高质量C _C编程指南--第6章函数设计 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/134/2021_2022__E9_AB_98_E8_B4_A8_E9_87_8FC_c97_134223.htm 第6章 函数设计函数是C /C程序的基本功能单元，其重要性不言而喻。函数设计的细微缺点很容易导致该函数被错用，所以光使函数的功能正确是不够的。本章重点论述函数的接口设计和内部实现的一些规则。 函数接口的两个要素是参数和返回值。C语言中，函数的参数和返回值的传递方式有两种：值传递（pass by value）和指针传递（pass by pointer）。C 语言中多了引用传递（pass by reference）。由于引用传递的性质象指针传递，而使用方式却象值传递，初学者常常迷惑不解，容易引起混乱，请先阅读6.6节“ 引用与指针的比较 ”。6.1 参数的规则l 【规则6-1-1】参数的书写要完整，不要贪图省事只写参数的类型而省略参数名字。如果函数没有参数，则用void填充。例如：void SetValue(int width, int height). // 良好的风格void SetValue(int, int). // 不良的风格float GetValue(void). // 良好的风格float GetValue(). // 不良的风格l 【规则6-1-2】参数命名要恰当，顺序要合理。例如编写字符串拷贝函数StringCopy，它有两个参数。如果把参数名字起为str1和str2，例如void StringCopy(char *str1, char *str2).那么我们很难搞清楚究竟是把str1拷贝到str2中，还是刚好倒过来。可以把参数名字起得更有意义，如叫strSource和strDestination。这样从名字上就可以看出应该把strSource拷贝到strDestination。还有一个问题，这两个参数那一个该在前那一个该在后？参数的顺序要遵循程序员的习惯。一般地，应将目的参数放在前面，源参数放

在后面。如果将函数声明为：void StringCopy(char *strSource, char *strDestination).别人在使用时可能会不假思索地写成如下形式：char str[20].StringCopy(str, “ Hello World ”). // 参数顺序颠倒l 【规则6-1-3】如果参数是指针，且仅作输入用，则应在类型前加const，以防止该指针在函数体内被意外修改。例如：void StringCopy(char *strDestination，const char *strSource).l 【规则6-1-4】如果输入参数以值传递的方式传递对象，则宜改用 “ const &amp. ” 方式来传递，这样可以省去临时对象的构造和析构过程，从而提高效率。2 【建议6-1-1】避免函数有太多的参数，参数个数尽量控制在5个以内。如果参数太多，在使用时容易将参数类型或顺序搞错。2 【建议6-1-2】尽量不要使用类型和数目不确定的参数。C标准库函数printf是采用不确定参数的典型代表，其原型为：int printf(const chat *format[, argument]…).这种风格的函数在编译时丧失了严格的类型安全检查。6.2 返回值的规则l 【规则6-2-1】不要省略返回值的类型。C语言中，凡不加类型说明的函数，一律自动按整型处理。这样做不会有什么好处，却容易被误解为void类型。C 语言有很严格的类型安全检查，不允许上述情况发生。由于C 程序可以调用C函数，为了避免混乱，规定任何C / C函数都必须有类型。如果函数没有返回值，那么应声明为void类型。l 【规则6-2-2】函数名字与返回值类型在语义上不可冲突。违反这条规则的典型代表是C标准库函数getchar。例如：char c.c = getchar().if (c == EOF)…按照getchar名字的意思，将变量c声明为char类型是很自然的事情。但不幸的是getchar的确不是char类型，而是int类型，其原型如下：int getchar(void).由于c是char类型，取值范围

是[-128，127]，如果宏EOF的值在char的取值范围之外，那么if语句将总是失败，这种“危险”人们一般哪里料得到！导致本例错误的责任并不在用户，是函数getchar误导了使用者。l【规则6-2-3】不要将正常值和错误标志混在一起返回。正常值用输出参数获得，而错误标志用return语句返回。回顾上例，C标准库函数的设计者为什么要将getchar声明为令人迷糊的int类型呢？他会那么傻吗？在正常情况下，getchar的确返回单个字符。但如果getchar碰到文件结束标志或发生读错误，它必须返回一个标志EOF。为了区别于正常的字符，只好将EOF定义为负数（通常为负1）。因此函数getchar就成了int类型。我们在实际工作中，经常会碰到上述令人为难的问题。为了避免出现误解，我们应该将正常值和错误标志分开。即：正常值用输出参数获得，而错误标志用return语句返回。函数getchar可以改写成 BOOL GetChar(char *c).虽然gechar比GetChar灵活，例如 putchar(getchar()). 但是如果getchar用错了，它的灵活性又有什么用呢？2 【建议6-2-1】有时候函数原本不需要返回值，但为了增加灵活性如支持链式表达，可以附加返回值。例如字符串拷贝函数strcpy的原型：char *strcpy(char *strDest，const char *strSrc).strcpy函数将strSrc拷贝至输出参数strDest中，同时函数的返回值又是strDest。这样做并非多此一举，可以获得如下灵活性：char str[20].int length = strlen(strcpy(str,“Hello World”)). 2 【建议6-2-2】如果函数的返回值是一个对象，有些场合用“引用传递”替换“值传递”可以提高效率。而有些场合只能用“值传递”而不能用“引用传递”，否则会出错。