1. 线性表
2. 线性表的逻辑结构
3. 定义：n个数据元素的有限序列，能够进行遍历
4. 特点：
   1. 除了第一个元素外，每个元素都有且仅有一个直接前驱
   2. 除了最后一个元素外，每个元素有且仅有一个直接后继
5. ADT基本操作（设计与实现的分离）
   1. 初始化与销毁（赋值与析构）
   2. 引用与查看（不增加与减少节点结构，但可以改变元素的值）

Get Elem( int position , Type &e ) const

//使用&进行引用传递地址可以直接改变元素的值

//增加const 后缀之后则不能改变元素的值，只能查看

* 1. 加工与处理（改变了节点结构与数量等）
  2. 遍历与访问

Void Traverse( void( \*visit ) , (Type &) )

//使用函数指针\*visit可以便于抽取公共部分，减少函数冗余

//安全指针问题：为了避免野指针的出现，我们在初始化指针与销毁指针时，可以将指针赋值为null，使用时可以先判断是否为null，从而实现指针的安全使用。

1. 线性表的顺序存储结构：顺序表
2. 将线性表的元素存储在一段连续的存储空间中，在C语言中的实例是数组

//可能存在的优化：将顺序表的容量设为可变，通过在构造函数中传参实现

1. Attention：

在线性表插入与删除的过程中，可能会产生数值溢出，即要删除的元素位序已经大于元素个数，可以通过每次循环删除首元素来避免溢出。 删除过程为从左到右，插入过程为从右到左的循环，因为删除时需要将删除位置后的元素向左移动而插入时候需要向右移动

1. 顺序表的求交集运算：用c表示a与b的集合的交集，在a中取出元素，在b中寻找是否有匹配，有匹配则将其插入c
2. 线性表的链式存储结构：链表
3. 单链表
   1. 定义：单链表是最简单的链式存储结构，每个数据元素用一个节点node来存储，一个是存储数据的数据域data，另一个是存储指向下一个节点指针的指针域next

//空头节点存在的意义：此头结点始终存在，即使链表为空也存在，使得指针p永远不会指向空，从而不需要对p进行判断是否为空，节约了n条if语句的判断

* 1. 特点：
     1. 线性结构
     2. 每个节点可以不连续存储
     3. 链表的空间是可以扩充的

（但是当数据量较大时，可能需要处理new新节点空间不够的问题）

* 1. 通过类模板来更方便地实现各个功能：
     1. 插入：temp ptr -> next = new Node<Type>(e , tmp ptr -> next)

//PC寄存器（Programme Counter）：用于存储当前正在执行的程序，运行结束后还会自动跳转到下一个要执行的语句。

* + 1. 删除：在删除时把中间指针保留一下，否则会再也无法找到它

//内存泄露：为指针分配空间后未及时释放，而再也无法找到它，便产生了内存泄露，即这样一块内存被浪费了，最后可能会发生内存不足系统崩溃

1. 循环链表：最后一个节点的指针域指向第一个节点的数据域
2. 双向链表：每一个节点均有前驱指针和后继指针（左链与右链）
3. 链表空间的分配优化
4. new与delete的分配速度较慢：

因为使用了堆空间分配，而在释放后内存空间不连续，在以后的分配过程中，每次都需要重新寻找一块能够装得下的空间，故速度较慢。

1. 使用free list来实现空间换时间：

当链表中进行delete时，通过运算符的重载，将需要删除的节点连在另外的一个链表free list中，然后之后需要new时，就从此free list中取值，从而加快运行速度。但是由于delete掉的节点并没有被释放，而是存在了free list中，故牺牲了一定的空间。

1. 使用异或链表来实现时间换空间：

每个节点的指针域存入前驱节点与后继节点地址的异或，由前一个节点访问到此节点后，可以通过前一个节点的地址与当前节点指针域的数据取异或，就可以随意得到前驱地址或后继地址。相比于双向链表，异或链表牺牲了时间来进行一次异或运算，但是节约了一个指针域的空间。