C语言最大难点揭秘[3] PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/246/2021_2022_C_E8_AF_AD_E8_A8_80_E6_9C_80_c97_246240.htm 悬空指针 悬空指针比较棘手。当程序员在内存资源释放后使用资源时会发生悬空指针（请参见清单 5）： 清单 5. 悬空指针 void f8() { struct x *xp. xp = (struct x *) malloc(sizeof (struct x)). xp.q = 13. ... free(xp). ... /* Problem! Theres no guarantee that the memory block to which xp points hasnt been overwritten. */ return xp.q. } 传统的“ 调试 ”难以隔离悬空指针。由于下面两个明显原因，它们很难再现： 即使影响提前释放内存范围的代码已本地化，内存的使用仍然可能取决于应用程序甚至（在极端情况下）不同进程中的其他执行位置。 悬空指针可能发生在以微妙方式使用内存的代码中。结果是，即使内存在释放后立即被覆盖，并且新指向的值不同于预期值，也很难识别出新值是错误值。 悬空指针不断威胁着 C 或 C 程序的运行状态。 数组边界违规 数组边界违规十分危险，它是内存错误管理的最后一个主要类别。回头看一下清单 1；如果 explanation 的长度超过 80，则会发生什么情况？回答：难以预料，但是它可能与良好情形相差甚远。特别是，C 复制一个字符串，该字符串不适于为它分配的 100 个字符。在任何常规实现中，“ 超过的 ”字符会覆盖内存中的其他数据。内存中数据分配的布局非常复杂并且难以再现，所以任何症状都不可能追溯到源代码级别的具体错误。这些错误通常会导致数百万美元的损失。 内存编程的策略 勤奋和自律可以让这些错误造成的影响降至最低限度。下面我们介绍一下您可以采用的几个特定步骤；我在各种

组织中处理它们的经验是，至少可以按一定的数量级持续减少内存错误。 编码风格 编码风格是最重要的，我还从没有看到过其他任何作者对此加以强调。影响资源（特别是内存）的函数和方法需要显式地解释本身。下面是有关标头、注释或名称的一些示例（请参见清单 6）。 清单 6. 识别资源的源代码示例 /******** * ... * * Note that any function invoking protected_file_read() * assumes responsibility eventually to fclose() its * return value, UNLESS that value is NULL. * ********/ FILE *protected_file_read(char *filename) { FILE *fp. fp = fopen(filename, "r"). if (fp) { ... } else { ... } return fp. } /******* * ... * * Note that the return value of get_message points to a * fixed memory location. Do NOT free() it. remember to * make a copy if it must be retained ... * ********/ char *get_message() { static char this_buffer[400]. ... (void) sprintf(this_buffer, ...). return this_buffer. } /******** * ... * While this function uses heap memory, and so * temporarily might expand the over-all memory * footprint, it properly cleans up after itself. * ********/ int f6(char *item1) { my_class c1. int result. ... c1 = new my_class(item1). ... result = c1.x. 0delete c1. return result. } /******** * ... * Note that f8() is documented to return a value 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com