# 面试学习笔记

## 实现memcpy的功能

char\* tempdest=(char\*)dest;

const char\* tempsrc=(const char\*)src;

while (num--) {

\*tempdest=\*tempsrc;

tempsrc=tempsrc+1;

tempdest=tempdest+1;

}

自己实现内存拷贝的功能,很容易忽略到内存重叠的情况,这也是为什么莫名其妙会发生报错的原因.内存重叠的情况如图所示:

1.src和dest所指内存区域不能重叠。

　　    2.与strcpy相比，memcpy并不是遇到'\0'就结束，而是一定会拷贝完n个字节。

           3.memcpy用来做内存拷贝，你可以拿它拷贝任何数据类型的对象，可以指定拷贝的数据长度。

## 成员在结构体中的内存偏移量

(struct\_type \*) 0

把一段地址看作struct\_type；

(struct\_type \*) 0)->member

指向它的member成员

&((struct\_type \*) 0)->member

取这个成员的地址

(size\_t) &((struct\_type \*) 0)->member

把地址硬转为size\_t类型

## hash 表存储实现

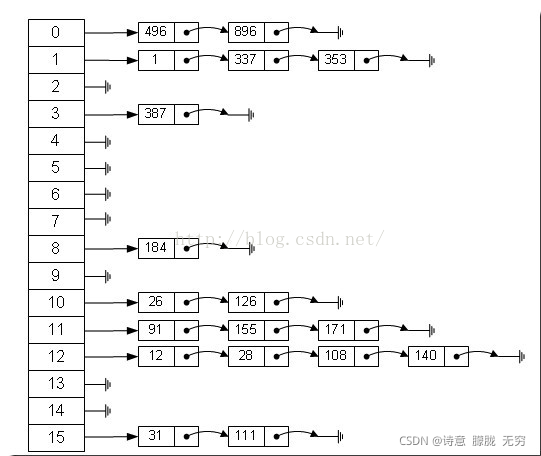
哈希表（Hash table，也叫散列表），所谓hash表，就是以 键-值(key-indexed) 的形式存储的数据结构。可以根据key来快速的查找到value。也就是说，它通过把key值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表

### hash表怎么存储

hash表把key通过hash函数转化成一个特定的整数，然后与数组的长度取余，取余结果（hash值）当做该数组的下标，然后将value存储在该下标中。hash(key)%len。

### 解决hash冲突

对于第一个问题，就是我们常说的hash冲突。有开放定制法和拉链法。



哈希表是由数组+链表组成

左边很明显是 数组，数组的每个成员包括一个指针，指向一个链表的头，当然这个链表可能为空，也可能元素很多。我们根据元素的一些特征把元素分配到不同的链表中去，也是根据这些特征，找到正确的链表，再从链表中找出这个元素。

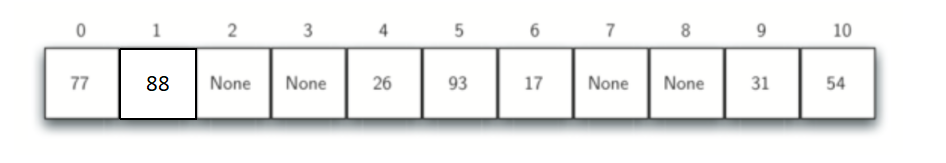
从上图我们可以发现哈希表是由数组+链表组成的，一个长度为16的数组中，每个元素存储的是一个链表的头结点。那么这些元素是按照什么样的规则存储到数组中呢。一般情况是通过hash(key)%len获得，也就是元素的key的哈希值对数组长度取模得到。比如上述哈希表中，12%16=12,28%16=12,108%16=12,140%16=12。所以12、28、108以及140都存储在数组下标为12的位置。

开放定制法

基本原理为，使用大小为M的数组来保存N个键值对，其中M>N，我们需要使用数组中的空位解决碰撞冲突

比如我们要存放88，哈希值是0，数组这个位置已经有值了，那我们再获取一个哈希值，比如在原哈希值的基础上加1，得到1，1的位置是空，我将88放进去。

插入88后的数组变为



冲突解决了，但我们读取数据的时候，好像又出现问题了，88的哈希值是0，发现数组0位置不是空的，那我们确定88在哈希表中？肯定不行，0这个位置存储的是77，不是88。我们的解决方法是判断0这个位置的值是不是88，不是的话，再计算88的哈希值是1，判断是1这个位置是否为空，为空，则88不在哈希表中；不为空，判断值是否为88，若是88，确定在哈希表中；如果值不是88，我们则继续计算哈希值是2，依次下去，直到找到88或者值为空的位置。

### Hash的应用

1、Hash主要用于信息安全领域中加密算法，它把一些不同长度的信息转化成杂乱的128位的编码,这些编码值叫做Hash值. 也可以说，Hash就是找到一种数据内容和数据存放地址之间的映射关系。

2、查找：哈希表，又称为散列，是一种更加快捷的查找技术。我们之前的查找，都是这样一种思路：集合中拿出来一个元素，看看是否与我们要找的相等，如果不等，缩小范围，继续查找。而哈希表是完全另外一种思路：当我知道key值以后，我就可以直接计算出这个元素在集合中的位置，根本不需要一次又一次的查找！

举一个例子，假如我的数组A中，第i个元素里面装的key就是i，那么数字3肯定是在第3个位置，数字10肯定是在第10个位置。哈希表就是利用利用这种基本的思想，建立一个从key到位置的函数，然后进行直接计算查找。

3、Hash表在海量数据处理中有着广泛应用。

## 软总线

软总线之发现连接：从手动发现，进化成自发现

软总线提出了异构网络组网可以很好解决设备间不同协议如何交互的问题。

软总线可以自动构建一个逻辑全连接网络，用户或者业务开发者无需关心组网方式与物理协议。

## TCP/IP协议、三次握手、四次挥手

https://blog.csdn.net/Q0717168/article/details/111933810?ops\_request\_misc=&request\_id=&biz\_id=102&utm\_term=TCPIP%20%E6%8F%A1%E6%89%8B%E6%8C%A5%E6%89%8B&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~sobaiduweb~default-0-111933810.142^v88^control\_2,239^v2^insert\_chatgpt&spm=1018.2226.3001.4187

①客户端向服务端发送TCP报文：

标志位为FIN=1，表示“请求释放连接”，序列号为seq=u，随后客户端进入FIN-WAIT-1状态并停止向服务端发送数据

②服务端收到从客户端发出的释放连接的TCP报文后结束ESTABLISHED状态，进入CLOSE-WAIT状态并返回一段TCP报文：

标志位为ACK=1，表示“收到释放请求”，序列号为seq=v，确认号ack=u+1，表示将来自客户端报文序号seq值加1作为自己的确认号ack的值，随后服务端进入CLOSE-WAIT状态

③客户端收到服务端确认结果后，进入FIN-WAIT-2状态。服务端做好了释放连接准备后再次向客户端发出一段TCP报文：

标志位为FIN=1，ACK=1，表示“已准备好释放”，序号为seq=w，确认号ack=u+1，服务器进入LAST-ACK状态并停止向客户端发送数据

④客户端收到回复后，结束FIN-WAIT-2状态，进入TIME-WAIT状态，并向服务端发送一段报文：

标志位为ACK=1，表示“已收到准备释放信号”，序列号为seq=u+1，确认号为ack=w+1，表示将来自服务端报文序号seq值加1作为自己的确认号ack的值，客户端进入TIME-WAIT状态，客户端经过2MSL后进入CLOSE状态；服务器收到确认后，立刻进入CLOSE状态

（MSL指的是Maximum Segment Lifetime：一段TCP报文在传输过程中的最大生命周期。2MSL即是服务端发出为FIN报文和客户端发出的ACK确认报文所能保持有效的最大时长）

派生函数重写加不加override的区别

重载运算符

指针和引用的区别

声明的定义的区别

析构函数是否是虚函数

函数后面加const表示什么

vector和链表区别

unordered和mapde 底层原理

快排以及非递归写法

智能指针