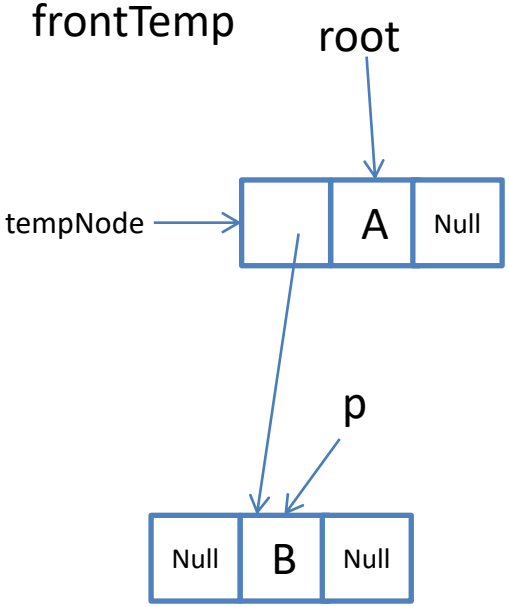

```
T ch;
cout<<"请输入创建一棵树的根结点数据"<<endl;
cin>>ch;
root = new TNode<T>;
root->data = ch;
root->firstchild = NULL;
root->rightsib = NULL;
queue[rear++] = root;
```

构造函数：创建节点B，建立父子关系





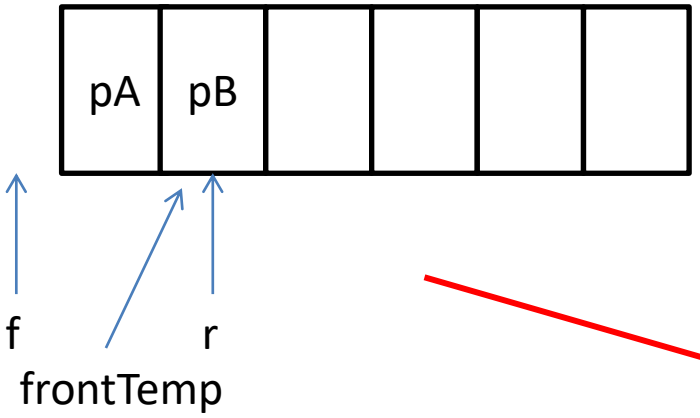
Ch1='A' ;Ch2='B'




```
while (!end)    //若继续创建树 - TODO 判断输入异常情况，end只能0或1
{
    T ch1,ch2;
    cout<<"请输入父结点数据和孩子结点数据对:"<<endl;
    cin>>ch1>>ch2;//ch1 父节点数据，ch2 子节点数据

    //找到输入的父节点
    //tempNode = queue[front];//头结点出队，同时对头元素的标识符后移
    frontTemp=front;
    tempNode = queue[frontTemp++];//取出头结点，不是出队
    //查找已入队且数据和ch1匹配的节点
    while((ch1 != tempNode->data) && (frontTemp<rear) )
    {   //tempNode = queue[front++];
        tempNode = queue[frontTemp++];
    }

    if (ch1 != tempNode->data) // 扫描队列，没找到父亲节点
    {
        cout<<"此前不存在内容为"<<ch1<<"的父亲节点。重新输入节点对！！"<<endl;
        continue;
    }
```

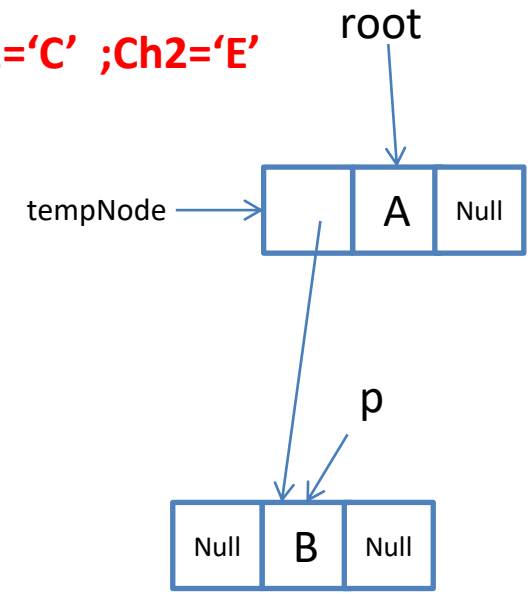
```
else {
    TNode<T>* p = new TNode<T>;        //生成一个结点（子节点）
    p->data = ch2;
    p->firstchild = NULL;
    p->rightsib = NULL;

    queue[rear++] = p;

    //---------以下代码建立父子关系-----------------
    if (tempNode->firstchild == NULL)
        tempNode->firstchild = p;//父节点无子节点，当前节点成为第一个孩子
    else {
        brotherNode = tempNode->firstchild;  //工作指针指向结点的第一个孩子
        while (brotherNode->rightsib != NULL)          //若第一个孩子有兄弟结点
        {
            brotherNode = brotherNode->rightsib;//工作指针指向第一个孩子的右兄弟
        }
        brotherNode->rightsib = p;
    }
    cout << "创建结束？如果结束请按1否则请按0 " << endl;
    cin >> end;
}
```

构造函数：输入C E（队列中找不到父节点）



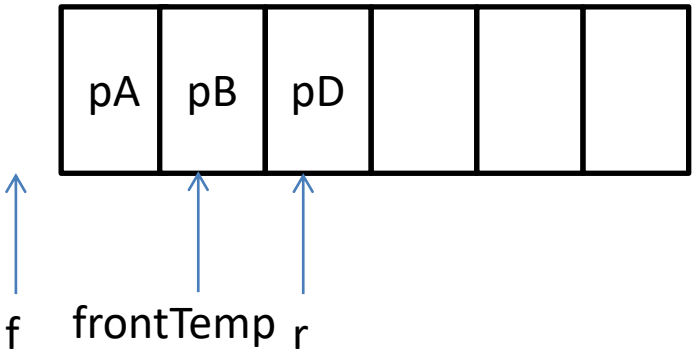

```
while (!end)    //若继续创建树 - TODO 判断输入异常情况，end只能0或1
{
    T ch1,ch2;
    cout<<"请输入父结点数据和孩子结点数据对:"<<endl;
    cin>>ch1>>ch2;//ch1 父节点数据，ch2 子节点数据

    //找到输入的父节点
    //tempNode = queue[front];//头结点出队，同时对头元素的标识符后移
    frontTemp=front;
    tempNode = queue[frontTemp++];//取出头结点，不是出队
    //查找已入队且数据和ch1匹配的节点
    while((ch1 != tempNode->data) && (frontTemp<rear) )
    {   //tempNode = queue[front++];
        tempNode = queue[frontTemp++];
    }

    if (ch1 != tempNode->data) // 扫描队列，没找到父亲节点
    {
        cout<<"此前不存在内容为"<<ch1<<"的父亲节点。重新输入节点对！！"<<endl;
        continue;
    }
```

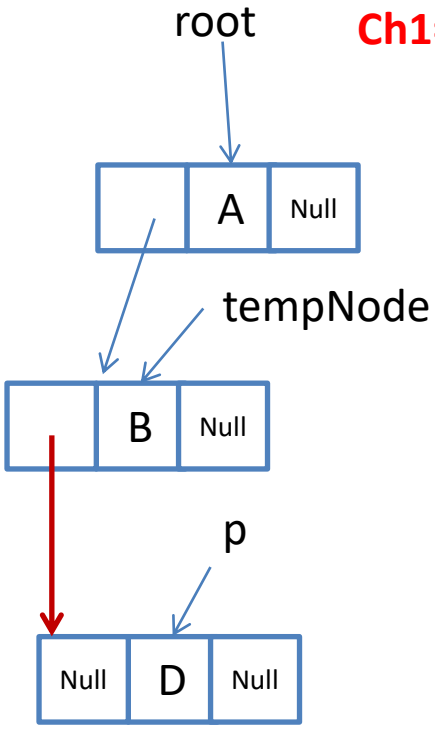
Ch1='C' ;Ch2='E'




```
else {
    TNode<T>* p = new TNode<T>;       //生成一个结点（子节点）
    p->data = ch2;
    p->firstchild = NULL;
    p->rightsib = NULL;

    queue[rear++] = p;

    //---------以下代码建立父子关系-----------------
    if (tempNode->firstchild == NULL)
        tempNode->firstchild = p;//父节点无子节点，当前节点成为第一个孩子
    else {
        brotherNode = tempNode->firstchild;  //工作指针指向结点的第一个孩子
        while (brotherNode->rightsib != NULL)          //若第一个孩子有兄弟结点
        {
            brotherNode = brotherNode->rightsib;//工作指针指向第一个孩子的右兄弟
        }
        brotherNode->rightsib = p;
    }
    cout << "创建结束? 如果结束请按1否则请按0 " << endl;
    cin >> end;
}
```

构造函数：创建节点D，建立父子关系



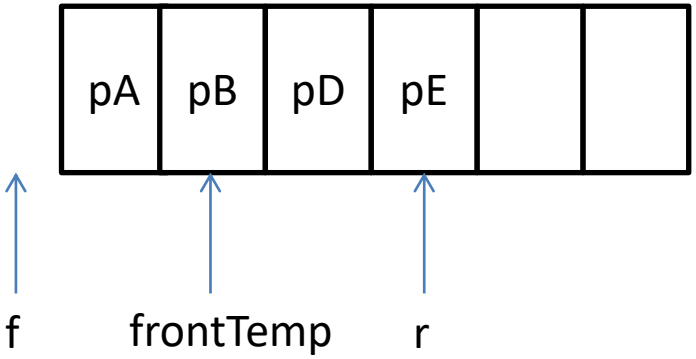


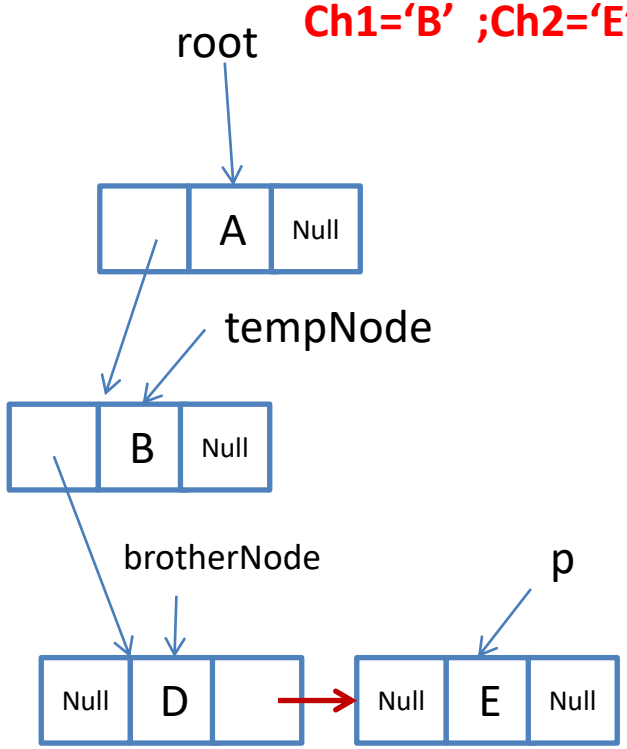
Ch1='B' ;Ch2='D'

```
while (!end)     //若继续创建树 - TODO 判断输入异常情况，end只能0或1
{
    T ch1,ch2;
    cout<<"请输入父结点数据和孩子结点数据对:"<<endl;
    cin>>ch1>>ch2;//ch1 父节点数据，ch2 子节点数据

    //找到输入的父节点
    //tempNode = queue[front];//头结点出队，同时对头元素的标识符后移
    frontTemp=front;
    tempNode = queue[frontTemp++];//取出头结点，不是出队
    //查找已入队且数据和ch1匹配的节点
    while((ch1 != tempNode->data) && (frontTemp<rear) )
    {   //tempNode = queue[front++];
        tempNode = queue[frontTemp++];
    }

    if (ch1 != tempNode->data) // 扫描队列，没找到父亲节点
    {
        cout<<"此前不存在内容为"<<ch1<<"的父亲节点。重新输入节点对！！"<<endl;
        continue;
    }
```

```
else {
    TNode<T>* p = new TNode<T>;       //生成一个结点（子节点）
    p->data = ch2;
    p->firstchild = NULL;
    p->rightsib = NULL;

    queue[rear++] = p;

    //---------以下代码建立父子关系-----------------
    if (tempNode->firstchild == NULL)
        tempNode->firstchild = p;//父节点无子节点，当前节点成为第一个孩子
    else {
        brotherNode = tempNode->firstchild;  //工作指针指向结点的第一个孩子
        while (brotherNode->rightsib != NULL)          //若第一个孩子有兄弟结点
        {
            brotherNode = brotherNode->rightsib;//工作指针指向第一个孩子的右兄弟
        }
        brotherNode->rightsib = p;
    }
    cout << "创建结束? 如果结束请按1否则请按0 " << endl;
    cin >> end;
}
```

构造函数：创建节点E，建立父子关系

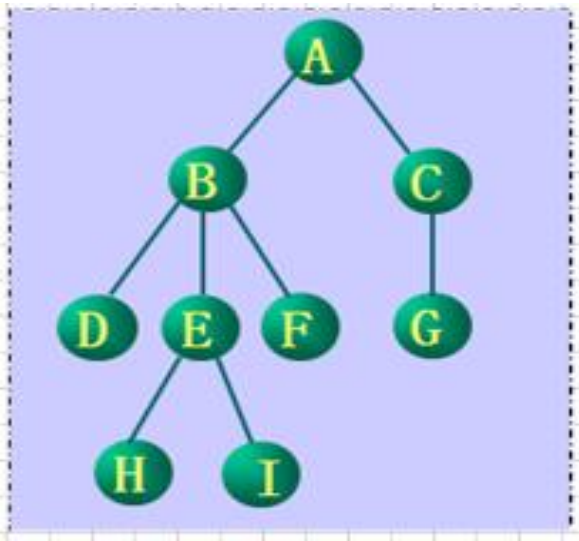




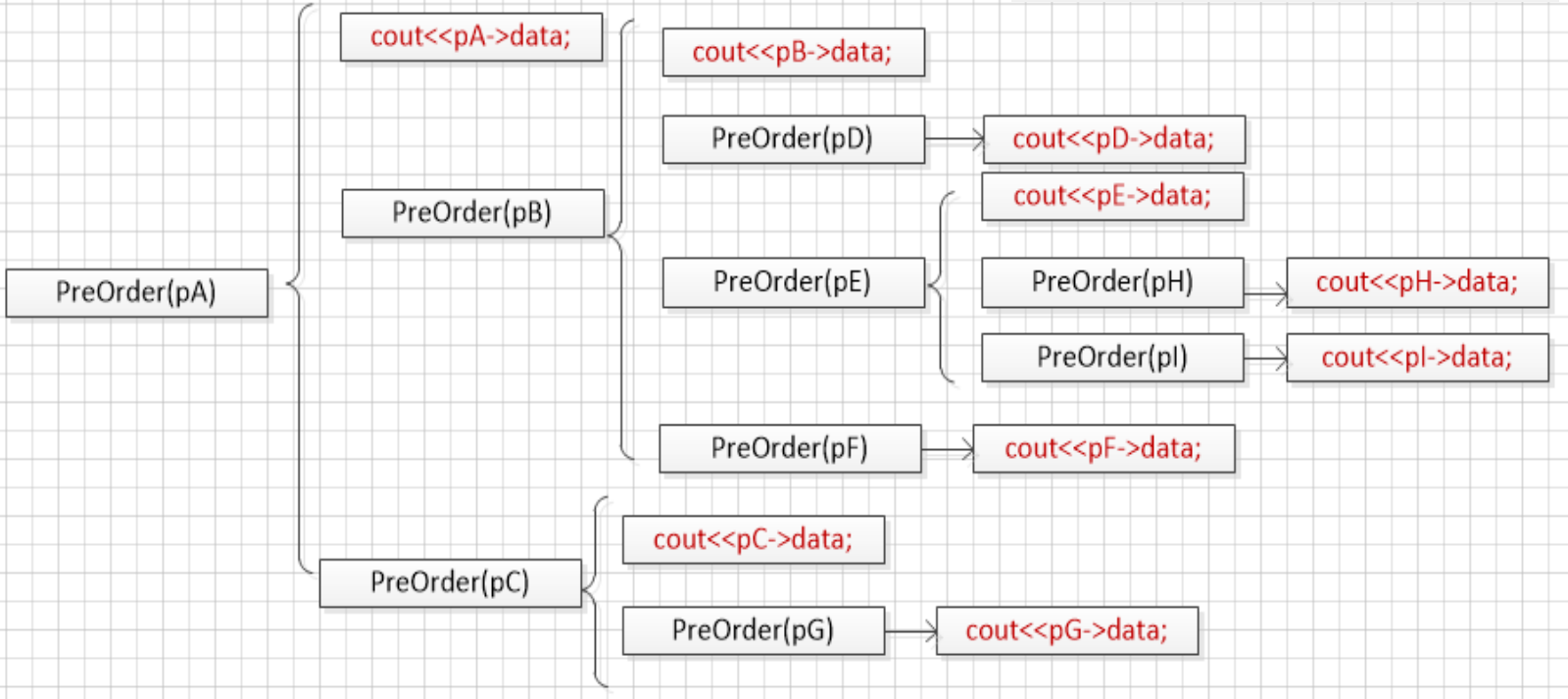

```
while (!end)     //若继续创建树 - TODO 判断输入异常情况，end只能0或1
{
    T ch1,ch2;
    cout<<"请输入父结点数据和孩子结点数据对:"<<endl;
    cin>>ch1>>ch2;//ch1 父节点数据，ch2 子节点数据

    //找到输入的父节点
    //tempNode = queue[front];//头结点出队，同时对头元素的标识符后移
    frontTemp=front;
    tempNode = queue[frontTemp++];//取出头结点，不是出队
    //查找已入队且数据和ch1匹配的节点
    while((ch1 != tempNode->data) && (frontTemp<rear) )
    {   //tempNode = queue[front++];
        tempNode = queue[frontTemp++];
    }

    if (ch1 != tempNode->data) // 扫描队列，没找到父亲节点
    {
        cout<<"此前不存在内容为"<<ch1<<"的父亲节点。重新输入节点对！！"<<endl;
        continue;
    }
```

```
else {
    TNode<T>* p = new TNode<T>;       //生成一个结点（子节点）
    p->data = ch2;
    p->firstchild = NULL;
    p->rightsib = NULL;

    queue[rear++] = p;

    //---------以下代码建立父子关系-----------------
    if (tempNode->firstchild == NULL)
        tempNode->firstchild = p;//父节点无子节点，当前节点成为第一个孩子
    else {
        brotherNode = tempNode->firstchild;  //工作指针指向结点的第一个孩子
        while (brotherNode->rightsib != NULL)          //若第一个孩子有兄弟结点
        {
            brotherNode = brotherNode->rightsib;//工作指针指向第一个孩子的右兄弟
        }
        brotherNode->rightsib = p;
    }
    cout << "创建结束? 如果结束请按1否则请按0 " << endl;
    cin >> end;
}
```

理解递归-前序遍历

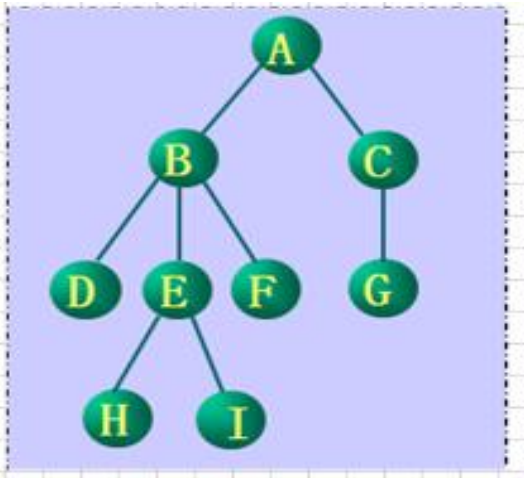


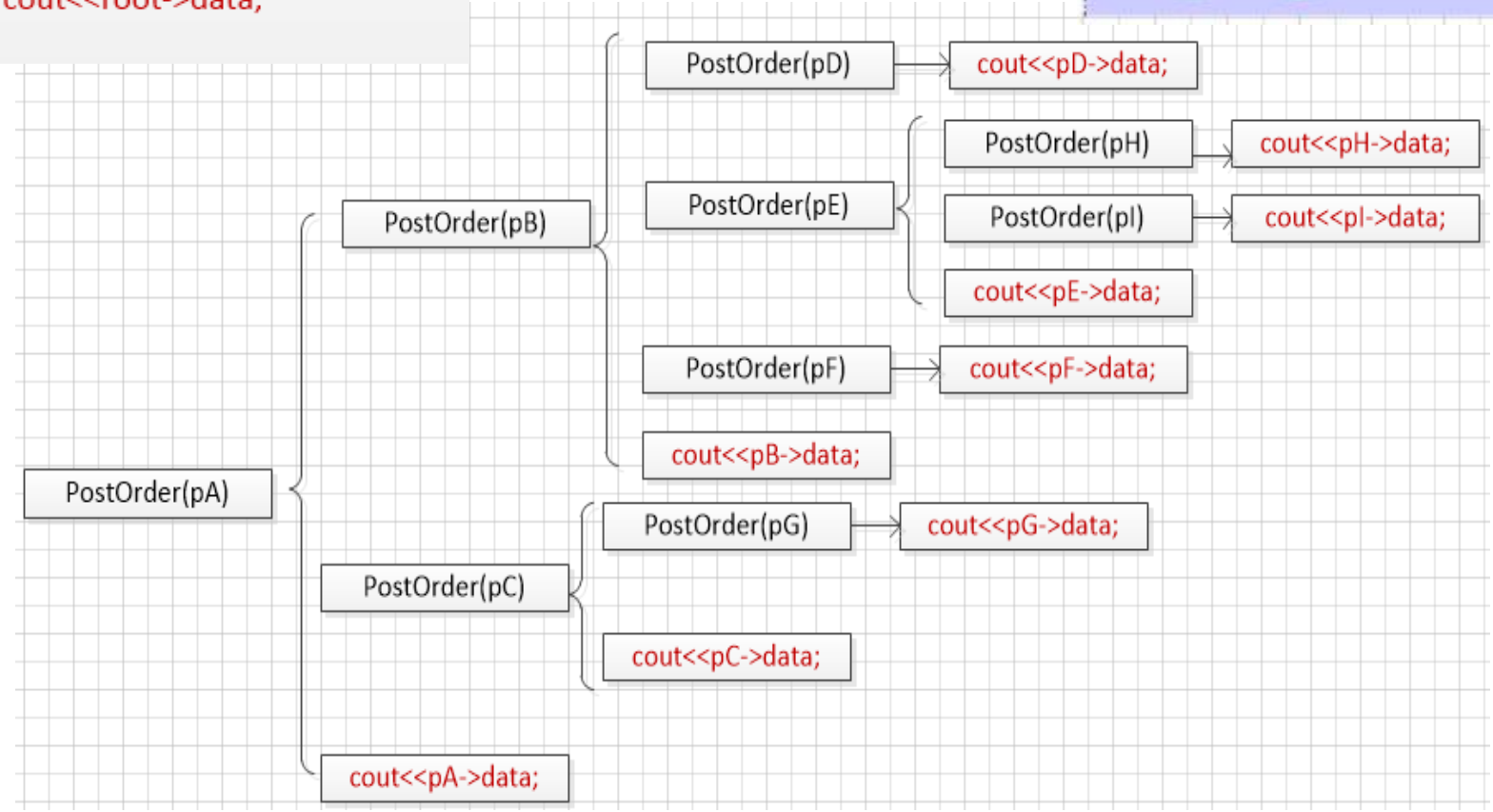

```
void PreOrder(TreeNode *root);
{  if root==Null
       return
   cout<<root->data;
    if root->Child1!=Null
       PreOrder( root->Child1);
    if root->Child2!=Null
       PreOrder( root->Child2);
    ...
    if root->Childn!=Null
       PreOrder( root->Childn);
}
```



理解递归-后序遍历

```
void PostOrder(TreeNode *root);
{  if root==Null
       return
   if root->Child1!=Null
       PostOrder( root->Child1);
    if root->Child2!=Null
       PostOrder( root->Child2);
    ...
    if root->Childn!=Null
       PostOrder( root->Childn);
   cout<<root->data;
}
```

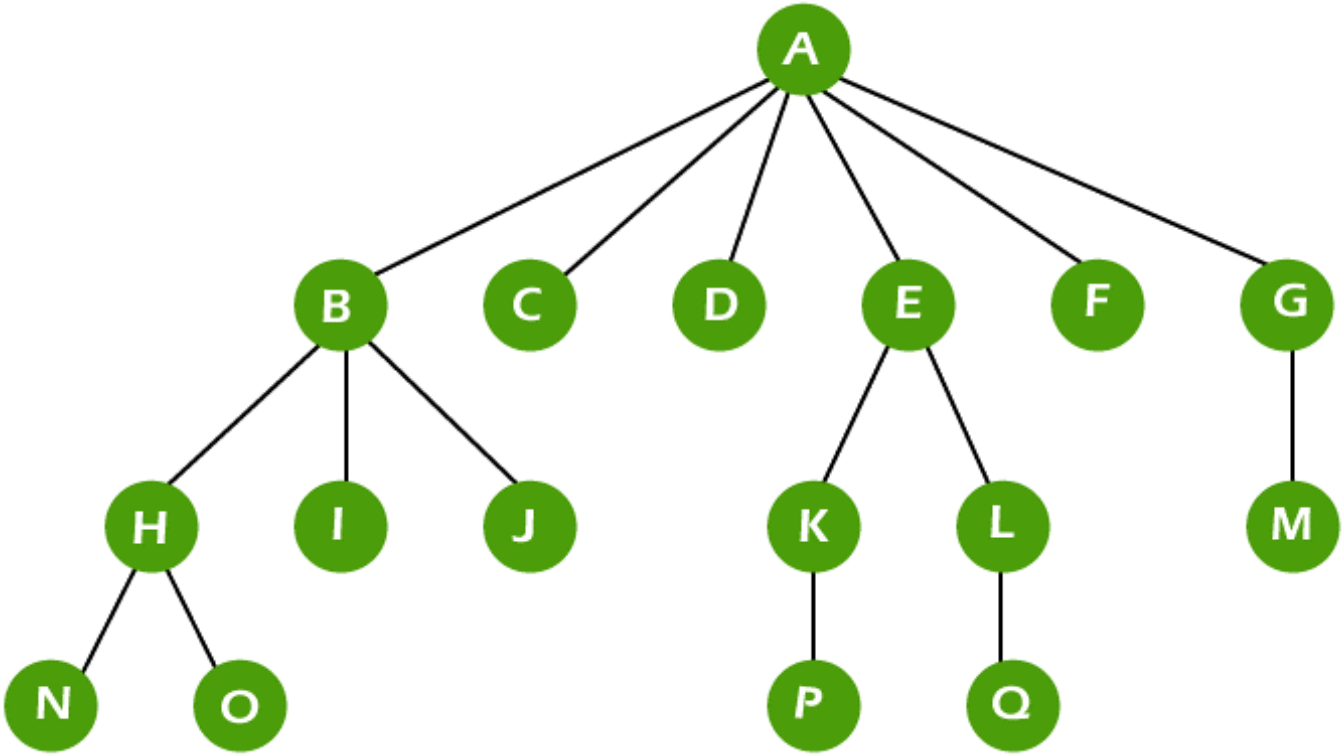




实验内容（四选一）

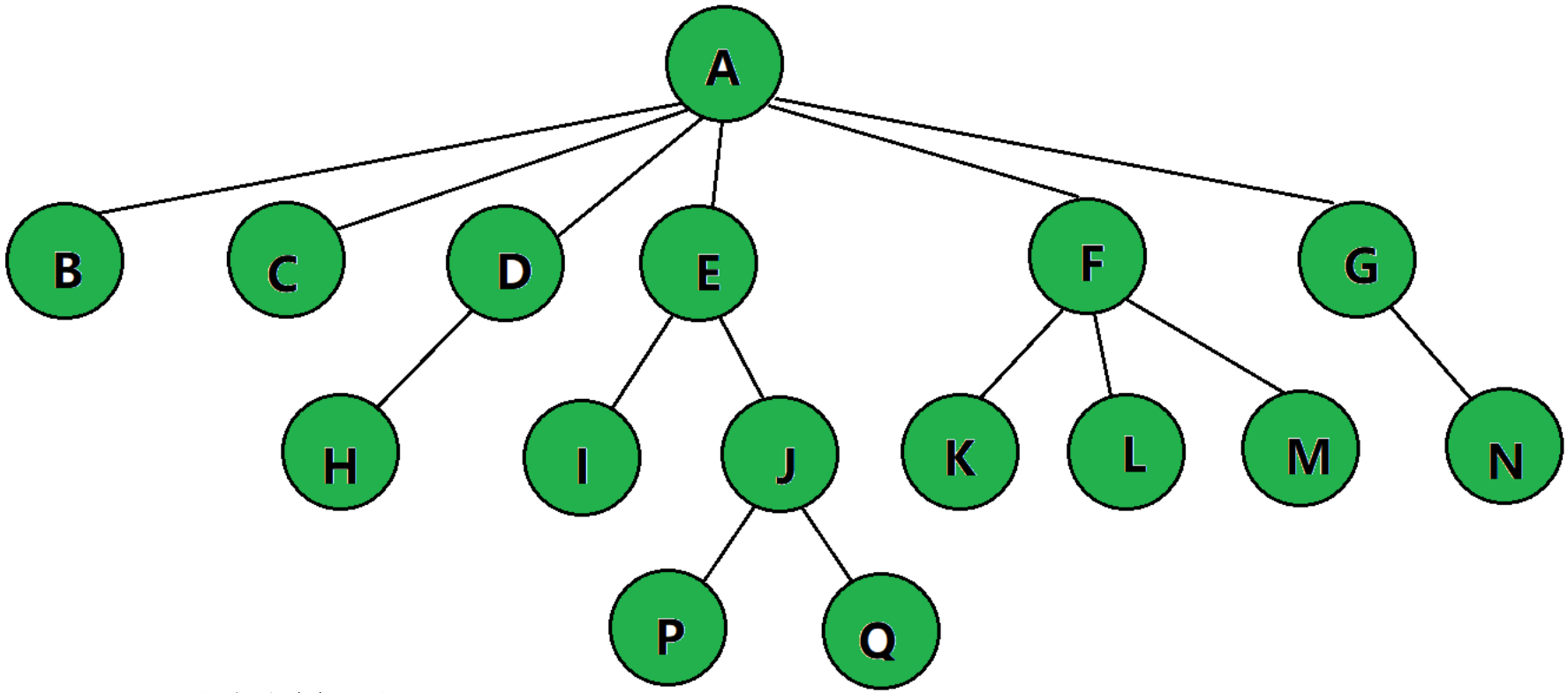
1. 测试树构造
2. 写出前序遍历和后序遍历结果
3. 画出递归版：前序遍历图和（或）后序遍历图

测试用例1



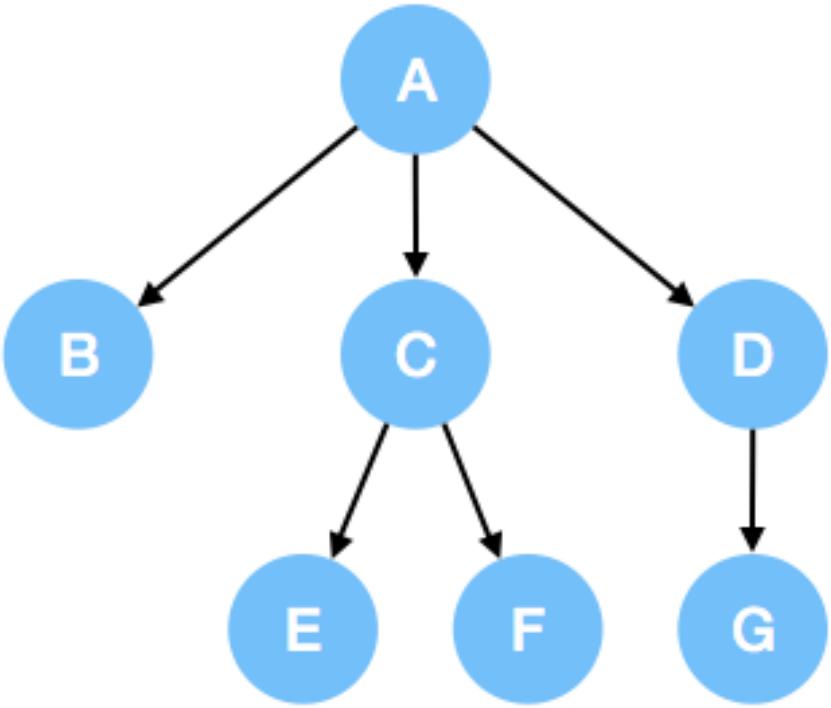
1. 测试树构造
2. 写出前序遍历和后序遍历结果
3. 画出递归版：前序遍历图

测试用例2



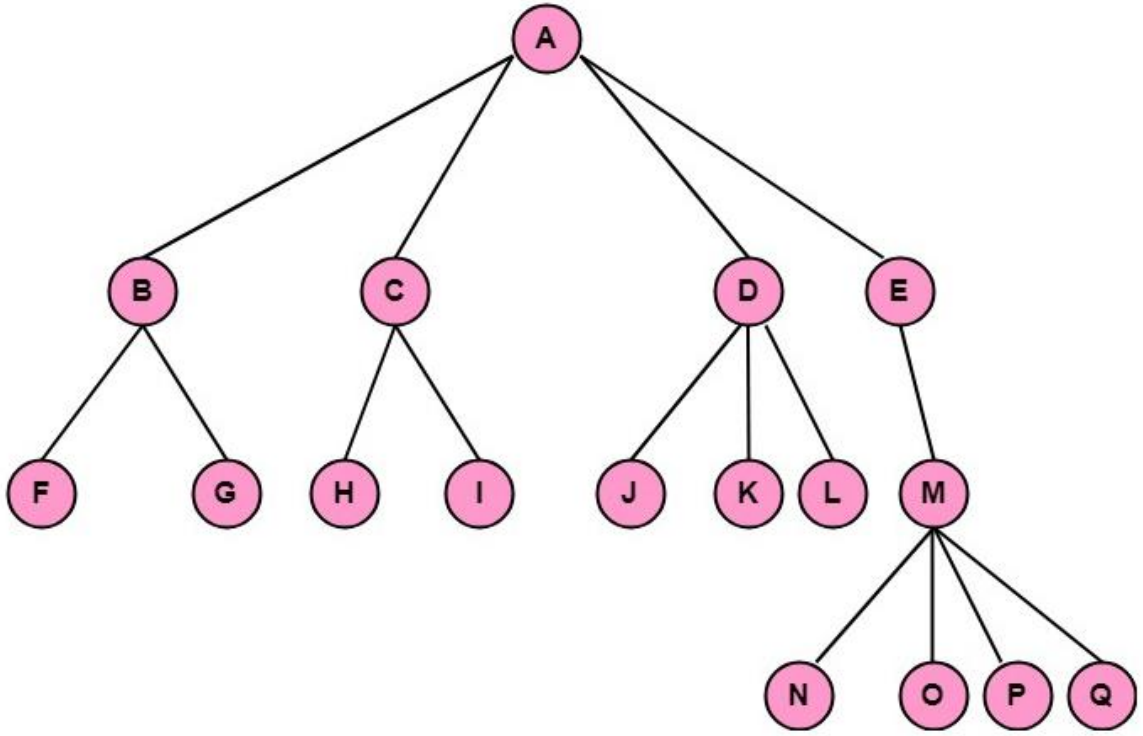
1. 测试树构造
2. 写出前序遍历和后序遍历结果
3. 画出递归版：后序遍历图

测试用例3



1. 测试树构造
2. 写出前序遍历和后序遍历结果
3. 画出递归版：前序遍历图和后序遍历图

测试用例4



1. 测试树构造
2. 写出前序遍历和后序遍历结果
3. 画出递归版：前序遍历图或后序遍历图（一个即可）