 事实上，仔细看一下free()的函数原型，也许也会发现似乎很神奇，free()函数非常简单，只有一个参数，只要把指向申请空间的指针传递 给free()中的参数就可以完成释放工作！这里要追踪到malloc()的申请问题了。申请的时候实际上占用的内存要比申请的大。因为超出的空间是用来记录对这块内存的管理信息。先看一下在《UNIX环境高级编程》中第七章的一段话： 大多数实现所分配的存储空间比所要求的要稍大一些，额外的空间用来记录管理信息分配块的长度，指向下一个分配块的指针等等。这就意味着如果写过一个已分配区的尾端，则会改写后一块的管理信息。这种类型的错误是灾难性的，但是因为这种错误不会很快就暴露出来，所以也就很难发现。将指向分配块的指针向后移动也可能会改写本块的管理信息。 以上这段话已经给了我们一些信息了。malloc()申请的空间实际我觉得就是分了两个不同性质的空间。一个就是用来记录管理信息的空间，另外一个就是可用空间了。而用来记录管理信息的实际上是一个结构体。在C语言中，用结构体来记录同一个对象的不同信息是 天经地义的事！下面看看这个结构体的原型： 程序代码： struct mem_control_block ...{ int is_available. //这是一个标记？ int size. //这是实际空间的大小 }. 对于size,这个是实际空间大小。这里其实我有个疑问，is_available是否是一个标记？因为我看了free()的源代码之后对这个变量感觉有点纳闷（源代码在下面分析）。这里还请大家指出！ 所以，free()就是根据这个结

构体的信息来释放malloc()申请的空间！而结构体的两个成员的大小我想应该是操作系统的事了。但是这里有一个问题，malloc()申请空间后返回一个指针应该是指向第二种空间，也就是可用空间！不然，如果指向管理信息空间的话，写入的内容和结构体的类型有可能不一致，或者会把管理信息屏蔽掉，那就没法释放内存空间了，所以会发生错误！（感觉自己这里说的是废话）好了！下面看看free()的源代码，我自己分析了一下，觉得比起malloc()的源代码倒是容易简单很多。只是有个疑问，下面指出！程序代码： void free(void *ptr) ...{ struct mem_control_block *free. free = ptr - sizeof(struct mem_control_block). free->is_available = 1. return. } 看一下函数第二句，这句非常重要和关键。其实这句就是把指向可用空间的指针倒回去，让它指向管理信息的那块空间，因为这里是在值上减去了一个结构体的大小！后面那一句free->is_available = 1.我有点纳闷！我的想法是：这里is_available应该只是一个标记而已！因为从这个变量的名称上来看，is_available 翻译过来就是“ 是可以用 ”。不要说我土！我觉得变量名字可以反映一个变量的作用，特别是严谨的代码。这是源代码，所以我觉得绝对是严谨的！！这个变量的值是1，表明是可以用的空间！只是这里我想了想，如果把它改为0或者是其他值不知道会发生什么事？！但是有一点我可以肯定，就是释放绝对不会那么顺利进行！因为这是一个标记！ 当然，这里可能还是有人会有疑问，为什么这样就可以释放呢？？我刚才也有这个疑问。后来我想到，释放是操作系统的事，那么就free()这个源代码来看，什么也没有释放，对吧？但是它确实是确定了管理信息的那块内存的内容。

所以，free()只是记录了一些信息，然后告诉操作系统那块内存可以去释放，具体怎么告诉操作系统的我不清楚，但我觉得这个已经超出了我这篇文章的讨论范围了。 那么，我之前有个错误的认识，就是认为指向那块内存的指针不管移到那块内存中的哪个位置都可以释放那块内存！但是，这是大错特错！释放是不可以释放一部分的！首先这点应该要明白。而且，从free()的源代码看，ptr只能指向可用空间的首地址，不然，减去结构体大小之后一定不是指向管理信息空间的首地址。所以，要确保指针指向可用空间的首地址！不信吗？自己可以写一个程序然后移动指向可用空间的指针，看程序会有会崩！ 