5.3，常见的内存错误及对策

5.3.1，指针没有指向一块合法的内存

定义了指针变量，但是没有为指针分配内存，即指针没有指向一块合法的内存。

浅显的例子就不举了，这里举几个比较隐蔽的例子。

5.3.1.1，结构体成员指针未初始化

```
struct student
{
    char *name;
    int score;
}stu,*pstu;

int main()
{
     strcpy(stu.name,"Jimy");
     stu.score = 99;
     return 0;
 }
```

很多初学者犯了这个错误还不知道是怎么回事。这里定义了结构体变量 stu，但是他没想到这个结构体内部 char *name 这成员在定义结构体变量 stu 时，只是给 name 这个指针变量本身分配了 4 个字节。name 指针并没有指向一个合法的地址，这时候其内部存的只是一些乱码。所以在调用 strcpy 函数时，会将字符串"Jimy"往乱码所指的内存上拷贝，而这块内

存 name 指针根本就无权访问，导致出错。解决的办法是为 name 指针 malloc 一块空间。

同样，也有人犯如下错误：

```
int main()
{
    pstu = (struct student*)malloc(sizeof(struct student));
    strcpy(pstu->name,"Jimy");
    pstu->score = 99;
    free(pstu);

    return 0;
}
```

为指针变量 pstu 分配了内存，但是同样没有给 name 指针分配内存。错误与上面第一种情况一样，解决的办法也一样。这里用了一个 malloc 给人一种错觉，以为也给 name 指针分配了内存。

5.3.1.2，没有为结构体指针分配足够的内存

```
int main()
{
    pstu = (struct student*)malloc(sizeof(struct student*));
    strcpy(pstu->name,"Jimy");
    pstu->score = 99;
    free(pstu);
    return 0;
}
```

为 pstu 分配内存的时候，分配的内存大小不合适。这里把 sizeof(struct student)误写为 sizeof(struct student*)。当然 name 指针同样没有被分配内存。解决办法同上。

5.3.1.3，函数的入口校验

不管什么时候，我们使用指针之前一定要确保指针是有效的。

一般在函数入口处使用 assert(NULL != p)对参数进行校验。在非参数的地方使用 if（NULL != p）来校验。但这都有一个要求，即 p 在定义的同时被初始化为 NULL 了。比如上面的例子，即使用 if（NULL != p）校验也起不了作用，因为 name 指针并没有被初始化为 NULL，其内部是一个非 NULL 的乱码。

assert 是一个宏，而不是函数，包含在 assert.h 头文件中。如果其后面括号里的值为假，则程序终止运行，并提示出错；如果后面括号里的值为真，则继续运行后面的代码。这个宏只在 Debug 版本上起作用，而在 Release 版本被编译器完全优化掉，这样就不会影响代码的性能。

有人也许会问，既然在 Release 版本被编译器完全优化掉，那 Release 版本是不是就完全没有这个参数入口校验了呢？这样的话那不就跟不使用它效果一样吗？

是的，使用 assert 宏的地方在 Release 版本里面确实没有了这些校验。但是我们要知道，assert 宏只是帮助我们调试代码用的，它的一切作用就是让我们尽可能的在调试函数的时候把错误排除掉，而不是等到 Release 之后。它本身并没有除错功能。再有一点就是，参数出现错误并非本函数有问题，而是调用者传过来的实参有问题。assert 宏可以帮助我们定位错误，而不是排除错误。

5.3.2，为指针分配的内存太小

为指针分配了内存，但是内存大小不够，导致出现越界错误。

```
char *p1 = “abcdefg”;
char *p2 = (char *)malloc(sizeof(char)*strlen(p1));
strcpy(p2,p1);
```

p1 是字符串常量，其长度为 7 个字符，但其所占内存大小为 8 个 byte。初学者往往忘了字符串常量的结束标志“\0”。这样的话将导致 p1 字符串中最后一个空字符“\0”没有被拷贝到 p2 中。解决的办法是加上这个字符串结束标志符：

```
char *p2 = (char *)malloc(sizeof(char)*strlen(p1)+1*sizeof(char));
```

这里需要注意的是，只有字符串常量才有结束标志符。比如下面这种写法就没有结束标志符了：

```
char a[7] = {‘a’,’b’,’c’,’d’,’e’,’f’,’g’};
```

另外，不要因为 char 类型大小为 1 个 byte 就省略 sizof（char）这种写法。这样只会使你的代码可移植性下降。

5.3.3，内存分配成功，但并未初始化

犯这个错误往往是由于没有初始化的概念或者是以为内存分配好之后其值自然为 0。未初始化指针变量也许看起来不那么严重，但是它确确实实是个非常严重的问题，而且往往出现这种错误很难找到原因。

曾经有一个学生在写一个 windows 程序时，想调用字库的某个字体。而调用这个字库需要填充一个结构体。他很自然的定义了一个结构体变量，然后把他想要的字库代码赋值给了相关的变量。但是，问题就来了，不管怎么调试，他所需要的这种字体效果总是不出来。我在检查了他的代码之后，没有发现什么问题，于是单步调试。在观察这个结构体变

量的内存时，发现有几个成员的值为乱码。就是其中某一个乱码惹得祸！因为系统会按照这个结构体中的某些特定成员的值去字库中寻找匹配的字体，当这些值与字库中某种字体的某些项匹配时，就调用这种字体。但是很不幸，正是因为这几个乱码，导致没有找到相匹配的字体！因为系统并无法区分什么数据是乱码，什么数据是有效的数据。只要有数据，系统就理所当然的认为它是有效的。

也许这种严重的问题并不多见，但是也绝不能掉以轻心。所以在定义一个变量时，第一件事就是初始化。你可以把它初始化为一个有效的值，比如:

```
int   i = 10;
char *p = (char *)malloc(sizeof(char));
```

但是往往这个时候我们还不确定这个变量的初值，这样的话可以初始化为 0 或 NULL。

```
int   i = 0;
char *p = NULL;
```

如果定义的是数组的话，可以这样初始化：

```
int a[10] =   {0};
```

或者用 memset 函数来初始化为 0：

```
memset（a,0,sizeof(a)）;
```

memset 函数有三个参数，第一个是要被设置的内存起始地址；第二个参数是要被设置的值；第三个参数是要被设置的内存大小，单位为 byte。这里并不想过多的讨论 memset 函数的用法，如果想了解更多，请参考相关资料。

至于指针变量如果未被初始化，会导致 if 语句或 assert 宏校验失败。这一点，上面已有分析。

5.3.4，内存越界

内存分配成功，且已经初始化，但是操作越过了内存的边界。

这种错误经常是由于操作数组或指针时出现“多 1”或“少 1”。比如：

```
int a[10] = {0};
for (i=0; i<=10; i++)
{
    a[i] = i;
}
```

所以，for 循环的循环变量一定要使用半开半闭的区间，而且如果不是特殊情况，循环变量尽量从 0 开始。

5.3.5，内存泄漏

内存泄漏几乎是很难避免的，不管是老手还是新手，都存在这个问题。甚至包括windows，Linux这类软件，都或多或少有内存泄漏。也许对于一般的应用软件来说，这个问题似乎不是那么突出，重启一下也不会造成太大损失。但是如果你开发的是嵌入式系统软件呢？比如汽车制动系统，心脏起搏器等对安全要求非常高的系统。你总不能让心脏起搏器重启吧，人家阎王老爷是非常好客的。

会产生泄漏的内存就是堆上的内存（这里不讨论资源或句柄等泄漏情况），也就是说由malloc系列函数或new操作符分配的内存。如果用完之后没有及时free或delete，这块内存就无法释放，直到整个程序终止。