1. 链表的基本概念
   1. 链表引出
      1. 数组有缺陷
      2. 静态空间，一旦分配内存就不可以动态扩展，要不分配不够，要不分配过多
      3. 对于数组头部进行插入和删除效率低
   2. 链表的组成
      1. 链表是由节点组成的
      2. 节点由 数据域 和 指针域组成
      3. struct LinkNode { int num ; struct LinkNode \* next; }
   3. 链表的分类
      1. 方式1 静态链表 动态链表
      2. 方式2 单向链表 双向链表 单向循环链表 双向循环链表
2. 静态链表和动态链表
   1. 静态链表 创建在栈上
   2. 动态链表 创建在堆区
3. 链表的基本使用
   1. 带头节点链表 好处在于 头节点永远都是固定的
   2. 初始化链表 struct LinkNode \* pHeader = init\_LinkList ()
   3. 遍历链表 void foreach\_LinkList( struct LinkNode \* pHeader )
   4. 插入链表 void insertLinkList( struct LinkNode\*pHeader , int oldval ,int newval )
      1. 在oldval前插入 newVal，如果没有oldval就进行尾插
   5. 删除链表 void delete\_LinkList( struct LinkNode\*pHeader, int val )
      1. 用户提供的有效数据 删除掉
      2. 无效数据 直接return
   6. 清空链表
      1. void clear\_LinkList(struct LinkNode\*pHeader )
      2. 将所有有数据的节点释放掉
   7. 销毁链表
      1. void destroy\_LinkList(struct LinkNode\*pHeader)
      2. 将整个链表都释放掉
4. 函数指针的定义
   1. 先定义出函数类型，再通过类型定义出函数指针
      1. typedef void(FUNC\_TYPE)();
      2. FUNC\_TYPE \* pFunc = func;
   2. 先定义出函数指针类型，再定义函数指针
      1. typedef void(\*FUNC\_TYPE)();
      2. FUNC\_TYPE pFunc = func;
   3. 直接定义函数指针变量
      1. void(\* pFunc )() = func;
   4. 函数指针和指针函数的区别
      1. //函数指针 是指向函数的 指针
      2. //指针函数 函数的返回值是一个指针的 函数
   5. 函数指针的数组定义
      1. void(\*pFunc[3])();
5. 函数指针做函数参数（ 回调函数 ）
   1. 提供一个通用的函数，可以打印任意的数据类型
6. 回调函数案例
   1. 提供一个函数，打印任意类型的数组
   2. 提供一个查找数组中元素的函数
7. 作业
   1. 提供一个函数，可以对任意类型的数组进行排序，排序规则 选择排序