**线性表顺序表实现的各种操作(SqList)**

**初始化、求表长、按值查找、按位查找、增、删、改、查、插入、判空、销毁、其中比较重要的的操作是按位插入和按位删除，这里面需要将元素依次向后移动和依次向前移动。**

**线性表链表实现的各种操作(LinkList)**

1. **单链表：头插法创建单链表、尾插法创建单链表、按序号查找节点、按值查找节点、插入节点操作（前插和后插）、删除节点、求表长。这里要区分一下带头节点和不带头节点的链表的区别，带头节点的链表在前插操作中要单独设立在第一个位置插入节点的条件，因为链表的遍历是从头节点的下一个位置开始的。**
2. **双链表：双链表的创建、双链表删除即诶单**
3. **循环链表：循环单链表、循环双链表、静态链表。**
4. **静态链表**

**栈的线性表实现以及各种操作**

1. **创建、判空、判满，出栈、入栈，获取栈顶元素的值，修改栈顶元素的值。**
2. **需要注意的是栈顶指针的位置，栈顶指针有两种初始指向方式，一种是指向-1，一种是指向1，前者的栈顶指针指向的元素是空节点-1的位置，后者则直接指向可存储的空节点。换句话讲就是，如果top在声明的时候指向是-1，那么在栈的操作过程中top指针所指向的就是真正的栈顶元素，而top在声明的时候指向的是0，那么在操作过程中top所指向的是栈顶元素的后一个位置。**
3. **共享栈，共享栈是两个栈共享同一片存储空间，在创建时设计两个top指针，一个top1指向存储空间的起始位置，一个top2指向存储空间的结束位置。在使用时指针的值都往中间靠拢。通过比较两个top指针的值就可以判断是否栈满，按理说top1的值应该一直小于等于top2的值**

**栈的链表实现以及各种操作**

1. **创建、初始化、判空、出栈、入栈，获取栈顶元素的值，修改栈顶元素的值，链栈不需要判空。**
2. **链栈的实现就是通过头插法创建链表的形式进行创建的，这里的链表可以带头节点或者不带头节点都可以，只是在判空的时候需要注意，带头节点的要判断头节点的Next指针的指向是否为NULL，不带头节点的直接判断链栈指针的指向是否尾NULL。**

**3.引用和解引用的一个问题：对于结构体变量LNode L，对应的结构体指针变量是LNode \*P,可以有这段代码成立，LNode \*P = &L，在使用时，单独的P就是一个指针变量，存放的是L的地址，在函数的参数传递中，P也是作为一个地址进行传入（可以理解为就是一串数字），实际上函数的内部是将P复制一份在函数内部进行操作，所以不管在内部对复制的P进行何种的操作，在函数调用结束时都不会影响原来的P的值。如果想改变P的值就需要对定义一个P的指针，LNode \*\*S = &P ，S是P的一个指针变量，在函数内部对S进行解引用操作\*S，对\*S进行操作就是直接对P的实际内存地址进行操作，就可以真正改变指针变量P中的值。（总结起来就是，想要改变谁的值，就要在函数传参数的时候传递谁的指针，通过解引用操作再改变谁的值）**

**“&p”是引用，可以取得变量p的内存地址，“\*p”是解引用，是通过地址p去找，在p中的内存数据，->也是解引用的一种方式**

**<http://t.csdn.cn/LWfns> 关于栈的一个实现**

**<http://t.csdn.cn/jJzQt> 关于C语言中函数值传递和引用传递的问题**

**队列的顺序表实现**

1. **队列的顺序表实现要考虑的操作有：初始化队列、计算队列的中的元素个数、判空、判满，进队、出队，读取队头元素，一般实现循环队列，无法循环的队列只能进行有限次进出队操作，通过取模运算进行队列的循环。**
2. **对于物理结构中只含有对头指针和队尾指针的循环队列，初始方式有两种，一般队头指针front指向的是当前队列的头元素，区别在于队尾指针的指向，是指向当前的队尾元素还是队尾元素的下一个元素（分别对应队尾指针初始指向0和-1），队尾指针的指向不同影响着入队、计算对内元素数这两种方法的实现，需要注意。**
3. **这里面比较核心的函数操作我觉得是如何计算当前队内的元素的数量，计算出数量就可以判断队内是否已满，进而可以判断是否可以进行入队操作，如果不对队列物理结构进行改变，在rear指针初始指向为0的情况下，队内元素的数量是x=(SqQ->rear+MaxSize - SqQ->front)%MaxSize，如果在rear指针初始指向为-1的情况下，队内元素的数量是x=(SqQ->rear+1+MaxSize - SqQ->front)%MaxSize。**
4. **分析这个计算方法，在第一种情况下，如果队尾指针没有进行循环，可以判定对头指针也没有进行循环，队头指针的值必定小于队尾指针，则队内元素的数量就是SqQ->rear- SqQ->front，但是当队尾指针进行循环，而对头指针没有进行循环，这个时候队头指针的值就大于队尾指针的值，这个值是物理内存上的值，所以要将队尾指针的值恢复到逻辑上的值，因为在循环的时候队尾指针要队MaxSize取模运算，相当于减去一个MaxSize，所以在作差的时候要在加回来，作差结束后在取模就可以得到当前队内元素的数量。**
5. **这里也要注意一个很重要的问题，如果要判断队列是不是空的，那就要判断对头指针和队尾指针的相对位置，如果rear指针的初始指向为0，队尾指针和队头指针指向的位置是相同的，那么队列为空，但这种情况下判断队列是满的就要牺牲一个内存空间，因为进队操作导致的rear指针的循环也会是rear指针和front指针指向同一个位置，所以rear指针在逻辑上指向front指针的后一个位置时，就要判断队列已满，但这个时候队列中的元素只有MaxSize-1个，所以就牺牲了一个内存空间。在rear指针初始指向-1的时候也是同理，当rear指针在逻辑上指向front指针的后面第二项的时候，整个队列已经满了，因为如果rear指针再增加一位，就会和判空的条件相同。**

**6.书上还给了另外两种实现的方法，可以避免浪费这一个存储的空间，第一种在定义循环队列的物理结构的时候就增加一个计数变量，用来计算当前队列中的元素个数，进队一次，计数加一，出队一次，计数减一，这样在判空的时候就先判断两个指针的指向，如果rear指针初始指向为0，那么当两个指针指向同一个节点的时候就代表此时队列是空的或者是满的，此时再判断队内的元素数量是否为0，这样就可以判断出是否为空，判满操作也是同理。第二种是在定义物理结构的时候加上一个tag标签，tag标签初始为0，进队的时候将标签置为1，出队的时候将标签置为0，在rear初始指向为0的情况下，如果rear指针和front指针指向是相同的，那么判断一下tag标签的数量，如果tag为0，就意味着上一次导致两指针指向同一位置的操作要么是出队，要么就是队列刚刚初始化，这个时候队列必然为空，反之如果tag为1，上次导致两指针相同的操作是进队，所以这个时候队列必然是满的。（注：使用这两种方式实现循环队列在计算队列长度的时候也要注意两种情况，因为队满和对空上面的公式的值都是0，所以还要单独判断一下）。**