```
typedef struct LNode *PtrToLNode;
struct LNode {
    ElementType Data;
    PtrToLNode Next;
};
typedef PtrToLNode Position;
typedef PtrToLNode List;

/* 查找 */
#define ERROR NULL

Position Find( List L, ElementType X )
{
    Position p = L; /* p指向L的第1个结点 */

    while ( p && p->Data!=X )
        p = p->Next;

    /* 下列语句可以用 return p; 替换 */
    if ( p )
        return p;
    else
        return ERROR;
}

/* 带头结点的插入 */
/*注意:在插入位置参数P上与课程视频有所不同，课程视频中i是序列位序（从1开始），这里P是链表结点指针，在P之前插入新结点 */
bool Insert( List L, ElementType X, Position P )
{ /* 这里默认L有头结点 */
    Position tmp, pre;

    /* 查找P的前一个结点 */
    for ( pre=L; pre&&pre->Next!=P; pre=pre->Next ) ;
    if ( pre==NULL ) { /* P所指的结点不在L中 */
        printf("插入位置参数错误\n");
        return false;
    }
    else { /* 找到了P的前一个结点pre */
        /* 在P前插入新结点 */
        tmp = (Position)malloc(sizeof(struct LNode)); /* 申请、填装结点 */
        tmp->Data = X;
        tmp->Next = P;
        pre->Next = tmp;
        return true;
    }
}

/* 带头结点的删除 */
/*注意:在删除位置参数P上与课程视频有所不同，课程视频中i是序列位序（从1开始），这里P是拟删除结点指针 */
bool Delete( List L, Position P )
{ /* 这里默认L有头结点 */
    Position tmp, pre;

    /* 查找P的前一个结点 */
    for ( pre=L; pre&&pre->Next!=P; pre=pre->Next ) ;
    if ( pre==NULL || P==NULL) { /* P所指的结点不在L中 */
        printf("删除位置参数错误\n");
        return false;
    }
    else { /* 找到了P的前一个结点pre */
        /* 将P位置的结点删除 */
        pre->Next = P->Next;
        free(P);
        return true;
    }
}
```