

二叉树

代码题 1

题干

- 题目：

树的结构体：

```
typedef struct BTreeNode
{
    char data;
    struct BTreeNode *lchild, *rchild;
} BTreeNode, *BiTree;
```

1、计算二叉树中所有结点个数

- 总结：
 - 方法一：操作型
 - 第一步：分析计算公式/操作公式与位置
 - 第二步：分析出口条件
 - 方法二：计算型

解答：计算型

```
int CalNumber(BTreeNode* p)
{
    if(p==null)
    {
        return 0;
    }

    int n1 = CalNumber(p->lchild);
    int n2 = CalNumber(p->rchild);

    return n1+n2+1;
}
```

解答：操作型

```
void CalNumber(BTreeNode* p,int& n)
{
    if(p==null)
    {
```

```

        return;
    }

    n++;
    CalNumber(p->lchild,n);
    CalNumber(p->rchild,n);
}

```

代码题 2

题干

- 题目：
 - 2、计算二叉树中所有叶子节点的个数
- 总结：
 - 出口条件为左右孩子都为空；

解答：计算型

```

int count(BTNode* p)
{
    if(p==null)
    {
        return 0;
    }
    if(p->lchild==null&& p->rchild==null)
    {
        return 1;
    }

    int n1 = count(p->lchild);
    int n2 = count(p->rchild);

    return n1+n2;
}

```

代码题 3

题干

- 题目：
 - 3、计算二叉树中所有双分支的节点个数
- 总结：
 -

解答：计算型

```

int count(BTNode* p)
{
    if(p==null)
    {
        return 0;
    }
    if(p->lchild!=null && p->rchild!=null)
    {
        // 左右结点都不为空时，return的时候多加1
        int n1 = count(p->lchild);
        int n2 = count(p->rchild);
        return n1+n2+1;
    }
    else
    {
        // 左右结点有一个为空，return的时候直接求和左右结点的返回值
        int n1 = count(p->lchild);
        int n2 = count(p->rchild);
        return n1+n2;
    }
}

```

代码题 4

题干

- 题目：
 - 4、计算二叉树的深度
- 总结：
 -

解答：计算型

```

int depth(BTNode* p)
{
    if(p==null)
    {
        return 0;
    }
    int n1 = depth(p->lchild);
    int n2 = depth(p->rchild);

    int n = (n1>n2?n1:n2) + 1;
    return n;
}

```

解答：操作型

```

void depth(BTNode* p,int& n,int& max)
{
    if(p==null)
    {
        return;
    }

    // 从上一层进入此层时，visit当前结点，把深度加1
    n++;

    // 如果当前深度更大，则更新max的值
    if(n>max)
    {
        max = n;
    }

    depth(p->lchild,n,max);
    depth(p->rchild,n,max);

    // 因为接下来返回上一层结点，当前的n要-1
    n--;
}

```

代码题 5

题干

- 题目：

5、 $(a-(b+c))*(d/e)$ 存储在二叉树，遍历求值

- 总结：

- 当前的值结点都为叶子节点，符号都存储在分支节点中；
- 因此可以用计算型递归，计算左右结点的值后，将他们按照操作处理；

解答

```

int CalValue(BTNode* root)
{
    if(root == null)
    {
        return 0;
    }

    // 当前节点类型为叶节点，因此直接返回值
    if(root->lchild==null && root->rchild==null)
    {
        return root->data;
    }
}

```

```

// 当前节点类型为分支节点，按照符号进行运算
if(root->lchild!=null && root->rchild!=null)
{
    int n1 = CalValue(root->lchild);
    int n2 = CalValue(root->rchild);

    // 返回n1和n2按照当前符号的运算结果
    return op(n1,n2,p->data);
}
}

```

代码题 6

题干

- 题目：
 - 6、找出二叉树中最大值的点
- 总结：
 -

解答

```

void Max(BTNode* p,int& max)
{
    if (p !=NULL)
    {
        if(p->data>max)
        {
            max=p->data;
            Max(p->lchild,max);
            Max(p->rchild,max);
        }
    }
}

```

代码题 7

题干

- 题目：
 - 7、判断两个二叉树是否相似（指都为空或者都只有一个根节点，或者左右子树都相似）
 -
- 总结：
 - 用两根指针，对两棵树同步进行遍历，判断它们结构是否相等；

解答

```

int fun(BTNode* p1,BTNode* p2)
{
    // 两节点都空，相似
    if(p1==null&& p2==null)
    {
        return 1;
    }

    // 两个结点只有一个为空，不相似
    if(p1==null||p2==null)
    {
        return 0;
    }

    // 两个节点都不为空时，进入到它下一层、继续比较
    int left = fun(p1->lchild,p2->lchild);
    int right = fun(p1->rchild,p2->rchild);
    return (left&&right);
}

```

代码题 8

题干

- 题目：

8、把二叉树所有节点左右子树交换

- 总结：

- 采用后根遍历，先将左右子树交换，确保左右子树已经符合交换定义后、再对整个树进行交换

解答

```

void Invert(BTNode* p)
{
    if(p==null)
    {
        return;
    }
    else
    {
        // 先将子树的子树倒转，最后交换自己的子树
        Invert(p->lchild);
        Invert(p->rchild);
        BTNode* temp = p->lchild;
        p->lchild = p->rchild;
        p->rchild = temp;
    }
}

```

```
}
```

代码题 9

题干

- 题目：

9、查找二叉树中 data 域等于 key 的结点是否存在，若存在，将 q 指向它，否则 q 为空

- 总结：

- 查找结点应当使用先序遍历，因为如果当前结点的值就等于 key，则无需再对左右进行遍历；

解答

```
void Find(BTNode* p,int data,BTNode* q)
{
    if(p==null)
    {
        return;
    }
    else
    {
        if(p->data == data)
        {
            q = p;
            return;
        }
        Find(p->lchild,data,q);
        Find(p->rchild,data,q);
    }
}
```

代码题 10

题干

- 题目：

10、输出先序遍历第 k 个结点的值

- 总结：

解答

```

void Find(BTNode* p,const int k,int& n)
{
    if(p==null)
    {
        return;
    }
    else
    {
        n++;
        if(n==k)
        {
            std::cout<<p->data;
        }
        else
        {
            Find(p->lchild,k,n);
            Find(p->rchild,k,n);
        }
    }
}

```

代码题 11

题干

- 题目：
 - **11、求二叉树中值为 x 的层号**
- 总结：
 - 用 n 表示层号，以引用的方式逐层传递；
 - 使用先序遍历，进入下一层时、先增加层号；

解答

```

// n从0开始，1表示根节点，即第一层
void Find(BTNode* p,const int x,int& n)
{
    if(p==null)
    {
        return;
    }
    else
    {
        n++;

        if(p->data == x)
        {
            std::cout<<n;
            return;
        }
    }
}

```

```

    }
    else
    {
        Find(p->lchild,x,n);
        Find(p->rchild,x,n);
    }

    n--;
}
}

```

代码题 12

题干

- 题目：
 - 12、树中元素为 x 的结点，删除以它为根的子树
- 总结：
 -

解答

```

void del(BTNode *&bt, char key)
{
    if (bt != NULL)
    {
        if (bt->data == key)
        {
            bt = NULL;
            return;
        }

        del(bt->lchild, key);
        del(bt->rchild, key);
        return;
    }
}

```

代码题 13

题干

- 题目：
 - 13、利用结点的右孩子指针将一个二叉树的叶子节点从左向右连接成一个单链表（head 指向第一个，tail 指向最后一个）

- 总结：
 - 将 tail 的右孩子指向新发现的叶节点；

解答

```
void fun(BTNode* p,BTNode* head,BTNode* tail)
{
    if(p==null)
    {
        return;
    }
    else
    {
        // 如果为叶节点
        if(p->lchild==null&& p->rchild==null)
        {
            // 如果是第一次到达叶节点，则此结点为单链表的头
            if(head==null)
            {
                head = p;
                tail = p;
            }
            else
            {
                tail->rchild = p;
                tail = p;
            }
        }
        else
        {
            fun(p->lchild,head,tail);
            fun(p->rchild,head,tail);
        }
    }
}
```

代码题 14

题干

- 题目：
 - 14、输出根节点到每个叶子结点的路径
- 总结：
 - 利用一个栈存储当前路径上的所有结点，进入下一层结点后，入栈。离开下一层结点、返回本层时，出栈；
 - 如果左右孩子都为空，打印当前栈中所有值；

解答

```

void PrintPath(BTNode* p,char* pathStack,int& top)
{
    if(p == null)
    {
        return;
    }

    // 将当前元素压入栈
    pathStack[++top] = p->data;

    // 如果是叶节点，输出栈元素
    if(p->lchild==null&&p->rchild==null)
    {
        for(int i = 0;i<=top;i++)
        {
            std::cout<<pathStack[i];
        }
    }
    else
    {
        PrintPath(p->lchild,pathStack,top);
        PrintPath(p->rchild,pathStack,top);
        --top;
    }
}

```

代码题 15

题干

- 题目：

15、已知满二叉树先序序列存在于数组中，设计算法将其变成后序序列

- 总结：

- 由满二叉树性质可知：
- 当前先序序列的第一个结点为根节点；
- 当前先序序列的 $L1+1$ 到 $(L1+R1+1)/2$ 为左子树；
- 当前先序序列的 $(L1+R1+1)/2+1$ 到 $R1$ 为右子树；

解答

```

//pre存储先序序列，post存储后序序列
// L1、R1分别指向先序序列的第一个以及最后一个元素
// L2、R2分别指向后续序列的第一个以及最后一个元素
void change(char pre[],int L1,int R1,char post[],int L2,int R2)
{
    if(L1 <= R1 && L2 <= R2)
    {

```

```

// 把当前先序序列的第一个结点，即二叉树的根结点，放到后序序列中根所处的位置
post[R2]= pre[L1];

// 对当前结点的左子树递归此过程
change(pre, L1+1,(L1+R1+1)/2, post, L2,(L2+R2-1)/2);

// 对当前结点的右子树递归此过程
change(pre, (L1+R1+1)/2+1,R1,post, (L2+R2-1)/2+1,R2-1);
}
}

```

代码题 16

题干

- 题目：

16、先序与中序遍历分别存在两个一维数组 A, B 中，试着建立二叉链表

- 总结：

- 第一步：取出当前先序序列第一个值，用其值建立新结点
- 第二步：在中序序列中找到对应值的下标
- 第三步：分别对中序序列的左边、右边递归此操作
 - 当前 i 的左边： A_l+1 到 $i + A_r - A_l$
 - 当前 i 的右边： $i+A_r-A_l+1$ 到 A_r

解答

```

BiTree create(char A[],char B[],int Al,int Ar,int Bl,int Br)
{
    BTreeNode* root = new BTreeNode;

    // 当前A序列的第一个元素为二叉树的根结点
    root->data = A[Al];

    // 根据根节点的值，在中序序列中找到根节点的下标
    int i = 0;
    for(;i<Br;i++)
    {
        if(B[i]==root->data)
        {
            break;
        }
    }

    // 当前i为B序列中、根节点元素的位置
    // B序列的左子树不为空
    if(i>Bl)
    {

```

```

        root->lchild = create(A[],B[],Al+1,i+Ar-Al,Bl,i-1);
    }
    else
    {
        root->lchild = null;
    }

    // B系列的右子树不为空
    if(i<Br)
    {
        root->rchild = create(A[],B[],i+Ar-Al+1,Ar,i+1,Br);
    }
    else
    {
        root->rchild = null;
    }

    return root;
}

```

代码题 17

题干

- 题目：

17、二叉树以顺序方式存在于数组 A 的中，设计算法以二叉链表表示

- 总结：

- 递归的创建

解答

```

// A为数组，i为当前结点位置，n为数组长
BiTree create(int A[],int i,int n)
{
    if(i<n)
    {
        BTreeNode* root = new BTreeNode;
        root->data = A[i];
        // 当前结点的左元素在序列中的下标为 i*2+1
        root->lchild = create(A[],i*2+1,n);

        // 当前结点的右元素在序列中的下标为 i*2+2
        root->rchild = create(A[],i*2+2,n);

        return root;
    }
    return null;
}

```

代码题 19

题干

- 题目：

- 19、先序非递归遍历二叉树

- 总结：

- 用一个栈来接受结点。遍历当前节点时、先将右子树入栈，再将左子树入栈，因为栈是先入后出。所以如果想要先输出左子树，需要左子树后入栈；

解答

```
void NonPre(BTNode* BT)
{
    BTNode* treeStack[maxsize];
    int top = -1;

    BTNode* temp;

    // 如果根节点不为空，将根节点压入栈
    if(BT!=null)
    {
        treeStack[++top] = BT;

        while(top!=-1)
        {
            temp = treeStack[top--];

            if(temp->rchild!=null)
            {
                treeStack[++top] = temp->rchild;
            }

            if(temp->lchild!=null)
            {
                treeStack[++top] = temp->lchild;
            }
        }
    }
}
```

代码题 20

题干

- 题目：

• 20、中序非递归遍历二叉树

- 总结:

-

解答

```
void Nonin(BTNode *bt)
{
    BTNode *Stack[maxsize];
    int top=-1;

    while(top != -1 || bt!=null)
    {
        if (bt != NULL)
        {
            Stack[++top]= bt;
            bt = bt->lchild;
        }
        else
        {
            bt= Stack[top--];
            cout << bt->data <<" ";
            bt = bt->rchild;
        }
    }
}
```

代码题 21

题干

- 题目:

21、后序非递归遍历二叉树

- 总结:

-

解答

```
void Nonpost(BTNode* bt)
{
    BTNode* stack[maxsize];
    BTNode* tag = NULL;
    int top = -1;

    while(top!=-1||bt!=null)
    {
        // 因为是要后序遍历，所以是LRV，需要先走到最左下角
        if(bt->lchild!=NULL)
```



```

    if(bt->lchild!=NULL)
    {
        stack[++top] = bt;
        bt = bt->lchild;
    }
    else // 当前已经走到最左下角，要走向父节点、然后去右下角
    {
        // 走向父节点
        bt = stack[top];

        // 右边不为空，并且当前是第一次往右节点走，往右遍历
        // 如果当前 bt->rchild==tag，则表示当前是已经遍历完右节点、返回此结点因此进入执行else的内容，继续往回走
        if(bt->rchild!=NULL && bt->rchild!=tag)
        {
            bt = bt->rchild;
        }
        else // 左右都已经遍历完了，当前处于 LRV 中的 V
        {
            bt = stack[top--];
            if(bt->data == x)
            {
                int i = 0;
                while(i<top)
                {
                    cout<<stack[i++];
                }
            }
            tag = bt;
            bt = NULL;
        }
    }
}
}
}
}

```

代码题 23

题干

- 题目：
 - 23、找到 p 和 q 最近公共祖先结点 r
- 总结：
 -

解答

```

BTNode* Nonpost(BTNode* bt,BTNode* p,BTNode* q)
{

```

```
BTNode* stack[maxsize];
BTNode* tag = NULL;
int top = -1;
```

```
BTNode* stack1[maxsize];
int top1 = -1;
```

```
BTNode* stack2[maxsize];
int top2 = -1;
```

```
while(top!=-1||bt!=null)
{
```

```
    // 因为是要后序遍历，所以是LRV，需要先走到最左下角
```

```
    if(bt->lchild!=NULL)
```

```
    {
```

```
        stack[++top] = bt;
```

```
        bt = bt->lchild;
```

```
    }
```

```
    else // 当前已经走到最左下角，要走向父节点、然后去右下角
```

```
    {
```

```
        // 走向父节点
```

```
        bt = stack[top];
```

```
        // 右边不为空，并且当前是第一次往右节点走，往右遍历
```

// 如果当前 bt->rchild==tag，则表示当前是已经遍历完右节点、返回此结点因此进入执行else的内容，继续往回走

```
        if(bt->rchild!=NULL && bt->rchild!=tag)
```

```
        {
```

```
            bt = bt->rchild;
```

```
        }
```

```
    else // 左右都已经遍历完了，当前处于 LRV 中的 V
```

```
    {
```

```
        // 当前结点左右都为空，将其弹出，并且指针往上走一格
```

```
        bt = stack[top--];
```

```
        // 如果当前为p，则此时栈内存储的就是p的祖先元素，因此需要将其读入到p自己的
```

栈中

```
        if(bt==p)
```

```
        {
```

```
            int temp = top;
```

```
            while(temp!=-1)
```

```
            {
```

```
                stack1[++top1] = stack[temp--];
```

```
            }
```

```
        }
```

```
        // 如果当前为q，则此时栈内存储的就是q的祖先元素，因此需要将其读入到q自己的
```

栈中

```
        if(bt==q)
```

```
        {
```

```
            int temp = top;
```

```

        while(temp!=-1)
        {
            stack2[++top2] = stack[temp--];
        }
    }

    tag = bt;
    bt = NULL;
}

}

}

// 当前在两个栈中分别存储了p和q的祖先节点
for(int i = 0;i<=top1;i++)
{
    for(int j =0;j<=top2;j++)
    {
        if(stack1[i]==stack[j])
        {
            return stack[i];
        }
    }
}

return null;
}

```

代码题 24

题干

- 题目：

• 24、层次遍历

- 总结：

- 第一步：初始化队列，将根节点入队
- 第二步：进入循环，如果队头不等于队尾，执行循环
- 第三步：在循环内，出队队头元素，输出其值，判断其左右结点是否存在
 - 如果左子树存在，入队；
 - 如果右子树存在，入队

解答

```

void fun(BTNode* bt)
{
    BTNode* p = bt;
    BTNode* queue[maxsize];

    int front = 0;

```

```

int rear = 0;

if(bt!=null)
{
    queue[rear++] = p;

    while(rear !=front)
    {
        // 将当前队列的队头出队，并分析其是否有左右结点
        p = queue[front++];
        std::cout<<p->data;

        // 如果当前p的左右有子节点，将其入队
        if(p->lchild!=null)
        {
            queue[rear++] = p->lchild;
        }
        if(p->rchild!=null)
        {
            queue[rear++] = p->rchild;
        }
    }
}
}

```

代码题 25

题干

- 题目：
 - 25、试给出自下而上从右到左的层次遍历
- 总结：
 - 在对树进行层次遍历的同时，将每个入队的元素都放入栈中。最后从上往下输出栈中元素；

解答

```

typedef struct
{
    BTreeNode* data[maxsize];
    int top;
}stack

void fun(BTreeNode* bt)
{
    BTreeNode* queue[maxsize];
    int rear = 0;
    int front = 0;

```

```

stack st;
st.top = -1;

if(bt!=null)
{
    queue[rear++] = bt->data;
    while(rear!=front)
    {
        // 队头出队
        bt = queue[front++];
        st->data[++top] = bt->data;

        if(bt->lchild!=null)
        {
            queue[rear++] = bt->lchild;
        }
        if(bt->rchild!=null)
        {
            queue[rear++] = bt->rchild;
        }
    }
}

for(int i =top;i>=0;i--)
{
    cout<<st->data[i];
}
}

```

代码题 26

题干

- 题目：

26、求解二叉树的宽度

- 总结：

- 在队列中增加一个属性，用于记录当前层次的结点所拥有的个数；
- 进行层次遍历时、如果为进入下一层，则一直执行此命令；
- 最后遍历此属性，则可知道哪一种最常见；

解答

```

typedef struct
{
    BTreeNode *p;
    int lno;

}St[maxsize]

```

```

int width(BTNode* bt)
{
    BTNode* p = bt;
    BTNode* queue[maxsize];

    int front = 0;
    int rear = 0;

    int max = 0;

    if(bt!=null)
    {
        queue[rear++] = p;

        // 根节点这一层的深度为1
        lno = 1;

        while(rear !=front)
        {
            // 将当前队列的队头出队，并分析其是否有左右结点
            p = queue[front++];

            // 当前深度
            int Lno = p.lno;

            // 如果当前p的左右有子节点，将其入队
            if(p->lchild!=null)
            {
                queue[rear++] = p->lchild;

                // 下一层结点的深度加1
                queue[rear].lno = Lno+1;
            }
            if(p->rchild!=null)
            {
                queue[rear++] = p->rchild;
                queue[rear].lno = Lno+1;
            }
        }
    }

    int level = 1;
    int n = 0;
    // 遍历所有结点，计算哪层最宽
    for(int i =1;i<rear;i++)
    {
        if(queue[i].lno==level)
        {
            n++;
        }
    }
}

```

```

    }
    else
    {
        level = queue[i].lno;
        // 重新开始计数
        n=0;
    }

    if(n>max)
    {
        max = n;
    }
}
}

```

代码题 27

题干

- 题目：
- 27、用层次遍历求解二叉树的高度

- 总结：

解答

```

int fun(BTNode* bt)
{
    BTNode* p = bt;
    BTNode* queue[maxsize];

    int front = 0;
    int rear = 0;

    int level = 0; // 高度
    int last = ; // 当前结点的下一层的最右结点下标

    if(bt!=null)
    {
        queue[rear++] = p;

        while(rear !=front)
        {
            // 将当前队列的队头出队，并分析其是否有左右结点
            p = queue[front++];

            // 如果当前p的左右有子节点，将其入队
            if(p->lchild!=null)

```

```

        {
            queue[rear++] = p->lchild;
        }
        if(p->rchild!=null)
        {
            queue[rear++] = p->rchild;
        }

        if(front == last)
        {
            level++;
            // last指向当前队列的队尾，因为其为下一层的最右边结点
            last = rear;
        }
    }
}

return level;
}

```

代码题 28

题干

- 题目：

28、判断二叉树是否为完全二叉树

- 总结：

- 完全二叉树如果按照层次遍历，将所有结点存储在数组中，则符合二叉树在数组中的存储规则，并且因为是完全二叉树，因此在数据中应当在最后一个结点之前、无空缺位置，即位置全部填满；
- 所以可以在出现空结点时判断，当前队列后背是否都是空结点。如果后背还有结点，则肯定不是完全二叉树。如果后背都是空结点，则此时此结点是最右下角的结点；

解答

```

bool IsFun(BTNode* bt)
{
    BTNode* queue[maxsize];
    int rear = 0;
    int front = 0;

    if(bt==true)
    {
        return true;
    }
    queue[rear++] = bt;

    while(rear!=front)

```

```

{
    bt = queue[front++];

    if(bt!=null)
    {
        queue[rear++] = bt->lchild;
        queue[rear++] = bt->rchild;
    }
    else // 当前为一个空结点，一个个出队、检查队列后背是否都是空结点。如果后背还有结
    点，则肯定不是完全二叉树
    {
        while(front!=rear)
        {
            // 从当前空结点继续
            bt = queue[front++];

            if(bt!=null)
            {
                return false;
            }
        }
    }

    return true;
}

```

代码题 29

题干

- 题目：
 - 29、计算二叉树的带权路径长度（叶子节点）
- 总结：
 - 当左右为空时，当前结点为叶子节点，将深度和权值相乘，返回上一层；

解答：方法一 - 递归

```

int GetWPL(BTNode* bt,int deep)
{
    if(bt==null)
    {
        return 0;
    }
    if(bt->lchild==null&&bt->rchild==null)
    {
        return deep*(bt->data);
    }
}

```

```

int A = GetWPL(bt->lchild,deep+1);
int B = GetWPL(bt->rchild,deep+1);

return A+B;
}

```

解答：方法二 - 层次遍历

```

int GetWPL(BTNode* bt)
{
    BTNode* queue[maxsize];
    int rear = 0;
    int front = 0;

    int level = 0;
    int last = 1;

    int wpl = 0;

    if(bt==null)
    {
        return 0;
    }

    queue[rear++] = bt;

    while(rear!=front)
    {
        bt = queue[front++];

        if(bt->lchild==null&&bt->rchild==null)
        {
            wpl += (bt->data)*deep;
        }
        if(bt->lchild!=null)
        {
            queue[rear++] = bt->lchild;
        }
        if(bt->rchild!=null)
        {
            queue[rear++] = bt->rchild;
        }
        if(front==last)
        {
            deep++;
            last = rear;
        }
    }
    return wpl;
}

```

代码题 31

题干

- 题目：
 - **31、建立中序线索二叉树**
- 总结：
 -

解答

```
typedef struct TNode
{
    char data;
    struct TNode* lchild,rchild;
    int ltag = 0;
    int rtag = 0'
}TNode,iTree*

void fun(TNode* &p,TNode* &pre)
{
    if(p!=null)
    {
        fun(p->lchild,pre);

        if(p->lchild == null)
        {
            p->ltag = 1;
            p->lchild = pre;
        }

        if(pre!=null&&pre->rchild == null)
        {
            pre->rtag = 1;
            pre->rchild = p;
        }

        pre = p;
        fun(p->rchild,pre);
    }
}
```

代码题 32

题干

- 题目：
 - **32、中序遍历线索二叉树**
- 总结：

解答

```
typedef struct TNode
{
    char data;
    struct TNode* lchild,rchild;
    int ltag = 0;
    int rtag = 0;
}TNode, iTree*

// 获得当前子树的最前结点，即中序遍历时、此子树最前驱的结点
TNode* GetFirst(TNode* bt)
{
    while(bt->ltag == 0)
    {
        bt = bt->lchild;
    }
    return bt;
}

// 沿着前驱、后继关系往后走
TNode* Next(TNode* bt)
{
    // 如果当前子树右tag为0，则表示其不是当前最前驱，在当前子树中使用GetFirst，找到最前驱
    if(bt->rtag == 0)
    {
        return GetFirst(bt->rchild);
    }
    else if(bt->rtag == 1)
    {
        return bt->rchild;
    }
}

void In(TNode* bt)
{
    TNode* q = GetFirst(bt);
    for(;q!=null;q = Next(q))
    {
        cout<<q->data;
    }
}
```

代码题 33

题干

- 题目：

33、先序建立二叉搜索树并先序遍历

- 总结:

解答

代码题 34

题干

- 题目:

34、寻找中序线索二叉树的前驱结点

- 总结:

解答

```
TNode* fun(TNode* bt)
{
    if(bt->ltag == 1)
    {
        return bt->lchild;
    }
    else if(bt->ltag==0) // 当前bt左孩子不是其前驱，因此在其左子树中、寻找前驱
    {
        p = p->lchild;
        // 找到其 p->rtag为1的结点，此结点的后继就是当前目标结点，因此此节点也是目标结点的
        前驱
        while(p->rtag == 0)
        {
            p = p->rchild;
        }
        return p;
    }
}
```

代码题 35

题干

- 题目:

35、用孩子兄弟表示法求树所有叶子结点个数

- 总结:

解答

```
int fun(BTNode* bt)
{
    if(bt==null)
    {
        return 0;
    }

    // 孩子兄弟表示法中，右孩子为空时、为 叶子节点
    if(bt->lchild==0)
    {
        return fun(p->rchild)+1;
    }

    return fun(bt->lchild) + fun(bt->rchild);
}
```

获取笔记持续更新可加微信: Wanda