在良好的代码风格中，其中有一项要求就是，一个函数只做一件事情。如果该函数实现了多个功能，那基本上可以说这不是一个设计良好的函数。

今天看C库中的函数realloc。其原型是`void *realloc(void *ptr, size_t size);`函数说明如下：

realloc() changes the size of the memory block pointed to by *ptr* to *size* bytes. The contents will be unchanged to the minimum of the old and new sizes; newly allocated memory will be uninitialized. If *ptr* is NULL, the call is equivalent to **malloc(size)**; if *size*is equal to zero, the call is equivalent to ***free(ptr)***. Unless *ptr* is NULL, it must have been returned by an earlier call to **malloc**(), **calloc**() or **realloc**(). If the area pointed to was moved, a ***free(ptr)*** is done.

总结一下，有以下几种行为：

1. 与名字相符，真正的realloc，参数ptr和size均不为NULL，重新调整内存大小，并将新的内存指针返回，并保证最小的size的内容不变；
2. 参数ptr为NULL，但size不为0，那么行为就等于malloc(size)；
3. 参数size为0，则realloc的行为为free(ptr)；这时原有的指针已经被free掉，不能继续使用。而此时realloc的返回值为NULL。这意味着不检查realloc的返回值，直接使用，会导致crash。

看，一个简单C库函数，却赋予了三种行为，所以这个realloc并不是设计良好的库函数。估计也是为了兼容性，才容忍这个函数一直在C库中。虽然在编码中，realloc会提供一定的方便，但是也很容易引发bug。

下面就举两个例子，来说明一下。

1. realloc第一种行为引发的bug

```
1. void *ptr = realloc(ptr, new_size);
2. if (!ptr) {
3.     错误处理
4. }
```

这里就引出了一个内存泄露的问题，当realloc分配失败的时候，会返回NULL。但是参数中的ptr的内存是没有被释放的。如果直接将realloc的返回值赋给ptr。那么当申请内存失败时，就会造成ptr原来指向的内存丢失，造成泄露。

正确的处理应该是这样

```
1. void *new_ptr = realloc(ptr, new_size);
2. if (!new_ptr) {
3.     错误处理。
4. }
5. ptr = new_ptr
```

说明:

realloc 重新调整内存大小，

1：扩大内存时，

A：当原指针后面有充足内存时，realloc 就将后面的内存合并到原指针上，并返回原指针

B：当原指针后面没有足够内存时，系统会另外寻找内存，若分配成功，会将原内存数据拷贝至新内存，原内存会自动free掉；

若分配失败，realloc会返回NULL，原内存不会free 掉