

C语言中的指针和内存泄漏 PDF转换可能丢失图片或格式，
建议阅读原文

https://www.100test.com/kao_ti2020/224/2021_2022_C_E8_AF_AD_E8_A8_80_E4_B8_AD_c103_224248.htm 在使用 C 语言时，您是否对花时间调试指针和内存泄漏问题感到厌倦？如果是这样，那么本文就适合您。您将了解可能导致内存破坏的指针操作类型，您还将研究一些场景，了解要在使用动态内存分配时考虑什么问题。引言 对于任何使用 C 语言的人，如果问他们 C 语言的最大烦恼是什么，其中许多人可能会回答说是指针和内存泄漏。这些的确是消耗了开发人员大多数调试时间的事项。指针和内存泄漏对某些开发人员来说似乎令人畏惧，但是一旦您了解了指针及其关联内存操作的基础，它们就是您在 C 语言中拥有的最强大工具。本文将与您分享开发人员在开始使用指针来编程前应该知道的秘密。本文内容包括：导致内存破坏的指针操作类型 在使用动态内存分配时必须考虑的检查点 导致内存泄漏的场景 如果您预先知道什么地方可能出错，那么您就能够小心避免陷阱，并消除大多数与指针和内存相关的问题。什么地方可能出错？有几种问题场景可能会出现，从而可能在完成生成后导致问题。在处理指针时，您可以使用本文中的信息来避免许多问题。未初始化的内存 在本例中，p 已被分配了 10 个字节。这 10 个字节可能包含垃圾数据，如图 1 所示。char *p = malloc (10).图 1. 垃圾数据 如果在对这个 p 赋值前，某个代码段尝试访问它，则可能会获得垃圾值，您的程序可能具有不可预测的行为。p 可能具有您的程序从未曾预料到的值。良好的实践是始终结合使用 memset 和 malloc，或者使用 calloc。char *p = malloc

(10). `memset(p, ' \0 ', 10)`. 现在，即使同一个代码段尝试在对 `p` 赋值前访问它，该代码段也能正确处理 Null 值（在理想情况下应具有的值），然后将具有正确的行为。内存覆盖 由于 `p` 已被分配了 10 个字节，如果某个代码片段尝试向 `p` 写入一个 11 字节的值，则该操作将在不告诉您的情况下自动从其他某个位置“吃掉”一个字节。让我们假设指针 `q` 表示该内存。图 2. 原始 `q` 内容 图 3. 覆盖后的 `q` 内容 结果，指针 `q` 将具有从未预料到的内容。即使您的模块编码得足够好，也可能由于某个共存模块执行某些内存操作而具有不正确的行为。下面的示例代码片段也可以说明这种场景。

```
char *name = (char *) malloc(11). // Assign some value to name
memcpy ( p,name,11). // Problem begins here
```

在本例中，`memcpy` 操作尝试将 11 个字节写到 `p`，而后者仅被分配了 10 个字节。作为良好的实践，每当向指针写入值时，都要确保对可用字节数和所写入的字节数进行交叉核对。一般情况下，`memcpy` 函数将是用于此目的的检查点。内存读取越界 内存读取越界 (overread) 是指所读取的字节数多于它们应有的字节数。这个问题并不太严重，在此就不再详述了。下面的代码提供了一个示例。

```
char *ptr = (char *)malloc(10).char name[20] . memcpy ( name,ptr,20). // Problem begins here
```

在本例中，`memcpy` 操作尝试从 `ptr` 读取 20 个字节，但是后者仅被分配了 10 个字节。这还会导致不希望的输出。[1] [2] 下一页 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