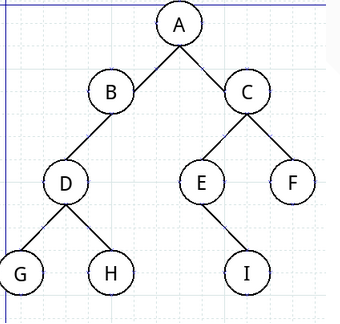
# 遍历二叉树、线索二叉树

**一、遍历二叉树**

**二叉树存储结构：**二叉链表结构，即：1个数据域、2个指针域(分别指向左右孩子)

如这样一个二叉树：

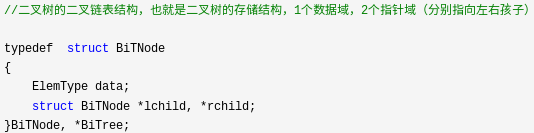


它的前序遍历结果：A B D G H C E I F (规则：先根结点、再先序遍历左子树、最后先序遍历右子树)

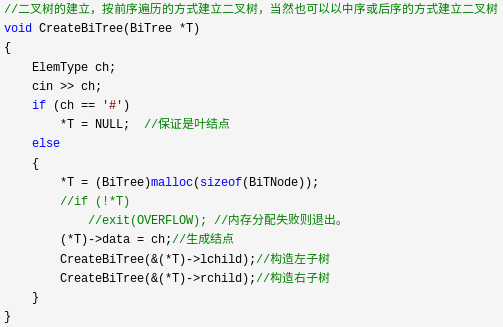
它的中序遍历结果：G D H B A E I C F (规则：中序遍历左子树、访问根结点、中序遍历右子树)

它的后序遍历结果：G H D B I E F C A (规则：后序遍历左子树、后序遍历右子树、访问根结点)

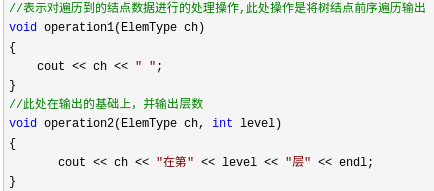
**二叉树的存储结构**(**二叉链表**)



**建立一个二叉树**

****

**1、递归方法遍历**

****

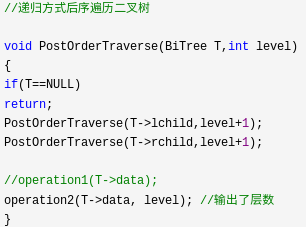
1. **、递归方式前序遍历二叉树**



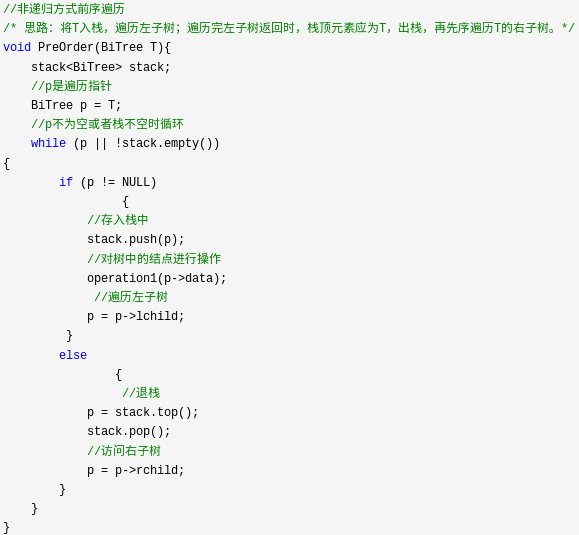
1. **、递归方式中序遍历二叉树**

****

1. **、递归方式后序遍历二叉树**

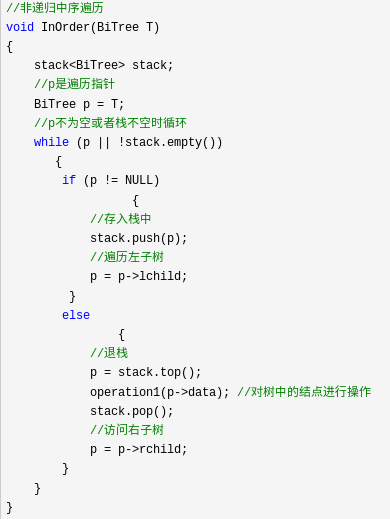
****

1. **非递归方法遍历**
2. **、非递归方式前序遍历二叉树**

****

**2)、非递归方式中序遍历二叉树**

**#include <stack>**

****

1. **、非递归方式后序遍历二叉树(有点绕，还没仔细研究)**

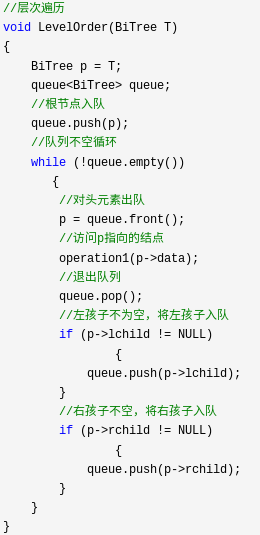
**#include <stack>**





**3、层次遍历**

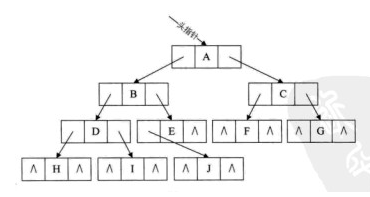
#include <queue>



**二、线索二叉树**

线索二叉树原理：

   通过考察各种二叉链表，不管儿叉树的形态如何，空链域的个数总是多过非空链域的个数。准确的说，n各结点的二叉链表共有2n个链域，非空链域为n-1个，但其中的空链域却有n+1个。如下图所示。



 因此，提出了一种方法，利用原来的空链域存放指针，指向树中其他结点。这种指针称为线索。

    记ptr指向二叉链表中的一个结点，以下是建立线索的规则：

    （1）如果ptr->lchild为空，则存放指向中序遍历序列中该结点的前驱结点。这个结点称为ptr的**中序前驱**；

    （2）如果ptr->rchild为空，则存放指向中序遍历序列中该结点的后继结点。这个结点称为ptr的**中序后继**；

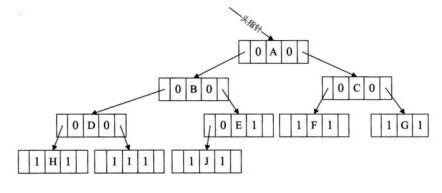
显然，在决定lchild是指向左孩子还是前驱，rchild是指向右孩子还是后继，需要一个区分标志的。因此，我们在每个结点再增设两个标志域ltag和rtag，注意ltag和rtag只是区分0或1数字的布尔型变量，其占用内存空间要小于像lchild和rchild的指针变量。结点结构如下所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| lchild | Ltag | Data | Rtag | rchild |

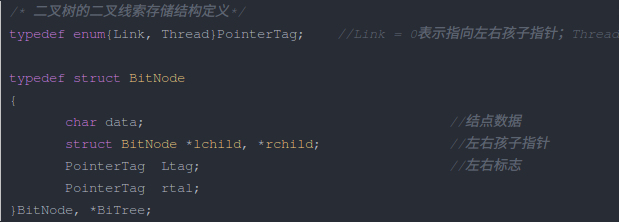
其实：1）ltag为0时指向该结点的左孩子，为1时指向该结点的前驱

2）rtag为0时指向该结点的右孩子，为1时指向该结点的后继

3）因此对于上图的二叉链表图可以修改为下图的养子

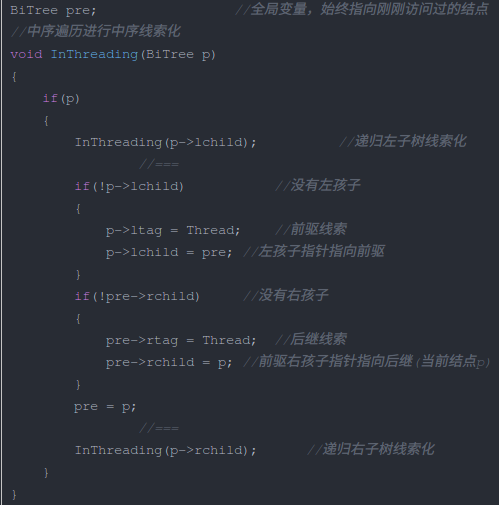


二叉树存储结构如下：

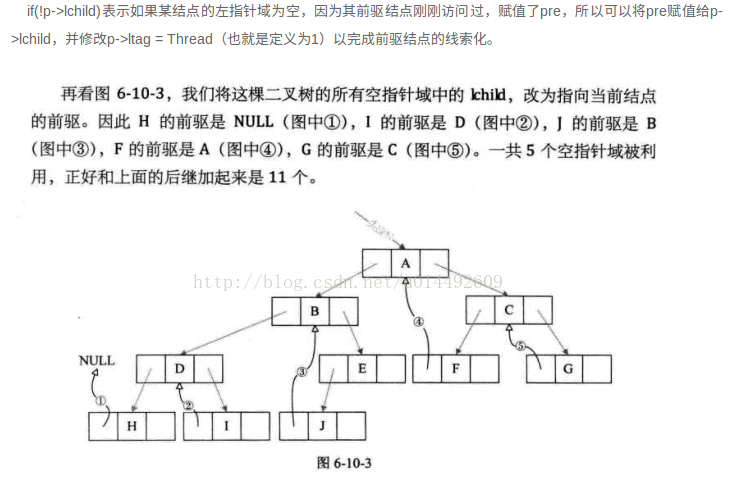
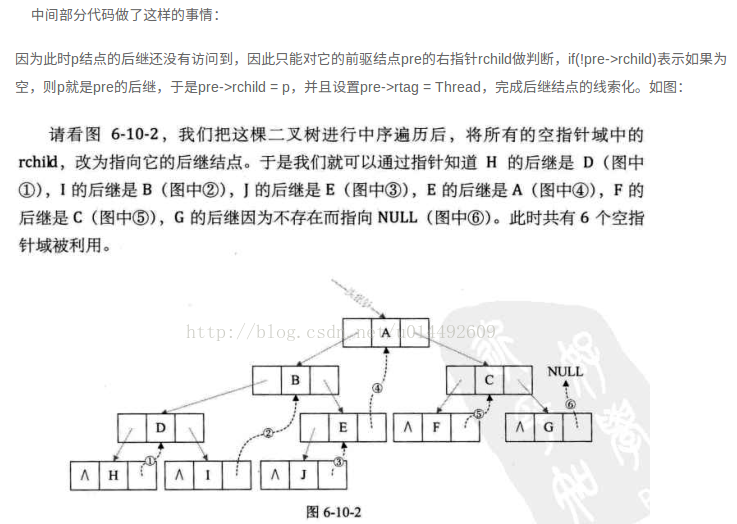


线索化的实质就是将二叉链表中的空指针改为指向前驱或后继的线索。由于前驱和后继信息只有在遍历该二叉树时才能得到，所以，线索化的过程就是在遍历的过程中修改空指针的过程。

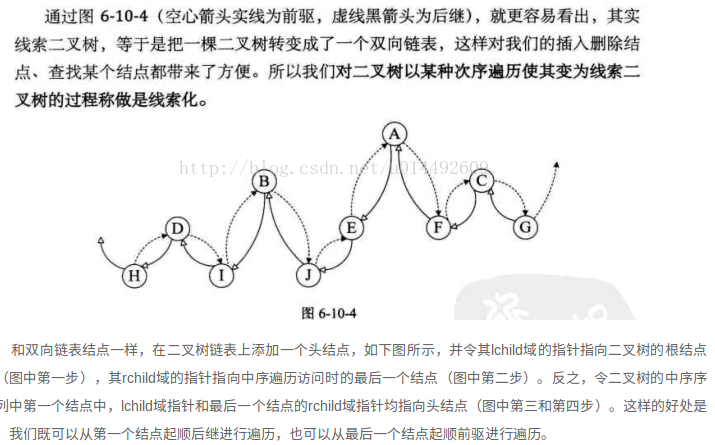
 中序遍历线索化的递归函数代码如下

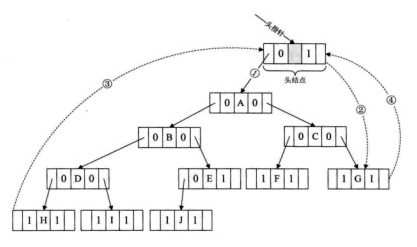


上述代码除了//===之间的代码以外，和二叉树中序遍历的递归代码机会完全一样。只不过将打印结点的功能改成了线索化的功能。

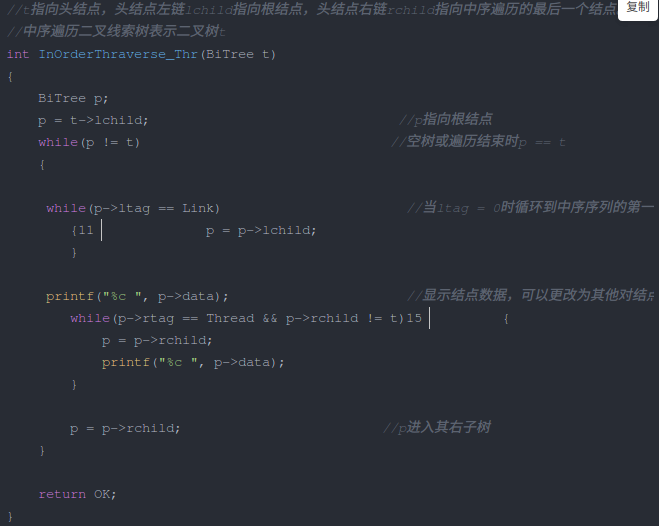


 完成前驱和后继的判断后，不要忘记当前结点p赋值给pre，以便于下一次使用。

 有了线索二叉树后，对它进行遍历时，其实就等于操作一个双向链表结构。



 遍历代码如下所

（1）代码中，p = t->lchild；意思就是上图中的第一步，让p指向根结点开始遍历；  
    （2）while(p != t)其实意思就是循环直到图中的第四步出现，此时意味着p指向了头结点，于是与t相等（t是指向头结点的指针），结束循环，否则一直循环下去进行遍历操作；  
    （3）while(p-ltag == Link)这个循环，就是由A->B->D->H，此时H结点的ltag不是link（就是不等于0），所以结束此循环；  
    （4）然后就是打印H；  
    （5）while(p->rtag == Thread && p->rchild != t)，由于结点H的rtag = Thread（就是等于1），且不是指向头结点。因此打印H的后继D，之后因为D的rtag是Link，因此退出循环；  
    （6）p=p->rchild；意味着p指向了结点D的右孩子I；  
    （7）.....，就这样不断的循环遍历，直到打印出HDIBJEAFCG，结束遍历操作。

从这段代码可以看出，它等于是一个链表的扫描，所以时间复杂度为O(n)。  
    由于充分利用了空指针域的空间（等于节省了空间），又保证了创建时的一次遍历就可以终生受用后继的信息（意味着节省了时间）。所以在实际问题中，如果所用的二叉树需要经过遍历或查找结点时需要某种遍历序列中的前驱和后继，那么采用线索二叉链表的存储结构就是非常不错的选择。

**创建二叉树：**



**创建头结点并线索化**

****