C语言辅导:可移植性 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_E8_BE_85_c97_285289.htm 可移植性并不是指所写的程序不作修改就可以在任何计算机上运行，而是指当条件有变化时，程序无需作很多修改就可运行。 你不要把“ 我不会遇到这种情况 ”这句话说得太早。直到MSWindows出现之前，许多MSDOS程序员还不怎么关心可移植性问题。然后，突然之间，他们的程序不得不在一个看起来不同的操作系统上运行。当Power PC流行起来后，Mac机的程序员不得不去应付一个新的处理器。任何一个在同版本的UNIX下维护过程序的人所了解的可移植性的知识，恐怕都足以写成一本书，更别说写成一章了。 假设你用基本ALBATROS(Anti-lock Braking and Tire Rotation operating system)的Tucker C来编写防抱死刹车软件，这听起来好象是一个最典型的不可移植软件。即便如此，可移植性仍然很重要：你可能需要把它
从Tucker C的7．55c版本升级到8．O版本，或者
从ALBATROS的3．o版本升级到3．2a版本，以修改软件中的某些错误；你也可能会出于仿真测试或宣传的目的，而把它(或其中一部分)移植到MS-Windows或UNIX工作站上；更为可能的是，在它尚未最终完工之前，你会把它从一个程序员手中交到另一个程序员手中。 可移植性的本意是按照意料之中的方式做事情，其目的不在于简化编译程序的工作，而在于使改写(重写!)程序的工作变得容易。如果你就是接过别人的程序的“ 倒霉蛋 ”，那么原程序中的每一处出乎意料之外的地方都会花去你的时间，并且将来可能会引起微妙的错误。

如果你是原程序的编写者，你应该注意不要使你的程序中出现出乎接手者意料之外的代码。你应该尽量使程序容易理解，这样就不会有人抱怨你的程序难懂了。此外，几个月以后，下一个“倒霉蛋”很可能就会是你自己了，而这时你可能已经忘记了当初为什么用这样复杂的一种方式来写一个for循环。使程序可移植的本质非常简单：如果做某些事情有一种既简单又标准的方法，就按这种方法做。使程序可移植的第一步就是使用标准库函数，并且把它们和ANSI／ISO C标准中定义的头文件放在一起使用，详见第11章“标准库函数”。第二步是尽可能使所写的程序适用于所有的编译程序，而不是仅仅适用于你现在所使用的编译程序。如果你的手册提醒你某种功能或某个函数是你的编译程序或某些编译程序所特有的。你就应该谨慎地使用它。有许多关于c语言编程的好书中都提出了一些关于如何保持良好的可移植性的建议。特别地，当你不清楚某个东西是否会起作用时，不要马上写一个测试程序来看看你的编译程序是否会接受它，因为即使这个版本的编译程序接受它，也不能说明这个程序就有很好的可移植性(C 程序员比c程序员应该更重视这个问题)。此外，小的测试程序很可能会漏掉要测试的性能或问题的某些方面。第三步是把不可移植的代码分离出来。如果你无法确定某段程序是否可移植，你就应该尽快注释出这一点。如果有一些大的程序段(整个函数或更多)依赖于它们的运行环境或编译方式，你就应该把其中不可移植的代码分离到一些独立的“．c”文件中。如果只在一些小的程序段中存在可移植性问题，你可以使用#ifdef预处理指令。例如，在MS-DOS中文件名的形式为“＼tools＼readme”，而在UNIX中文件名的形式

为“ ／tools／readme ”。如果你的程序需要把这样的 文件名分解为独立的部分，你就需要查找正确的分隔符。如果有这样一段代码 #ifdef unix #define FILE_SEP_CHAR ’ / ’ #endif #ifdef __MSDOS__ define FILE SEP CHAR ’ \\ ’ #endif 你就可以通过把FILE_SEP_CHAR传递给strchr()或strtok()来找出文件名中的路径部分。尽管这一步还无法找出一个MS-DOS文件的驱动器名，但它已经是一个正确的开头了。 最后，找出潜在的可移植性问题的最好方法之一就是请别人来查找!如果可以的话，最好请别人来检查一下你的程序。他或许知道一些你不知道的东西，或许能发现一些你从未想过的问题(有些名称中含"lint"的工具和有些编译程序选项可以帮助你找出一些问题，但你不要指望它们能找出大的问题)。 15．1 编译程序中的C 扩充功能可以用在C程序中吗? 不可以，它们只能用在真正的C 程序中。 C 中的一些突出性能已被ANSI／ISO C标准委员会所接受，它们不再是“ C 扩充功能 ”，而已经成为C的一部分。例如，函数原型和const关键字就被补充到C中，因为它们确实非常有用。 有一些C 性能，例如内联(inline)函数和用const代替#define的方法，有时被称为“ 高级C ”性能。有些C和C 共用的编译程序提供了一些这样的性能，你可以使用它们吗? 有些程序员持这样一种看法：如果要写C代码，就只写C代码，并且使它能被所有的C编译程序接受。如果想使用C 性能，那么就转到C 上。你可以循序渐进，每次用一点新的技巧；也可以一步到位，用大量的内联函数，异常处理和转换运算符编写模块化的抽象基类。当你跨过这一步之后，你的程序就是现在的C 程序了，并且你不要指望C编译程序还会接受它。 笔者的看法是：你的工作是从一个新的C标准

开始的，这个标准中包含一些C 性能和一些崭新的性能。在以后的几年中，一些编译程序的开发商会去实现这些新的性能的一部分，但这并不能保证所有的编译程序都会去实现这些性能，也不能保证下一个C标准会纳入这些性能。你应该保持对事态发展的关注，当一项新的性能看上去已经真正流行起来，并且不仅仅出现在你现在所使用的编译程序中，而是出现在所有你可能用到的编译程序中时，你就可以考虑使用它了。例如，如果过去有人非要等到1989年才开始使用函数原型，那么这其实就不是一种明智之举；另一方面，在保证可移植性的前提下，过去也没有一个开始使用noalias关键字的最佳时机。 请参见： 15．2 C 和C有什么区别? 15．2 C 和C有什么区别? 这个问题要从C程序员和C 程序员两个角度去分析。 对C程序员来说，C 是一种古怪的难以掌握的语言。大多数C 库无法通过C编译程序连接到c程序中(在连接时编译程序必须创建模型或“ 虚拟表 ”，而C编译程序不提供这种支持)。即使用c 编译程序来连接程序，c程序仍然无法调用许多C 函数。除非非常小心地编写c 程序，否则C 程序总会比类似的c程序慢一些，并且大一些。C 编译程序中的错误也比C编译程序中的多。C 程序更难于从一种编译程序移植到另一种编译程序上。最后一点，C 是一种庞大的难以学会的语言，它的定义手册(1990)超过400页，而且每年还要加入大量的内容。另一方面，c语言是一种既漂亮又简炼的语言，并且这几年来没有什么改动(当然不可能永远不会有改动，见14．1)。C编译程序工作良好，并且越来越好。好的c程序可以很方便地在好的C编译程序之间移植。虽然在C中做面向对象的设计并不容易，但也不是非常困难。如果需要的话，你(几

乎)总是可以用c 编译程序来生成C程序。 对于C 程序员来说，c是一个好的开端。在C 中你不会重犯在C中犯过的许多错误，因为编译程序不会给你这个机会。C的有些技巧，如果使用稍有不当，就会带来很大的危险。 另一方面，c 是一种优秀的语言。只需应用少数原则，稍作一点预先的设计工作，就能写出安全、高效并且非常容易理解和维护的C 程序。用有些方法写C 程序，能使C 程序比类似的C程序更快并且更小。面向对象的设计在C 中非常容易，但你不一定要按这种方式工作。编译程序日臻完善，标准也逐渐确立起来。如果需要的话，你随时可以返回到C中。 那么，c和C 之间有什么具体的区别呢?C的有些成分在c 中是不允许使用的，例如老式的函数定义。大致来说，C 只是一种增加了一些新性能的C：新的注释规则(见15．3)；带有真正的true和false值的布尔类型，与现有的c或c 程序兼容(你可以把贴在显示器上的写着“ O=false，1=true ”的纸条扔掉了。它仍然有效，但已不是必须的了)。 内联函数比#define宏定义更加安全，功能也更强，而速度是一样的。 如果需要的话，可以确保变量的初始化，不再有用的变量会被自动清除。 类型检查和内存管理的功能更好，更安全，更强大。 封装(encapsulation)使新的类型可以和它们的所有操作一起被定义。c 中有一种complex类型，其操作和语法规则与float和double相同，但它不是编译程序所固有的，而是在C 中实现的，并且所使用的是每一个C 程序员都能使用的那些性能。 访问权控制(access contr01)使得只能通过一个新类型所允许的操作来使用该类型。 继承和模板(inheritance and templates)两种编写程序的辅助方法，提供了函数调用之外的代码复用方式。 异常处理(exceptions)使一

个函数可以向它的调用者之外的函数报告问题。 一种新的I／O处理方法比printf()更安全并且功能更强，能把格式和要写入的文件的类型分离开。 一个数据类型丰富的库你永远不需要自己编写链表或二叉树了(这一点是千真万确的!)。