C语言编程常见问题解答之调试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/285/2021_2022_C_E8_AF_AD_E8_A8_80_E7_BC_96_c97_285296.htm 调试(debugging)是指去掉程序中的错误(通常被称为bugs)的过程。一个错误可能非常简单，例如拼错一个单词或者漏掉一个分号；也可能比较复杂，例如使用一个指向并不存在的地址的指针。无论错误的复杂程度如何，掌握正确的调试方法都能使程序员受益匪浅。 11．1 如果我运行的程序挂起了，应该怎么办? 当你运行一个程序时会有多种原因使它挂起，这些原因可以分为以下4种基本类型： (1)程序中有死循环； (2)程序运行的时间比所期望的长； (3)程序在等待某些输入信息，并且直到输入正确后才会继续运行； (4)程序设计的目的就是为了延迟一段时间，或者暂停执行。 在讨论了因未知原因而挂起的程序的调试技巧后，将逐个分析上述的每种情况。 调试那些因未知原因而挂起的程序是非常困难的。你可能花费了很长的时间编写一个程序，并努力确保每条代码都准确无误，你也可能只是在一个原来运行良好的程序上作了一个很小的修改，然而，当你运行程序时屏幕上却什么也没有显示。如果你能得到一个错误的结果，或者部分结果，你也许知道应该作些什么修改，而一个空白的屏幕实在令人沮丧，你根本不知道错在哪里。 在开始调试这样一个程序时，你应该先检查一下程序结构，然后再按执行顺序依次查看程序的各个部分，看看它们是否能正确运行。 例如，如果主程序只包含3个函数调用A()、B()和C()，那么在调试时，你可以先检查函数A()是否把控制权返回给了主程序。为此，你可以在调用函数A()的语句后

面加上exit()命令，也可以用注释符把对函数B()和C()的调用括起来，然后重新编译并运行这个程序。 注意：通过调试程序(debugger)也可以做到这一点，然而上述方法是一种很传统的调试方法。调试程序是一个程序，它的作用是让程序员能够观察程序的运行情况、程序的当前运行行号、变量的值，等等。 此时你将看到函数A()是否将控制权返回给了主程序如果该程序运行并退出，你可以判断是程序的其它部分使程序挂起。你可以用这种方法测试程序的每一部分，直到发现使程序挂起的那一部分，然后集中精力修改相应的函数。 有时，情况会更复杂一些。例如，使程序挂起的函数本身是完全正常的，问题可能出在该函数从别的地方得到了一些错误的数据。这时，你就要检查该函数所接受的所有的值，并找出是哪些值导致了错误操作。 技巧：监视函数是调试程序的出色功能之一。 分析下面这个简单的例子将帮助你掌握这种技巧的使用方法： #include #include /* * Declare the functions that the main function is using */ int A(), B(int), C(int, int). /* * The main program */ int A(), B(), C(). /*These are functions in some other module * / int main() { int v1, v2, v3. v1 = A(). v2 = B(v1). v3 = C(v1, v2). printf ("The Result is %d. \n" , v3). return(0) . } 你可以在调用函数A()的语句后输出变量v1的值，以确认它是否在函数B()所能接受的值的范围之内，因为即使是函数B()使程序挂起，它本身并不一定就有错，而可能是因为函数A()给了函数B()一个并非它所期望的值。 现在，已经分析了调试“ 挂起 ”的程序的基本方法，下面来看看一些使程序挂起的常见错误。 死循环 当你的程序出现了死循环时，机器将无数次地执行同一段代码，这种操作当然是程序员所不希望的。出现死

循环的原因是程序员使程序进行循环的判断条件永远为真，或者使程序退出循环的判断条件永远为假。下面是一个死循环的例子： /* initialize a double dimension array */ for (a = 0 . a { for(b = 0. b { array[a][b]==0. } } 这里的问题是程序员犯了一个错误(事实上可能是键入字母的错误)，第二个循环本应在变量b增加到10后结束，但是却从未让变量b的值增加!第二个for循环的第三部分增加变量a的值，而程序员的本意是要增加变量b的值。因为b的值将总是小于10，所以第二个for循环会一直运行下去。 怎样才能发现这个错误呢?除非你重新阅读该程序并注意到变量b的值没有增加，否则你不可能发现这个错误。当你试图调试该程序时，你可以在第二个for循环的循环体中加入这样一条语句： printf(" %d %d %d\n" , a , b , array[a][b]) . 这条语句的正确输出应该是： 0 0 0 0 1 0 (and eventually reaching) 9 9 0 但你实际上看到的输出却是： 0 0 0 1 0 0 2 0 0 ... 你所得到是一个数字序列，它的第一项不断增加，但它本身永远不会结束。用这种方法输出变量不仅可以找出错误，而且还能知道数组是否由所期望的值组成。这个错误用其它方法似乎很难发现!这种输出变量内容的技巧以后还会用到。