编写可移植C_C 程序的要点 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/260/2021_2022__E7_BC_96_E5_86_99_E5_8F_AF_E7_c103_260911.htm 1.分层设计，隔离平台相关的代码。就像可测试性一样，可移植性也要从设计抓起。一般来说，最上层和最下层都不具有良好的可移植性。最上层是GUI，大多数GUI都不是跨平台的，如Win32 SDK和MFC。最下层是操作系统API，大多部分操作系统API都是专用的。 如果这两层的代码散布在整个软件中，那么这个软件的可植性将非常的差，这是不言自明的。那么如何避免这种情况呢？当然是分层设计了： 最底层采用Adapter模式，把不同操作系统的API封装成一套统一的接口。至于封装成类还是封装成函数，要看你采用的C还是C 写的程序了。这看起来很简单，其实不尽然（看完整篇文章后你会明白的），它将耗去你大量的时间去编写代码，去测试它们。采用现存的程序库，是明智的做法，有很多这样的库，比如，C库有glib（GNOME的基础类），C 库有ACE(ADAPTIVE Communication Environment)等等，在开发第一个平台时就采用这些库，可以大大减少移植的工作量。 最上层采用MVC模型，分离界面表现与内部逻辑代码。把大部分代码放到内部逻辑里面，界面仅仅是显示和接收输入，即使要换一套GUI，工作量也不大。这同时也是提高可测试性的手段之一，当然还有其它一些附加好处。所以即使你采用QT或者GTK 等跨平台的GUI设计软件界面，分离界面表现与内部逻辑也是非常有用的。 若做到了以上两点，程序的可移植性基本上有保障了，其它的只是技术细节问题。 2.事先熟悉各目标平台，

合理抽象底层功能。这一点是建立在分层设计之上的，大多数底层函数，像线程、同步机制和IPC机制等等，不同平台提供的函数，几乎是一一对应的，封装这些函数很简单，实现Adapter的工作几乎只是体力活。然而，对于一些比较特殊的应用，如图形组件本身，就拿GTK 来说吧，基于X Window的功能和基于Win32的功能，两者差巨大，除了窗口、事件等基本概念外，几乎没有什么相同的，如果不事先了解各个平台的特性，在设计时就精心考虑的话，抽象出来的抽口在另外一个平台几乎无法实现。 3.尽量使用标准C/C 函数。大多数平台都会实现POSIX(Portable Operating System Interface)规定的函数，但这些函数较原生(Native) 函数来说，性能上的表现可能较次一些，用起来也不如原生函数方便。但是，最好不要贪图这种便宜而使用原生函数函数，否则搬起的石头最终会轧到自己的脚。比如，文件操作就用fopen之类的函数，而不要用CreateFile之类的函数等。 4.尽量不要使用C/C 新标准里出现的特性。并不是所有的编译器都支持这些特性，像VC就不支持C99里面要求的可变参数的宏，VC对一些模板特性的支持也不全面。为了安全起见，这方面不要太激进了。 5.尽量不要使用C/C 标 准里没有明确规定的特性。比如你有多个动态库，每个动态库都有全局对象，而且这些全局对象的构造还有依赖关系，那你迟早会遇到麻烦的，这些全局对象构造的 先后顺序在标准里是没有规定的。在一个平台上运行正确，在另外一个平台上可能莫明其妙的死机，最终还是要对程序作大量修改。 6.尽量不要使用准标准函数。有些函数大多数平台上都有，它们使用得太广泛了，以至于大家都把它们当成标准了，比如atoi（把字符串转换成整数）

、strdup(克隆字符串)、alloca（在栈分配自动内存）等等。不怕一万，就怕万一，除非明白你在做什么，否则还是别碰它们为好。 7.注意标准函数的细节。也许你不相信，即使是标准函数，抛开内部实现不论，就其外在表现的差异也有时令人惊讶。这里略举几个例子： int accept(int s, struct sockaddr *addr, socklen_t *addrlen).addr/ addrlen本来是输出参数，如果是C 程序员，不管怎么样，你已经习惯于初始化所有的变量，不会有问题。如果是C程序员，就难说了，若没有初始化它们，程序可能莫名其妙的crash，而你做梦也怀疑不到它头它。这在Win32下没问题，在Linux下才会出现。 int snprintf(char *str, size_t size, const char *format, ……).第二个参数size，在Win32下不包括空字符在内，在Linux下包括空字符，这一个字符的差异，也可能让你耗上几个小时。 int stat(const char *file_name, struct stat *buf).这个函数本身没有问题，问题出在结构stat上，st_ctime在Win32下代表创建(create)时间，在Linux下代表最后修改(change)时间。 FILE *fopen(const char *path, const char *mode).在读取二进制文件，没有什么问题。在读取文本文件可要小心，Win32下自动预处理，读出来的内容与文件实际都长度不一样，在Linux则没有问题。 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