本章介绍如何定义和声明函数。我们将介绍如何将参数传递给函数以及如何从函数返回值。在C++中，函数可以被重载，这意味着我们可以为几个不同的函数使用相同的名称。我们将讨论如何重载函数以及编译器如何从多个重载函数中为特定调用选择匹配的版本。本章最后描述了指向函数的指针。.

我们通过调用操作符执行一个函数，它是一对括号。调用操作符接受一个表达式，该表达式是一个函数或指向一个函数。括号内是一个以逗号分隔的参数列表。参数用于初始化函数的参数。调用表达式的类型是函数的返回类型。

函数调用做两件事：从相应的实参中提取函数的形参，并将控制权转移给该函数。调用函数的执行被挂起，被调用函数的执行开始。

函数的执行从（隐式）定义和初始化其参数开始。因此，当我们调用fact时，发生的第一件事是创建一个名为瓦尔的int变量。这个变量由fact调用中的参数初始化，在本例中为5。

当遇到return语句时，函数的执行结束。与函数调用类似，return语句做两件事：返回返回值（如果有的话），并将控制权从被调用函数转移回调用函数。函数返回的值用于初始化调用表达式的结果。执行将继续执行调用所出现的表达式的剩余部分。

每个实参的类型必须匹配相应的形参，就像任何初始化器的类型必须匹配它所构造的对象的类型一样。我们必须传递与函数的参数数量完全相同的参数。因为每个调用都保证传递与函数具有的参数一样多的参数，所以参数总是初始化的。

参数名称是可选的。但是，无法使用未命名的参数。因此，参数通常都有名称。有时候，函数有一个参数没有被使用。这样的参数通常不命名，以表明它们没有被使用。保留参数未命名不会更改调用必须提供的参数数量。调用必须为每个形参提供一个实参，即使该形参没有被使用。

在C++中，名字有作用域（§ 2.2.4），对象有生存期。理解这两个概念很重要。

·名称的作用域是程序文本中该名称可见的部分。·

对象的生存期是对象在程序执行期间存在的时间。

文本

描述已自动生成

在任何函数之外定义的对象存在于程序的整个执行过程中。这些对象在程序开始时创建，直到程序结束时才被销毁。局部变量的生存期取决于它是如何定义的。

文本

描述已自动生成

对应于函数参数的自动对象由传递给函数的参数初始化。与局部变量对应的自动对象如果其定义包含初始化式，则会被初始化。否则，它们被默认初始化（第2.2.1节），这意味着未初始化的内置类型的局部变量具有未定义的值。

如果有一个局部变量，其生存期在函数的调用之间延续，则会很有用。我们通过将局部变量定义为static来获得这样的对象。每个局部静态对象在第一次执行通过对象的定义之前被初始化。局部静态数据不会在函数结束时被销毁;它们会在程序终止时被销毁

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

将函数声明直接放在使用该函数的每个源文件中可能是诱人的，也是合法的。然而，这样做是乏味和容易出错的。当我们为函数声明使用头文件时，我们可以确保给定函数的所有声明都一致。此外，如果函数的接口发生更改，只需更改一个声明。

声明函数的头文件应该包含在定义该函数的源文件中。

如果我们只修改了一个源文件，我们只想重新编译实际修改过的文件。大多数编译器都提供了一种单独编译每个文件的方法。此过程通常会生成一个文件，其扩展名为.obj（Windows）或.o（UNIX），表示该文件包含目标代码。

参数初始化的工作方式与变量初始化相同

与任何其他变量一样，参数的类型决定了参数与其实参之间的交互。如果形参是一个引用（第2.3.1节），那么形参被绑定到它的实参上。否则，复制参数的值。

当参数值被复制时，形参和参数是独立的对象。我们说这样的参数是“按值传递”，或者说函数是“按值调用”。

Pointer Parameters

指针（第2.3.2节）的行为与任何其他非引用类型一样。当我们复制一个指针时，指针的值被复制。复制后，两个指针是不同的。然而，指针也使我们间接访问指针所指向的对象。我们可以通过指针赋值来更改该对象的值

在C++中，程序员通常使用引用参数。

在这个调用中，参数i只是j的另一个名字。在reset中使用i就是使用j。

复制大型类类型或大型容器的对象可能效率低下。此外，某些类类型（包括IO类型）不能被复制。函数必须使用引用参数对不能复制的类型的对象进行操作

**如果一个引用参数在函数内部不会被修改，那么它应该被声明为 const 引用**。

使用引用参数返回附加信息。引用能使函数返回多个值。

const参数和参数

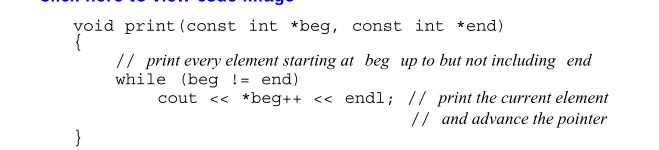
图形用户界面, 文本

描述已自动生成

尽可能使用Reference to const

组有两个特殊的属性影响我们如何定义和使用对数组进行操作的函数：我们不能复制数组（第3.5.1节），当我们使用数组时，它（通常）被转换为指针（第3.5.3节）。

与任何使用数组的代码一样，接受数组参数的函数必须确保数组的所有使用都在数组边界内。



The parentheses around &arr are necessary

在argv中使用参数时，请记住可选参数开始在argv[1]中; argv[0]包含程序名，而不是用户输入。

省略号参数应该只用于C和C++通用的类型。特别是，大多数类类型的对象在传递给省略号参数时都不能正确复制。

void函数使用return在中间点退出函数。return的这种用法类似于使用break语句（第5.5.1节）退出循环。

返回的值必须与函数返回类型具有相同的类型，或者它必须具有可以隐式转换（第4.11节）为该类型的类型。

在包含返回值的循环后未能提供返回值是错误的。但是，许多编译器不会检测到此类错误。

**确保返回引用安全的一种好方法是问：这个引用指向的是哪个已经存在的对象？**

文本

描述已自动生成

递归函数中必须有一条不涉及递归调用的路径;否则，函数将“永远”递归，这意味着函数将继续调用自己，直到程序堆栈耗尽。这样的函数有时被描述为包含递归循环。在阶乘的情况下，当瓦尔为1时发生停止条件。

因为我们不能复制数组，所以函数不能返回数组。然而，函数可以返回一个指针或数组的引用（第3.5.1节）。不幸的是，用于定义返回指针或数组引用的函数的语法可能令人生畏。幸运的是，有一些方法可以简化这种声明。最直接的方法是使用类型别名（第2.5.1节）

6.4. Overloaded Functions

尽管重载让我们避免了为普通操作命名，但我们应该只重载那些实际上做类似事情的操作。在某些情况下，提供不同的函数名会增加使程序更容易理解的信息。考虑一组在屏幕上移动光标的函数。

然而，通过重载这些函数，我们丢失了函数名中固有的信息。虽然光标移动是所有这些函数共享的通用操作，但该移动的特定性质对于这些函数中的每一个都是唯一的。例如，moveHome表示光标移动的一个特殊实例。是否重载这些函数取决于这两个调用中哪一个更容易理解：

现在，重要的是要认识到，对于任何给定的调用重载函数，有三种可能的结果：·编译器找到一个函数，它是实际参数的最佳匹配，并生成代码来调用该函数。·没有函数的参数与调用中的参数匹配，在这种情况下，编译器会发出错误消息，指出没有匹配。·存在多个匹配的函数，并且没有一个匹配明显是最佳的。这种情况也是一个错误;它是一个二义性调用。

通常，在局部声明函数是一个坏主意。然而，为了解释作用域如何与重载交互，我们将违反这一惯例，使用局部函数声明。

在C++中，名称查找发生在类型检查之前。

我们可以为一个或多个参数定义默认值。但是，如果一个参数有一个默认参数，那么它后面的所有参数也必须有默认参数。

设计带有默认参数的函数的部分工作是对参数进行排序，以便最不可能使用默认值的参数出现在第一位，最可能使用默认值的参数出现在最后。

我们不能改变已经声明的默认值：

但我们可以添加一个默认参数如下：

默认参数通常应该在适当的头文件中用函数声明来指定。

将shorterString作为函数有一个潜在的缺点：调用函数往往比计算等效表达式要慢。在大多数机器上，函数调用做了很多工作：寄存器在调用前保存，并在返回后恢复;参数可能被复制;程序分支到一个新的位置。

内联函数避免函数调用开销

内联规范只是对编译器的一个请求。编译器可以选择忽略此请求。

一般来说，内联机制旨在优化频繁调用的小型直线函数。许多编译器不会内联递归函数。

constexpr函数不需要返回常量表达式。

**当一个函数有默认参数时，函数调用时看起来传递的参数可能比实际需要的参数少**。

如果没有可行的函数，编译器将抱怨没有匹配的函数。

**在调用重载函数时，不应该需要类型转换（cast）。如果需要类型转换，通常表明参数集设计得不够好**

在函数匹配的上下文中，内置类型之间的提升和转换可能会产生令人惊讶的结果。幸运的是，设计良好的系统很少包含像下面的例子中那样具有密切相关参数的函数。

\* pf周围的括号是必要的。如果我们省略括号，那么我们将pf声明为一个返回bool指针的函数：

函数是命名的计算单元，对于构造最简单的程序也是必不可少的。每个函数都有一个返回类型，一个名称，一个（可能为空）参数列表，以及函数体。函数体是调用函数时执行的块。调用函数时，传递给函数的实参必须与相应参数的类型兼容。在C++中，函数可以重载：只要函数中参数的数量或类型不同，就可以使用相同的名称来定义不同的函数。编译器会根据调用中的参数自动确定要调用的函数。从一组重载函数中选择正确函数的过程称为函数匹配。