踏入C 中的雷区C 内存管理详解 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/464/2021_2022__E8_B8_8F_E5_85_A5C___E4_c67_464891.htm 伟大的Bill Gates 曾经失言：640K ought to be enough for everybody Bill Gates 1981 程序员们经常编写内存管理程序，往往提心吊胆。如果不想触雷，唯一的解决办法就是发现所有潜伏的地雷并且排除它们，躲是躲不了的。本文的内容比一般教科书的要深入得多，读者需细心阅读，做到真正地通晓内存管理。 1、内存分配方式 内存分配方式有三种： （1）从静态存储区域分配。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量，static变量。 （2）在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。 （3） 从堆上分配，亦称动态内存分配。程序在运行的时候用malloc或new申请任意多少的内存，程序员自己负责在何时用free或0delete释放内存。动态内存的生存期由我们决定，使用非常灵活，但问题也最多。 2、常见的内存错误及其对策 发生内存错误是件非常麻烦的事情。编译器不能自动发现这些错误，通常是在程序运行时才能捕捉到。而这些错误大多没有明显的症状，时隐时现，增加了改错的难度。有时用户怒气冲冲地把你找来，程序却没有发生任何问题，你一走，错误又发作了。 常见的内存错误及其对策如下：* 内存分配未成功，却使用了它。 编程新手常犯这种错误，因为他们没有意识到内存分配会不成功。常用解决办法是，在使

用内存之前检查指针是否为NULL。如果指针p是函数的参数，那么在函数的入口处用assert(p!=NULL)进行 检查。如果是用malloc或new来申请内存，应该用if(p==NULL)或if(p!=NULL)进行防错处理。 * 内存分配虽然成功，但是尚未初始化就引用它。 犯这种错误主要有两个起因：一是没有初始化的观念；二是误以为内存的缺省初值全为零，导致引用初值错误（例如数组）。 内存的缺省初值究竟是什么并没有统一的标准，尽管有些时候为零值，我们宁可信其无不可信其有。所以无论用何种方式创建数组，都别忘了赋初值，即便是赋零值也不可省略，不要嫌麻烦。 * 内存分配成功并且已经初始化，但操作越过了内存的边界。 例如在使用数组时经常发生下标“ 多1 ”或者“ 少1 ”的操作。特别是在for循环语句中，循环次数很容易搞错，导致数组操作越界。 * 忘记了释放内存，造成内存泄露。 含有这种错误的函数每被调用一次就丢失一块内存。刚开始时系统的内存充足，你看不到错误。终有一次程序突然死掉，系统出现提示：内存耗尽。 动态内存的申请与释放必须配对，程序中malloc与free的使用次数一定要相同，否则肯定有错误（new/0delete同理）。 * 释放了内存却继续使用它。 有三种情况： （1）程序中的对象调用关系过于复杂，实在难以搞清楚某个对象究竟是否已经释放了内存，此时应该重新设计数据结构，从根本上解决对象管理的混乱局面。 （2）函数的return语句写错了，注意不要返回指向“ 栈内存 ”的“ 指针 ”或者“ 引用 ”，因为该内存在函数体结束时被自动销毁。 （3）使用free或0delete释放了内存后，没有将指针设置为NULL。导致产生“ 野指针 ”。 【规则1】用malloc或new申请内存之后，应该立即检查指

针值是否为NULL。防止使用指针值为NULL的内存。 【规则2】不要忘记为数组和动态内存赋初值。防止将未被初始化的内存作为右值使用。 【规则3】避免数组或指针的下标越界，特别要当心发生“ 多1 ”或者“ 少1 ”操作。 【规则4】动态内存的申请与释放必须配对，防止内存泄漏。 【规则5】用free或0delete释放了内存之后，立即将指针设置为NULL，防止产生“ 野指针 ”。 3、指针与数组的对比 C /C程序中，指针和数组在不少地方可以相互替换着用，让人产生一种错觉，以为两者是等价的。 数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。数组名对应着（而不是指向）一块内存，其地址与容量在生命期内保持不变，只有数组的内容可以改变。 指针可以随时指向任意类型的内存块，它的特征是“ 可变 ”，所以我们常用指针来操作动态内存。指针远比数组灵活，但也更危险。 下面以字符串为例比较指针与数组的特性。 3.1 修改内容 示例3-1中，字符数组a的容量是6个字符，其内容为hello。a的内容可以改变，如a[0]=‘ X ’。指针p指向常量字符串“ world ”（位于静态存储区，内容为world），常量字符串的内容是不可以被修改的。从语法上看，编译器并不觉得语句p[0]=‘ X ’有什么不妥，但是该语句企图修改常量字符串的内容而导致运行错误。 char a[] = “ hello ”. a[0] = ‘ X ’. cout char *p = “ world ”. // 注意p指向常量字符串 p[0] = ‘ X ’. // 编译器不能发现该错误 cout