

高质量C\_C编程指南--第8章C函数的高级特性 PDF转换可能丢失图片或格式，建议阅读原文

[https://www.100test.com/kao\\_ti2020/134/2021\\_2022\\_\\_E9\\_AB\\_98\\_E8\\_B4\\_A8\\_E9\\_87\\_8FC\\_c97\\_134208.htm](https://www.100test.com/kao_ti2020/134/2021_2022__E9_AB_98_E8_B4_A8_E9_87_8FC_c97_134208.htm)

第8章 C函数的高级特性 对比于C语言的函数，C增加了重载（overloaded）、内联（inline）、const和virtual四种新机制。其中重载和内联机制既可用于全局函数也可用于类的成员函数，const与virtual机制仅用于类的成员函数。重载和内联肯定有其好处才会被C语言采纳，但是不可以当成免费的午餐而滥用。本章将探究重载和内联的优点与局限性，说明什么情况下应该采用、不该采用以及要警惕错用。

### 8.1 函数重载的概念

#### 8.1.1 重载的起源

自然语言中，一个词可以有許多不同的含义，即该词被重载了。人们可以通过上下文来判断该词到底是哪种含义。“词的重载”可以使语言更加简练。例如“吃饭”的含义十分广泛，人们没有必要每次非得说清楚具体吃什么不可。别迂腐得象孔已己，说茴香豆的茴字有四种写法。在C程序中，可以将语义、功能相似的几个函数用同一个名字表示，即函数重载。这样便于记忆，提高了函数的易用性，这是C语言采用重载机制的一个理由。例如示例8-1-1中的函数EatBeef,EatFish,EatChicken可以用同一个函数名Eat表示，用不同类型的参数加以区别。

```
void EatBeef(...). // 可以改为 void Eat(Beef ...). void EatFish(...). // 可以改为 void Eat(Fish ...). void EatChicken(...). // 可以改为 void Eat(Chicken ...).
```

示例8-1-1 重载函数Eat C语言采用重载机制的另一个理由是：类的构造函数需要重载机制。因为C规定构造函数与类同名（请参见第9章），构造函数只能有一个名字。如果想用几种不同的方法

创建对象该怎么办？别无选择，只能用重载机制来实现。所以类可以有多个同名的构造函数。8.1.2 重载是如何实现的？几个同名的重载函数仍然是不同的函数，它们是如何区分的呢？我们自然想到函数接口的两个要素：参数与返回值。如果同名函数的参数不同（包括类型、顺序不同），那么容易区别出它们是不同的函数。如果同名函数仅仅是返回值类型不同，有时可以区分，有时却不能。例如：`void Function(void)`. `int Function (void)`. 上述两个函数，第一个没有返回值，第二个的返回值是int类型。如果这样调用函数：`int x = Function ()`. 则可以判断出Function是第二个函数。问题是在C /C++程序中，我们可以忽略函数的返回值。在这种情况下，编译器和程序员都不知道哪个Function函数被调用。所以只能靠参数而不能靠返回值类型的不同来区分重载函数。编译器根据参数为每个重载函数产生不同的内部标识符。例如编译器为示例8-1-1中的三个Eat函数产生象`_eat_beef`、`_eat_fish`、`_eat_chicken`之类的内部标识符（不同的编译器可能产生不同风格的内部标识符）。如果C++程序要调用已经被编译后的C函数，该怎么办？假设某个C函数的声明如下：`void foo(int x, int y)`. 该函数被C编译器编译后在库中的名字为`_foo`，而C++编译器则会产生像`_foo_int_int`之类的名字用来支持函数重载和类型安全连接。由于编译后的名字不同，C++程序不能直接调用C函数。C++提供了一个C连接交换指定符号`extern "C"`来解决这个问题。例如：`extern "C" { void foo(int x, int y). ... // 其它函数 }`或者写成`extern "C" { #include "myheader.h" ... // 其它C头文件 }`这就告诉C++编译器，函数foo是个C连接，应该到库中找名字`_foo`而不是

找foo\_int\_int。C编译器开发商已经对C标准库的头文件作了extern“C”处理，所以我们可以用#include直接引用这些头文件。注意并不是两个函数的名字相同就能构成重载。全局函数和类的成员函数同名不算重载，因为函数的作用域不同。例如：`void Print(...)` // 全局函数 `class A { ... void Print(...)` // 成员函数 } 不论两个Print函数的参数是否不同，如果类的某个成员函数要调用全局函数Print，为了与成员函数Print区别，全局函数被调用时应加‘::’标志。如：`::Print(...)` // 表示Print是全局函数而非成员函数

### 8.1.3 当心隐式类型转换导致重载函数产生二义性

示例8-1-3中，第一个output函数的参数是int类型，第二个output函数的参数是float类型。由于数字本身没有类型，将数字当作参数时将自动进行类型转换（称为隐式类型转换）。语句`output(0.5)`将产生编译错误，因为编译器不知道该将0.5转换成int还是float类型的参数。隐式类型转换在很多地方可以简化程序的书写，但是也可能留下隐患。

```
#include void output( int x). // 函数声明 void output( float x). // 函数声明 void output( int x) { cout
```