对于结点类型为LNode的单链表，编写算法，统计出单链表中结点值等于给定值x的结点个数。

函数原型：int Count(LNode *HL, ElemType x)

```
int Count(LNode *HL, ElemType x){
  LNode *p = HL -> next;
  int count = 0;
  while (p!= NULL){
    if(p->data == x)
      count++;
    pp->next;
  }
  return count;
}
```

查找二叉树值为x的结点。

```
// 查找二叉树中值为x的结点
/*
 ①判断根节点是否为空
 ②否则，若根节点即为找到的值，返回根节点
 ③否则，  p= 递归左孩子
         若p不为空，返回p
         否则，递归有孩子

 */
BTNode *FindNode(BTNode *b,ElemType x)
{
    BTNode *p;
    if (b==NULL)
        return NULL;
    else if (b->data==x)
        return b;
    else
    {
        p=FindNode(b->lchild,x);
        if (p!=NULL)
            return p;
        else
            return FindNode(b->rchild,x);
    }
}
```

求二叉树的高度

```
int BTHeight(BTNode *b)
{
    int lchildh,rchildh;
    if (b==NULL) return(0);                 //空树的高度为0
    else
    {
        lchildh=BTHeight(b->lchild);    //求左子树的高度为lchildh
        rchildh=BTHeight(b->rchild);    //求右子树的高度为rchildh
        return (lchildh>rchildh)? (lchildh+1):(rchildh+1);
    }
}
```

打印二叉树

```
void DispBTree(BTNode *b)
{
    if (b!=NULL)
    {    printf("%c",b->data);
        if (b->lchild!=NULL || b->rchild!=NULL)
        {    printf("(");                          //有孩子节点时才输出(
            DispBTree(b->lchild);                 //递归处理左子树
            if (b->rchild!=NULL) printf(",");    //有右孩子节点时才输出,
            DispBTree(b->rchild);                 //递归处理右子树
            printf(")");                          //有孩子节点时才输出)
        }
    }
}
```

输出叶子结点

```
void DispLeaf(BTNode *b)
{
    if (b!=NULL)
    {
        if (b->lchild==NULL && b->rchild==NULL)
            printf("%c ",b->data);    //访问叶子节点
        DispLeaf(b->lchild);        //输出左子树中的叶子节点
        DispLeaf(b->rchild);        //输出右子树中的叶子节点
    }
}
```

先序遍历非递归算法

```
// 算法思想1
将根节点进栈
while(栈不空){
  出栈结点p并访问
  若p有右孩子，将右孩子进栈
  若p有左孩子，将左孩子进栈
}

// 算法思想2
p=b;
while(栈不空||p!=NULL){
  while(p!=NULL){
    访问结点p; 进栈;
    p = p->lchild;
  }
  if (栈不空){
    出栈p;
    p=p->rchild;
  }
}
```

二叉树的层次遍历

```
// 层次遍历
void levelOrder(BTNode *b){
    BTNode *p;
    SqQueue *qu;      // 定义环形队列指针
    InitQueue(qu);    // 初始化队列
    enQueue(qu,b);    // 根节点入队
    while (!queueEmpty(qu))// 队列不为空
    {
        dequeue(qu,p); // 出队p
        visit p;       // 访问p
        if (p->lchild != NULL) {
            enQueue(qu,p->lchild); // 左孩子进队
        }
        if (p->rchild != NULL) {
            enQueue(qu,p->rchild); // 右孩子进队
        }
    }
}
```

编写在以BST为树根指针的二叉搜索树上进行查找值为item的结点的非递归算法，若查找成功能则由item带回整个结点的值并返回true,否则返回false.

函数原型bool Find(BTreeNode *BST,ElemType &item)

```
bool Find(BTreeNode *BST,ElemType &item){
  while (BST != NULL){
    if(item == BST->data)
      return ture;
    else if(item < BST -> data)
      BST = BST -> left;
    else
      BST = BST -> right;
  }
}
```

求二叉树所有叶子结点值之和

函数原型int KeySum(BTree T)

```
int KeySum(BTree T){
  int sum = 0;
  if (T != NULL){
    if (T->lchild == NULL && T->rchild == NULL)
      sum += T->data;
    KeySum(T->lchild);
    KeySum(T->rchild);
  }
  return sum;
}
```

判别二叉树是否为二叉排序树

算法思想：二叉排序树为中序递增有序序列，对给定的二叉树进行中序遍历，若始终能保持前一个值比后一个值小，则说明是二叉排序树。

```
int flag = 1, last = 0;
int is_BSTree(BTree *T){
  if(T->lchild && flag) is_BSTree(T->lchild);
  if(T->data < last) flag = 0;
  last = T->data;
  if(T->rchild && flag) is_BSTree(T->rrchild);
  return flag;
}
```

2013期末

编写递归算法，求二叉树位于先序序列中第K个位置结点的值。

注：二叉树的结点类型如下

Struct Bitree

int data;

struct Bitree *lchild, *rchild;

};

```
int c=0,k;
int GetPreSeq(Bitree *T){
  if(T){
    c++;
    if(c==k) return T->data;
    else{
      GetPreSeq(T->lchild);
      GetPreSeq(T->rchild);
    }
  }
}
```