程序员倾向于将运行他们的应用程序的人视为用户。类似地，类设计器为类的用户设计并实现类。在这种情况下，用户是程序员，而不是应用程序的最终用户。

当我们提到用户时，上下文会清楚地表明所指的是哪种用户。如果我们谈到用户代码或Sales\_data类的用户，我们指的是使用类的程序员。如果我们谈到书店应用程序的用户，我们指的是运行应用程序的商店经理。

C++程序员倾向于将用户互换地称为应用程序的用户或类的用户。

在简单的应用程序中，类的用户和类的设计者可能是同一个人。即使在这种情况下，保持角色的不同也是有用的。当我们设计一个类的接口时，我们应该考虑使用这个类有多容易。当我们使用类时，我们不应该考虑类是如何工作的。

设计者密切关注使用该类的程序员的需求。一个设计良好的类有一个直观且易于使用的接口，并且有一个足够有效的实现来实现其预期用途。

我们以类似于普通函数的方式定义（参见6.1节）和声明（参见6.1.2节）成员函数。成员函式必须在类别内宣告。成员函式可以定义在类别本身内部或类别主体外部。属于界面一部分的非成员函式（例如add、read和print）是在类别外部宣告和定义的。

类中定义的函数是隐式内联的

**类内定义的成员函数**（即在类的定义体中直接实现的函数）**默认是隐式内联（implicitly inline）的**。这意味着编译器会尝试将这些函数的代码直接插入到调用它们的地方，而不是通过常规的函数调用机制。

当isbn引用Sales\_data的成员时（例如，bookNo），它隐式地引用调用该函数的对象的成员。在这个调用中，当isbn返回bookNo时，它隐式地返回total.bookNo。

成员函数通过一个名为this的额外隐式参数访问调用它们的对象。当我们调用一个成员函数时，它会被初始化为调用该函数的对象的地址。

Sales\_data::isbn(&total)

this参数是为我们隐式定义的。实际上，我们定义一个名为this的参数或变量是非法的。在成员函数体内，我们可以使用它。将isbn定义为

const成员函数

关于isbn函数的另一个重要部分是参数列表后面的关键字const。该常量的目的是修改隐式this指针的类型。

虽然这是隐式的，但它遵循正常的初始化规则，这意味着（默认情况下）我们不能将其绑定到const对象。这一事实反过来意味着我们不能在const对象上调用普通的成员函数。

默认情况下，this是个顶层的const指针，我们不能修改它指向其他类，函数体后面假如const后，添加了底层const，这样就不能修改指针指向的对象

const对象和指向const对象的引用或指针只能调用const成员函数。

类的成员函数的定义嵌套在类本身的作用域中。因此，isbn对名称bookNo的使用被解析为在Sales\_data中定义的数据成员。

值得注意的是，isbn可以使用bookNo，即使bookNo是在isbn之后定义的。正如我们将在§ 7.4.1中看到的，编译器分两步处理类--首先编译成员声明，然后处理成员函数体（如果有的话）。因此，成员函数体可以使用它们类中的其他成员，而不管这些成员出现在类中的什么位置。

与任何其他函数一样，当我们在类体外部定义成员函数时，成员的定义必须与其声明相匹配。也就是说，返回类型、参数列表和名称必须与类体中的声明相匹配。如果成员被声明为const成员函数，则定义还必须在参数列表后指定const。在类外部定义的成员的名称必须包括它所属的类的名称

一旦编译器看到函数名，代码的其余部分就被解释为在类的作用域内。因此，当avg\_price引用revenue和units\_sold时，它隐式地引用Sales\_data的成员。

**按值传递（Pass by Value）**

按值传递时，右值会被拷贝或移动到函数的参数中。

文本, 信件

描述已自动生成

MyClass& 作为函数的返回类型，**限定了函数必须返回一个 MyClass 类型的引用**。具体来说，它要求函数返回一个指向 MyClass 对象的引用，而不是返回一个副本或临时对象。

通常，作为类接口一部分的非成员函数应该在类本身的同一个头中声明。

**通用翻译**

鲸译AI



**进阶翻译**

鲸译AI计算机科学

但是，关于这些功能，有两点值得注意。首先，读和写都引用它们各自的IO类类型。IO类是无法复制的类型，因此我们只能通过引用传递它们（第6.2.2节，第210页）

每个类定义了如何初始化它的类型的对象。类通过定义一个或多个称为构造函数的特殊成员函数来控制对象初始化。构造函数的工作是初始化类对象的数据成员。每当创建类类型的对象时，都会运行构造函数。

* 在构造过程中，构造函数可以修改对象的状态，包括 const 成员变量。
* 这是因为在构造函数完成之前，对象的“常量性”还没有生效。

默认构造函数是不带参数的构造函数。

其中之一是如果我们的类没有显式定义任何构造函数，编译器将隐式地为我们定义默认构造函数。编译器生成的构造函数被称为合成默认构造函数。对于大多数类，这个合成构造函数将类的每个数据成员如下所示：

·如果有类内初始化器（第2.6.1节），用它来初始化成员。·否则，默认初始化（第2.2.1节）成员。

类必须定义自己的默认构造函数的最常见原因是，只有当我们不为类定义任何其他构造函数时，编译器才会为我们生成默认构造函数

只有当类没有声明构造函数时，编译器才会自动生成默认构造函数。

因此，具有内置或复合类型成员的类通常应该在类内部初始化这些成员，或者定义自己版本的默认构造函数。否则，用户可以创建具有未定义值的成员的对象。

构造函数不应重写类内初始值设定项，除非使用不同的初始值。如果不能使用类内初始化器，则每个构造函数都应该显式初始化内置类型的每个成员。