重载的运算符是有特殊名字的函数：名字由关键字 operator 和要定义的运算符号组成。重载的运算符作为函数，也包含返回类型、参数列表和函数体。

ps：

1.重载运算符的参数数量和该运算符作用的运算对象数量一样多。

2.除了重载的函数调用运算符 operator() 之外，其他重载运算符不能含有默认实参。

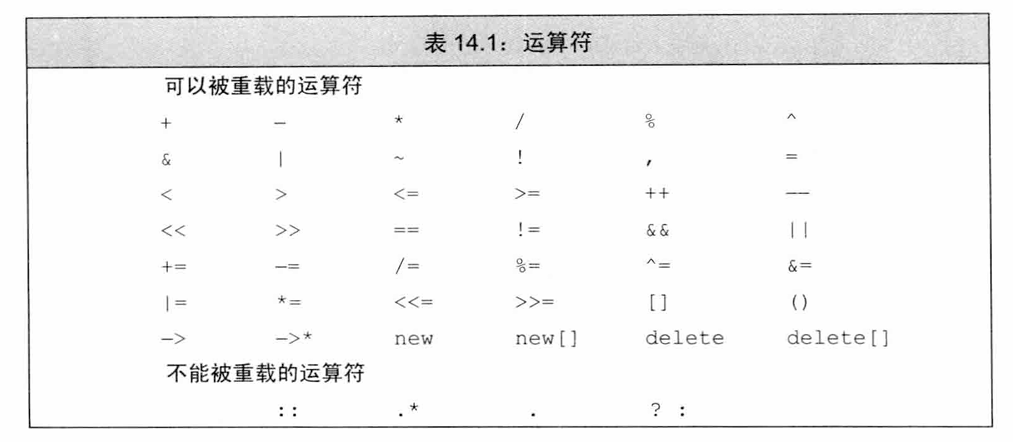
对一个运算符函数，参数数量应与参与该运算符作用的运算对象数量一样多。并且对于一个运算符函数，它应当是类的成员或者至少含有一个类类型参数。

ps：不能且不允许改变内置类型的运算符含义。

可被重载和不可被重载的运算符：

ps：1.只能重载已有的运算符，无权发明新的符号。

2.重载的运算符的优先级和结合律应与内置类型保持一致。



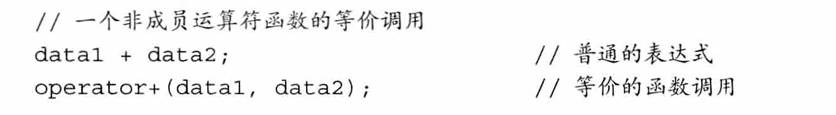
不应该被重载的运算符：

1.即使有些运算符可以被重载，但仍不建议重载，因为他们已经有明确的内置的定义，若它们的行为异于常态，会给使用者带来困惑与不适应，比如 & 和 逗号 。

2.某些运算符指定了对象求值的顺序，这些运算对象求值顺序无法应用到重载的运算符上，如 && 、|| 、&、|、逗号（逗号运算符用于在一个表达式中计算多个子表达式，并且返回最后一个子表达式的结果），这些运算符的求值顺序无法保留。

image.png

调用重载的运算符函数的两种方法：



总结，重载运算符的几个规则：

1.通常不应该重载逗号，取地址，逻辑与和逻辑或运算符。

2.重载的运算符应该使用和内置类型一样的含义，例如移位运算符使其与内置类型IO保持一致。

3.如果定义了 ==，一般也应该定义 !=。

4.如果定义了 <，也应该有其他关系操作。

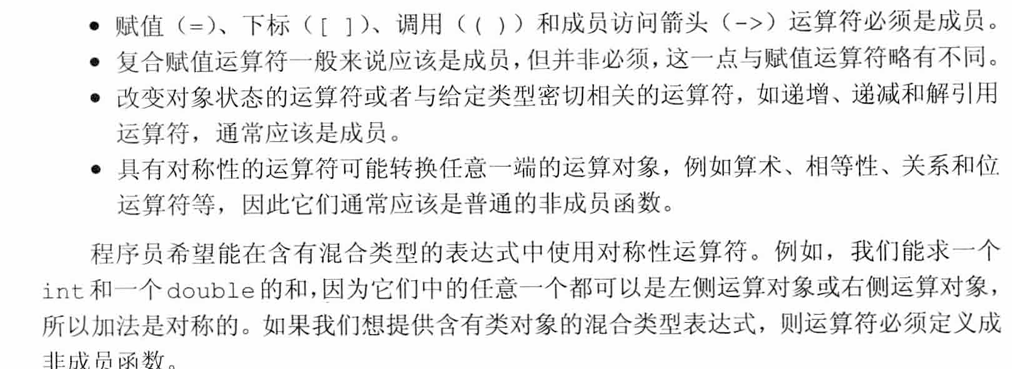
5.如果定义了算术运算符或位运算符，也应该有对应的复合赋值运算符，如有 +，也应该有 +=。

6.重载运算符的返回类型应该与内置类型的返回类型兼容。如逻辑和关系运算符返回 bool、如赋值符号重载返回引用，赋值和复合赋值运算符返回左侧运算对象的一个引用。

7.当运算符定义为成员函数，它的左侧运算对象必须是所属类的一个对象。比如下面这样，第一个参数必须是所属类的一个对象。

image.png

下面的情况，使用非成员函数的运算符重载比使用成员函数更好：



输入和输出运算符的重载：

重载输出运算符：

声明方式：

通常输出运算符的第一个形参是一个非常量 ostream 对象的引用（因为 IO 类型不能拷贝，所以必须是引用，因为要通过向流写入来输出，这会改变流的状态，所以必须是非常量），第二个形参一般是常量引用（为了避免复制实参，所以应为引用）。重载的 << 应该返回它的 ostream 形参。

image.png

ps：1.重载输出运算符应该尽量减少格式化操作，只负责打印什么内容，而不控制格式。（例如不应该打印换行符、制表符等）。

2.IO运算符重载必须是非成员函数，且应该声明为类的友元。（在ClassType这个类里声明为友元）

重载输出运算符：

声明方式：

通常输入运算符的第一个形参是一个非常量 istream 对象的引用，第二个形参是要读入的非常量对象的引用。重载的>>返回 istream 对象的引用。

image.png

输出运算符重载函数定义时需要注意的地方：

1.输入运算符必须检查是否输入成功，并处理输入失败的情况，而输出运算符不需要。

2.如果读取失败，输入运算符应该负责从错误中恢复，主要是将输入对象重置为合法状态（一般为未输入前的状态）。

输入流会产生的错误：

1.当流中含有错误的数据时可能会读取失败

2.当读取操作到达文件末尾时会失败

3.遇到输入流的其他错误时也会失败

算术和关系运算符

算术和关系运算符通常定义为非成员函数以允许对左侧或右侧的运算对象进行转换,且两个形参都应该是常量引用。

相等运算符

下面是书上例子：这是一对相等运算符和不等运算符

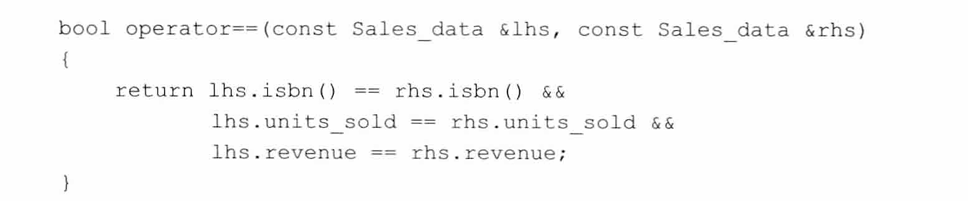
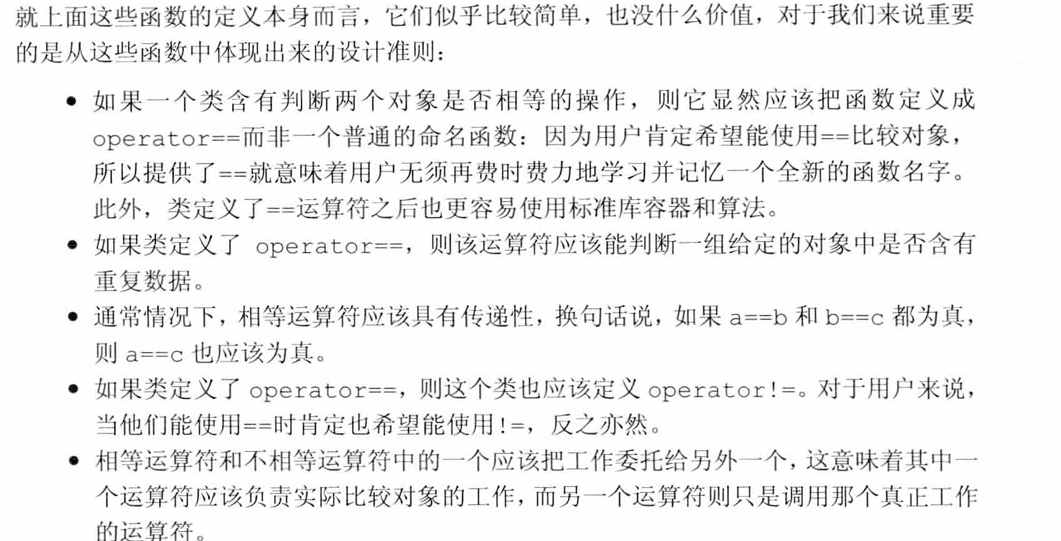


image.png



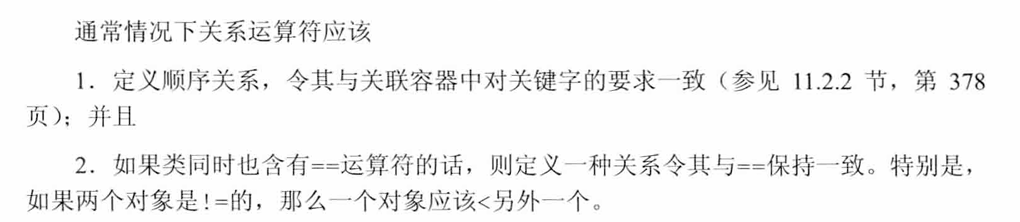
从上面的例子得出的设计准则：



关系运算符

定义了 == 的类，经常也会包含关系运算符，尤其是 <，因为关联容器和一些算法都要用到 <。（定义了 < 也应该定义与之相对的 >）因此应定义为类的成员函数。

关系运算符定义的准则：



赋值运算符

赋值运算符必须定义为类的成员函数，且应该返回左侧运算对象的引用。

类中一般已经定义了拷贝赋值和移动赋值运算符。如果需要时也可以继续重载赋值运算符以使用别的类型作为右侧运算对象。

ps：赋值运算符必须先释放当前的内存空间，再重新分配。

复合赋值运算符

复合赋值运算符可以不是类的成员函数，但是一般应该定义为成员函数。其左侧运算对象绑定到隐式的 this 指针。

复合赋值运算符也返回左侧运算对象的引用。

下标运算符

下标运算符的含义：表示容器的类可以通过元素在容器中的位置来访问元素，这些类一般会定义下标运算符 operator[ ]。下标运算符重载必须是成员函数。

下标运算符通常以所访问元素的引用作为返回值，这样下标可以出现在赋值运算符的任意一端。

下标运算符最好同时定义两个版本：

非常量版本：返回普通引用。

常量版本：是类的常量成员并返回常量引用。常量版本取得的元素不能放在赋值运算符的左侧。

递增和递减运算符

递增和递减运算符通常定义为成员函数，但不是必须的。

递增和递减运算符应该同时定义前置和后置版本。且前置版本返回递增或递减后的引用，后置版本返回修改前的副本。

如果是用于迭代器的递增递减运算符，应该检查递增递减是否合法，如 0 不能递减。

区分前置和后置

为了区分，可以使后置版本接受一个额外的不被使用的 int 类型的形参，当使用后置版本时，编译器为这个形参提供一个值为 0 的实参。

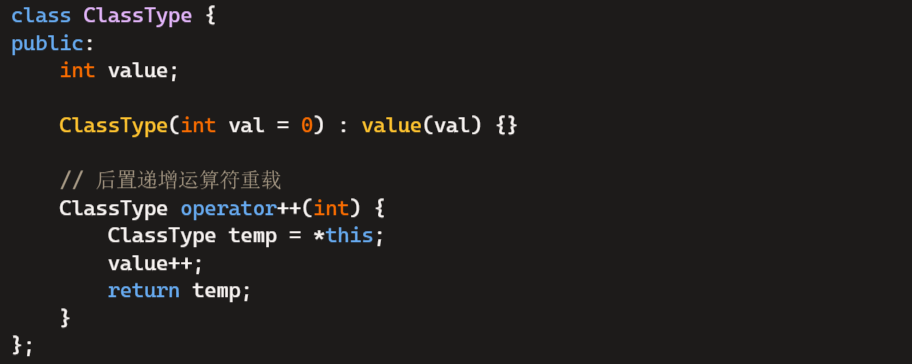
这个形参的唯一作用就是区分前置和后置。因为不会用到，所以该形参无需命名。

前置递增运算符：



后置递增运算符：

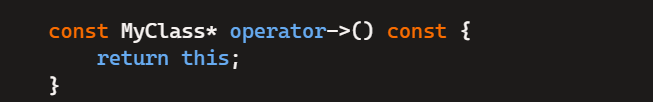
后置版本可以通过调用前置版本来实现。



成员访问运算符

箭头运算符必须是类的成员，箭头运算符一般通过调用解引用运算符来实现。解引用运算符通常也是类的成员，但不必须。

对于箭头运算符重载函数，最好把函数声明为const，因为->通常只作左值使用，一般只用于访问成员，这需要一个持久的变量支持



函数调用运算符

如果类重载了函数调用运算符，就可以像使用函数一样使用该类的对象 (被称为函数对象)。

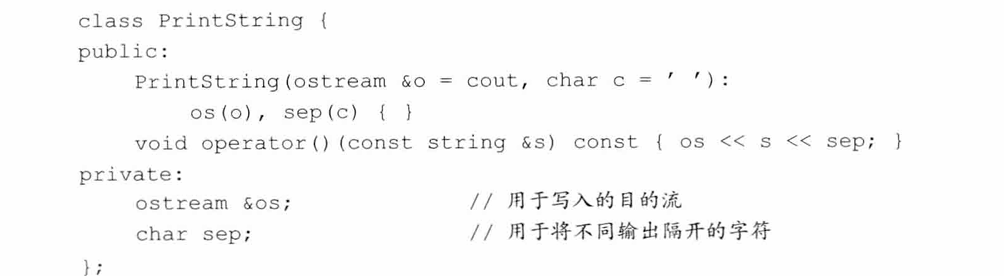
函数调用运算符必须是成员函数，一个类可以定义多个版本的调用运算符。不同版本的参数应有所区别。

ps：当重载了函数调用运算符，类对象的名字就相当于函数名。

含有状态的函数对象类

函数对象类通常会有一些数据成员，这些数据成员帮助实现函数。

函数对象常作为泛型算法的实参。



lambda函数对象

Lambda表达式是一种在被调用的位置或作为参数传递给函数的位置定义匿名函数对象（闭包）的简便方法。Lambda表达式的基本语法如下：

image.png

capture list 是捕获列表，用于指定 Lambda表达式可以访问的外部变量，以及是按值还是按引用的方式访问。捕获列表可以为空，表示不访问任何外部变量，也可以使用默认捕获模式 & 或 = 来表示按引用或按值捕获所有外部变量，还可以混合使用具体的变量名和默认捕获模式来指定不同的捕获方式。

parameter list 是参数列表，用于表示 Lambda表达式的参数，可以为空，表示没有参数，也可以和普通函数一样指定参数的类型和名称。

return type 是返回值类型，用于指定 Lambda表达式的返回值类型，可以省略，表示由编译器根据函数体推导，也可以使用 -> 符号显式指定。

function body 是函数体，用于表示 Lambda表达式的具体逻辑，可以是一条语句，也可以是多条语句。

当 lambda 通过引用捕获变量时，由程序确保 lambda 执行时所引用的对象确实存在。(联想到C#中的闭包，C#中的闭包会延长对象生命周期，不需要由程序员确保对象的存在)

当 lambda 通过值捕获变量时，捕获的变量被拷贝到 lambda 中，此时产生的类中会建立对应的数据成员并创建构造函数来初始化数据成员。

编译器会将 lambda 翻译成一个未命名类的未命名对象，在 lambda 表达式产生的类中含有一个重载的函数调用运算符。Lambda表达式的捕获列表实际上是匿名类的数据成员，Lambda表达式的参数列表和返回值类型实际上是匿名类的 operator() 的参数列表和返回值类型，Lambda表达式的函数体实际上是匿名类的 operator() 的函数体。

默认情况下 lambda 不能修改它的捕获变量(外层函数的局部非 static 变量)，lambda 产生的类中的函数调用运算符（即()运算符）是一个 const 成员函数（如果使用mutable，则不为const）。

标准库的functional头文件定义了一组表示算术运算符、关系运算符和逻辑运算符的类



ps：标准库规定的函数对象对于指针也是适用的，比如可以使用 less 来比较两个指针的大小（比较的是指针变量中的地址大小）。因为关联容器使用 less 来对元素排序，如果将指针作为 set 或 map 的关键字，元素将自动按地址进行排序

不同类型可能具有相同的调用形式

调用形式指明了调用返回的类型和传递给调用的实参类型。

image.png

有时会希望把这有相同签名的函数看成具有相同的类型。此时需要定义一个函数表用于存储指向这些可调用对象的指针。

ps：函数的签名包括函数名，参数列表和返回类型函数调用修饰符

可以使用map来完成函数表，下面是一个例子：

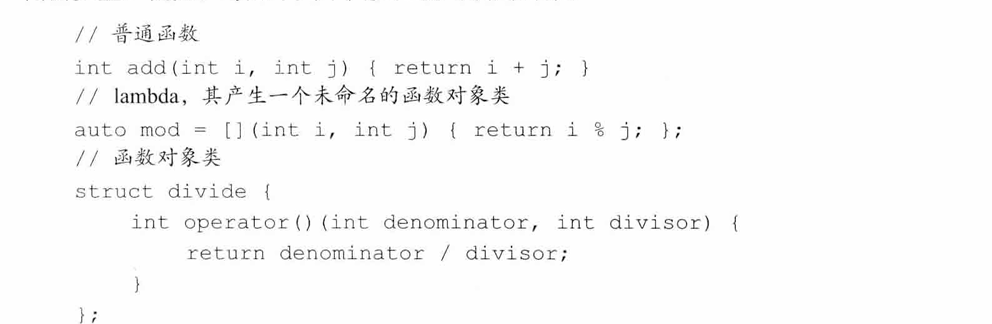


image.png

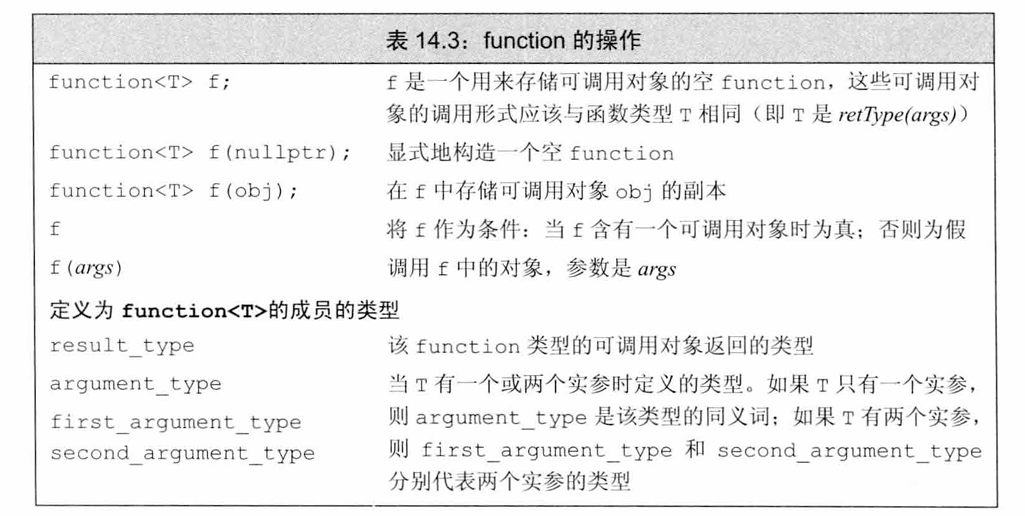
表的值分别对其参数执行了不同算数运算，即使表的值对应了不同类型的函数（能这么做的原因是它们的函数类型都是int(int,int)，map代表指向这类函数的指针）

但存在问题，如果是一个lambda表达式，是没有办法存入map的，因为map需要接受一个明确的类型；也没有办法存入divide，因为divide()的类型是divide本身。

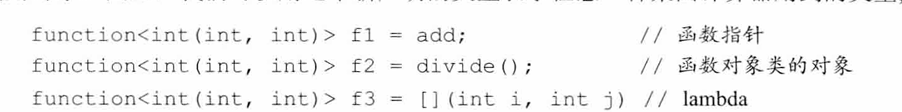
解决上面问题的方式是使用function类型包装：

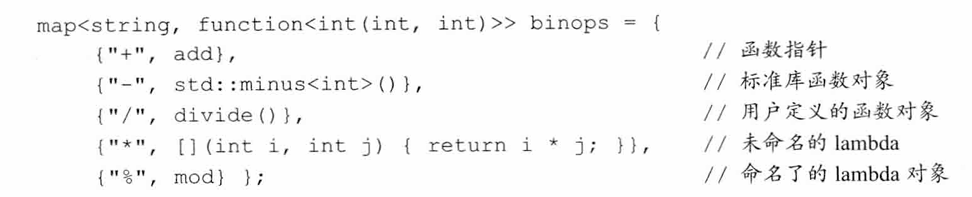
function定义在functional头文件中。function 是一个模板，创建具体的 function 类型时要指明具体的调用形式。

function类型提供的操作：



使用function包装后解决上面的问题：





ps：存在重载函数时，不能直接将重载函数的名字存入 function 类型的对象中，存在二义性，但是可以存储指向确定重载版本的函数指针。或者使用lambda来消除二义性。

类型转换运算符

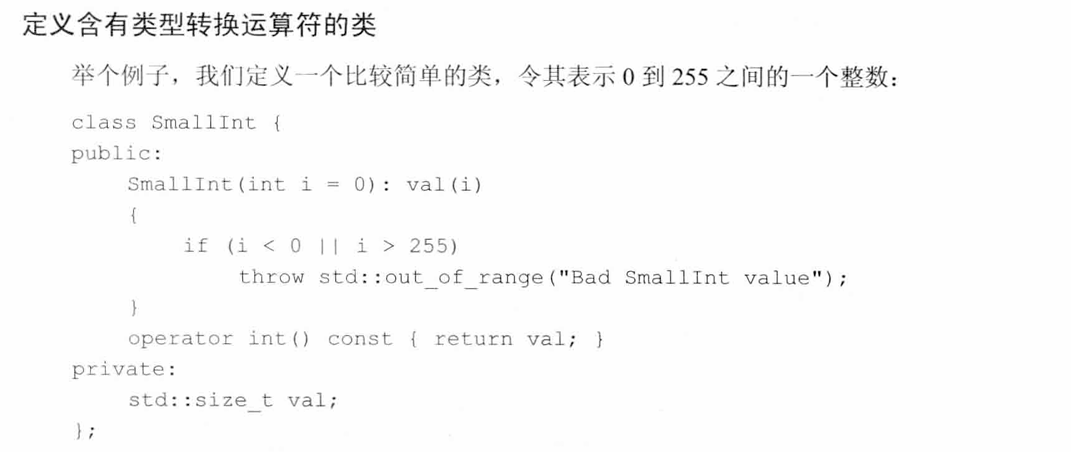
类型转换运算符是类的一种特殊成员函数，用来将一个类类型的值转换成其他类型。可以转换成除了 void 外任意可以作为函数返回值的类型，包括引用、指针，因此不可以是数组或者函数类型。

类型转换函数的一般形式：operator type() const;

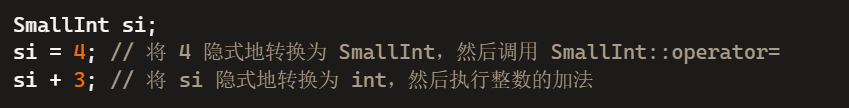
类型转换函数必须是类的成员函数，不能声明返回类型，形参列表也必须为空，且函数一般是 const 的。不指定返回类型，但是函数会返回一个对应类型的值。

ps：函数名实际上就是返回类型

定义含有类型转换运算符的类：



使用隐式的类型转换运算符：

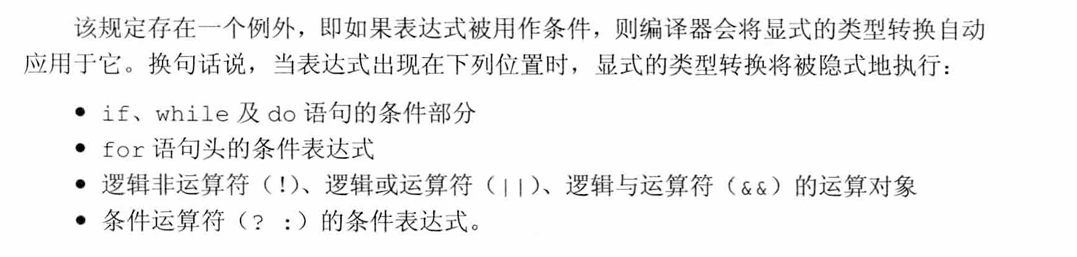


显式的类型转换运算符

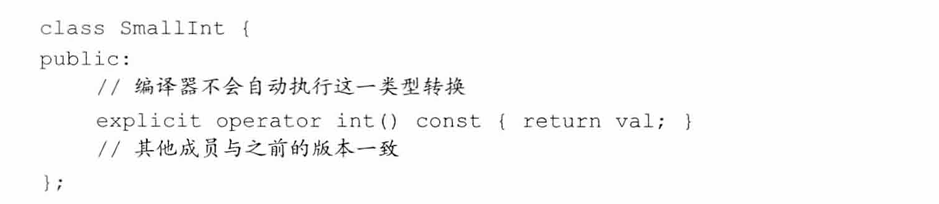
使用显式的类型转换运算符来避免在不合适的场合发生转换。

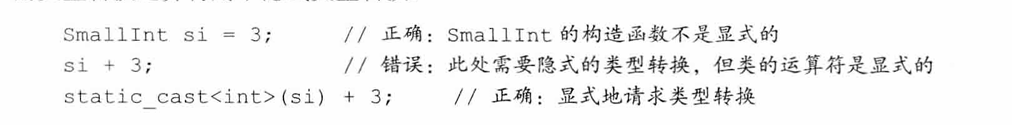
用关键字 explicit 来将运算符指定为显式的，调用时需使用 static\_cast 来显式转换。

如果表达式被用作条件，则显式的类型转换会被隐式地执行。比如用在 if 语句的条件部分。向 bool 类型的转换一般都用于条件部分，因此 operator bool() 一般定义成 explicit 的。



使用显式转换运算符的例子：





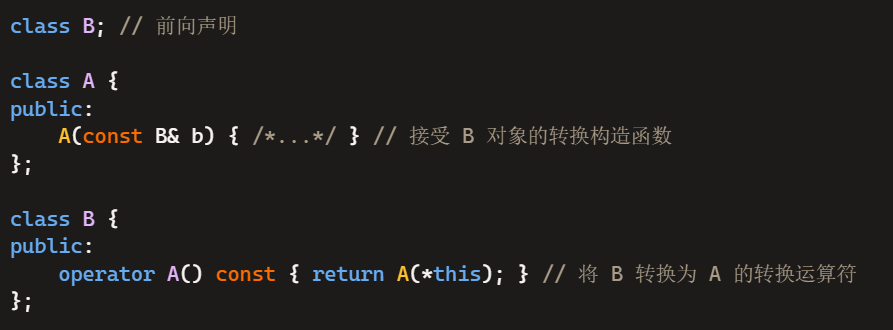
避免有二义性的类型转换

要确保在类类型和目标类型之间只存在唯一的转换方式。

有两种情况可能产生多重转换路径：

1.两个类提供相同的类型转换。A 类定义了一个接受 B 类对象的转换构造函数，同时 B 类定义了一个转换目标是 A 类的类型转换运算符。

下面例子：有两个类 A 和 B，其中 A 有一个接受 B 对象的构造函数，而 B 有一个转换运算符，可以将其转换为 A 对象。有一个 B 类型的对象，并且尝试将其赋值给 A 类型的对象，会产生二义性，因为编译器不知道是应该调用 A 的构造函数还是 B 的转换运算符。



2.定义了多个转换规则。

比如定义了一个转换为int的的转换运算符，再定义一个转化为double的转换运算符

设计重载运算符和类型转换函数时要加倍小心，避免产生二义性。建议：

1.不要令两个类执行相同的类型转换

2.避免转换目标是内置算数类型的类型转换。特别是已经定义了一个转换成算数类型的类型转换时。

3.当有重载函数时，定义类型转换运算符尤其容易发生错误。

函数匹配与重载运算符

重载的运算符也是重载的函数，因此也使用函数匹配规则。

使用重载运算符作用域类类型的运算对象时，候选函数集中包含该运算符的普通非成员版本和类的成员函数版本。

