操作符

Item 5 对于定制的“类型转换函数”保持警觉

`operator`

对于隐式转型，大部分人习以为常又担惊受怕，因为总是会有一些奇特的情况让你的程序不受控制，而这些不受控制的结果往往会影响整个程序。当然大部分的情况下影响不是很大。从某种程度上说，隐式转型给予语言极大的灵活性，但是也要求程序员成为一个神。

既然已经有麻烦了，为什么要自找更多的麻烦呢?

加入我们写了一个分数类`Rational`，显然要求实现转成小数的实现，如果你了解转型的操作符，你可能会这么写。

```
operator double()const;
```

但是这会导致一个问题，在你不希望的时候也有可能会产生这样的调用，比如`cout`而且你没有实现流输出操作符的时候。而这一类不会报错的非预期行为极难查出。

实际上，只要避免这玩意就行了，比如使用`double asDouble()const`来替换，毕竟这玩意不存在默认实现的，而且在代码中这样的调用绝不会隐式，而是非常显眼的调用方法。

constructor

初次以外，还有一类东西也是非常容易产生非预期行为的，就是单元素的构造函数，产生的原因对于这个层级的人来说不是一个大难题，通常的解决方法也有两个。

- 关键词`explicit`。
- 把那个参数单独打包形成构造函数。这是Proxy class的一种用法。

在本书之外，我们还要解决一个比较大的问题：我写了一个`int`的构造函数，但是我不希望对`double`有影响，又希望能让`int64_t`对此有反应，我应该怎么实现？这也是一个给予隐式转换容易产生的问题。

通常来说，解决办法是`template`，利用第二个参数和`std::is_same`筛掉不合适的类型，用默认值避免多于参数的麻烦。

```
template <typename T>
explicit Value(T _bool, typename enable_if<is_same<T, bool>::value>::type *
= 0)
    : _M_type(_T_Boolean), _M_val((bool)_bool) {}
template <typename T>
explicit Value(T _integer, typename enable_if<is_integral<T>::value &&
!is_same<T, bool>::value>::type * = 0)
    : _M_type(_T_Integer), _M_val((long long)_integer) {}
template <typename T>
explicit Value(T _decimal, typename
enable_if<is_floating_point<T>::value>::type * = 0)
    : _M_type(_T_Decimal), _M_val((double)_decimal) {}
```

Item 6 区别++/--的前置和后置

这似乎是一个不难的问题。我们可以调用++++i，而不能调用i++++，主要原因就是前置++的返回值是引用，而后置++返回的是const。如果返回引用则可以接着调用，反之则不行。

为了让前置和后置的操作走向一个效果（指位移），可以考虑把前置用作后置实现的一部分。反过来则不建议，因为后置产生了一个新的对象，会带来一定的复杂度。

Item 7 千万不要重载&&,||和，操作符

重置这三个玩意不是瞎扯淡么。

- 原本的&&和||是短路运算符，如果重载岂不是两个都要算了。有大量的代码基于短路模式进行计算，但是重载无法模拟这种操作。
- 重载逗号操作符意味着必须要进行类似的模仿，但是这也很显然是模仿不出来的。
- 操作符重载的目的是让程序更容易被阅读、被撰写、被理解，而不是为了重载而去重载。

Item 8 了解不同意义的new和delete

- new和operator new：new包括了开辟内存，初始化和指向，而operator new仅仅开辟了内存。
    - 仅仅编译器可以完成对特定对象构造函数调用，如果希望调用的可以考虑下后面的placement new。
    - operator new可以重载，也可以写更多的，但是第一个参数必须是size_t。
- placement new是operator new的一种重载形式。
    - 其他两种重载方式分别是只有size_t和size_t、nothrow_t版本。
    - placement new的行为和operator new不一样是因为重新定义了行为，而不是因为这是另外的一套体系。
- delete和new搭配：这是很简单的道理，new出来的要delete掉，operator new的要operator delele掉，构造出来的要析构掉，不要反过来。
- 数组的操作是配套的，不过要加一个[]。