1 链表

1.1 引出- 数组缺陷

1.1.1 数组是一个静态空间，一旦分配内存，就不可以动态扩展，空间可能分配多或者分配的少，操作不精准

1.1.2 对于头部的插入删除效率低

1.2 链表

1.2.1 由节点组成

1.2.2 而节点由 数据域 和 指针域组成

1.2.2.1 数据域是维护数据的

1.2.2.2 指针域 维护下一个节点的位置

1.2.3 链表可以解决数组的缺陷

1.2.4 链表的分类

1.2.4.1 静态链表、动态链表

1.2.4.2 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

2.1 静态链表分配在栈上

2.2 动态链表分配到堆区

2.3 实现链表的初始化以及遍历功能

3 链表基本使用

3.1 带头节点链表和不带头节点链表

3.1.1 带头好处：带着头节点的链表永远固定了头节点的位置

3.2 初始化链表 init_LinkList

3.3 遍历链表 foreach_LinkList

3.4 插入链表 insert_LinkList 利用两个辅助指针 实现插入

3.5 删除链表 delete_LinkList 利用两个辅助指针 实现删除

3.6 清空链表 clear_LinkList 将所有有数据节点释放掉，可以在使用

3.7 销毁链表 destroy_LinkList 将整个链表释放掉，不可以再使用

4 函数指针

4.1 函数名本质就是一个函数指针

4.2 可以利用函数指针 调用函数

5 函数指针定义方式

5.1 //1、先定义出函数类型，再通过类型定义函数指针

5.1.1 typedef void(FUNC_TYPE)(int, char);

5.2 //2、定义出函数指针类型，通过类型定义函数指针变量

5.2.1 typedef void(* FUNC_TYPE2)(int, char);

5.3 //3、直接定义函数指针变量

5.3.1 void(*pFunc3)(int, char) = func;

5.4 函数指针和指针函数

5.4.1 //函数指针 指向了函数的指针

5.4.2 //指针函数 函数返回值是指针的函数

5.5 函数指针数组

5.5.1 `void(*pArray[3])();`

6 函数指针做函数参数（回调函数）

6.1 利用回调函数实现打印任意类型数据

6.2 提供能够打印任意类型数组函数

6.3 利用回调函数 提供查找功能

7 作业：超难

7.1 提供一个函数，实现对任意类型的数组进行排序，排序规则利用选择排序，排序的顺序 用户可以自己指定

8