Point类（基础知识-关于操作符重载）

主要有三个要点

插入操作符重载

```
string Point::toString() {
    return "(" + integerToString(x) + "," + integerToString(y) + ")";
}
```

如果类中实现了 `toString` 方法，我们可以很好的将点 `(x,y)` 显示在终端上。

`cout << "pt = " << p.toString() << endl;`

操作符重载可以进一步简化这一过程。C++已经重载了流插入操作符<<，以便它可以显示字符串及基本类型数据。如果你重载了这一操作符以支持Point类，可以将上述语句简化为：

`cout << "pt = " << p << endl;`

```
ostream & operator<<(ostream &os, Point pt) {
    return os << pt.toString();
}
```

代码反应了问题：

- 流对象是不能拷贝的。这一约束意味着 `ostream` 类型参数必须采用引用传递。
- 同样地，这一约束也在插入操作符<<返回时产生影响。正如在本章开头所介绍的那样，***插入操作符通过返回输出流在连接流语句中起到重要作用，其返回结果可以参与到链中的下一个<<操作符运算。避免在该操作过程结束后拷贝流，operator<<的定义也必须引用来返回结果。***

判断两个点是否相等（重载 `==` 运算符）

给定两个点p1和p2，可以采用 `==` 作符来判断这两个点是否相等，这与检验字符串和基本类型是否相等一样。

C++供了两种机制用以重载内置的操作符，以保证它们可适用于新定义类的对象：

1. 可以在类中用一个方法来重载一个操作符。当采用这种方式在类中重载一个二元操作符时，其左操作数为该类型的对象，而右操作数则作为形参传递进来。
2. 可以在类外使用一个自由函数（free function）来重载定义一个操作符。如果采用这种方式，则二元操作符的两个操作数都必须通过形参传递进来。

```
bool Point::operator==(Point rhs) {
    return x == rhs.x && y == rhs.y;
}
```

> 为什么方法可以访问rhs的私有变量？

`operator==` 方法的代码展示了面向对象编程的一个重要特性。 `operator==` 方法显然可以访问当前对象的x和y域，因为一个类中的任意方法可以访问该类的私有变量。其中比较难理解的一点是，为何 `operator==` 方法在所属对象完全不同的情况下也可以同时访问rhs对象的私有变量。在C++中，这种引用是合法的，因为类中定义的私有部分对该类来说是私有的，**但对对象而言并非如此。**类中方法的代码可以引用该类型的任何对象的实例变量。

```
bool operator==(Point p1, Point p2) {
    return p1.x == p2.x && p1.y == p2.y;
}
```

为了使这种设计可行， Point类必须让C++编译器知道：对于一个特定的函数， `==` 操作符重载版本可适当地允许访问类中的私有实例变量。为了使这种访问合法，Point类必须将 `operator==` 函数定义为友元（friend）函数。此时，友元的特性与在社交网络中的友情特性是类似的。其私有信息一般都不会在社会中大范围地共享，而只会对你所认可的朋友开放。

在C++中，将自由函数声明为友元函数的语法如下：

`friend prototype;`

其中， prototype就是函数原型。在这个例子中，通过书写如下语句将operator==函数声明为Point类的友元函数：

```
friend bool operator==(Point p1, Point p2);
```

最后，每当为一个类重载操作符 `==` 时，最好的做法是同时为该类提供 `≠` 重载操作符。毕竟用户希望判断两个点是否不同与判断这两个点是否相同同样容易。C++并未默认==操作符和≠操作符运算返回相反的结果；如果你想要得到这种相反的结果，你必须单独重载这两个操作符。但是，在实现operator≠时，可以利用 `operator==` 的实现，因为 `operator==` 是类的一个公有方法。因此，最直截了当地重载≠操作符的函数看起来如以下代码所示：

```
bool operator≠(Point p1, Point p2) {
    return !(p1==p2);
}
```

重载前缀++与后缀++

当用C++重载++或--操作符时，必须告诉编译器你想要重载的操作符是前缀形式还是后缀形式。C++的设计者选择了通过传入一个无意义的整型参数的方式来说明其操作符为后缀形式，用以区别其前缀形式。

因此，为了重载Direction类型的**前缀++**操作符，定义如下函数：

```
Direction operator++(Direction & dir) {
    dir=Direction(dir+1);
    return dir;
}
```

为了重载其**后缀++操作符**，应如下所示定义函数：

```
Direction operator++(Direction & dir, int){
    Direction old=dir;
    dir= Direction(dir+1);
    return old;
}
```

> - 注意，dir参数必须以**引用**方式进行传递，以保证函数可以改变其变量值。
> - 在C++中，如果不需要使用形参值，则可以不用此形参名。

有理数（如何设计一个类）

定义类的机制

使用面向对象语言定义新类是你必须掌握的一项重要技能。在大多数编程中，设计新类既是一门科学更是一门艺术。开发高效的类设计需拥有良好的美学素养，同时需要考虑将类作为工具的用户的需求和期望。理论与实践是最好的老师，但是遵循一个普遍适用的设计框架可以对你设计高效的类提供帮助。

根据我个人的编程经验，我发现遵循按部就班的方法常常是很有用的：

1. ***从普遍性的角度出发，思考用户会如何使用一个类。***在设计过程的最开始，你就必须确立一种思想：类库是为了满足用户的需求而不是为了方便类的实现者。从专业的角度上说，保证新设计的类更好迎合用户需求的最佳方法就是让用户参与到设计过程中。不管怎样，你至少应该站在用户的角度来描绘类的设计蓝图。

图 6-6　有理数的算术运算法则

2. ***确定什么信息属于类的每个对象的私有部分。***虽然类中的私有部分从概念上说是类实现的一部分，但是对于该类的对象必须包括哪些信息有一个初步的了解可以简化接下来的类设计阶段。许多情况下，你可以写下将要放置在私有部分的实例变量。虽然在这个阶段并不需要做到如此清晰的划分，但是对于类的内部结构有一个清晰的印象可以使得构造函数和方法的定义变得更加简单。
3. ***定义一组重载的构造函数以创建新的类对象。***因为类通常会定义多个重载形式的构造函数，所以站在用户的角度考虑用户需要创建的对象类型及在创建对象时将会传入的信息将会非常有用。典型地，每个类都会提供一个默认构造函数，该函数允许用户声明该类对象并在之后对其进行初始化。在这个阶段，还必须思考构造函数是否需要设置约束条件来确保最终生成对象的合法性。
4. ***列举出将成为类的公有方法的所有操作。***这一阶段的目标是为类中提供的方法编写其原型，从而将你刚开始时开发的类框架转化成详细说明。也可以在这个阶段中细化整体设计，它遵循第2章提出的准则：统一性、简单性、充分性、通用性和稳定性。
5. ***编码并测试其实现。***一旦完成了类的接口设计，就需要编写代码来实现该类。编写实现过程不仅是为了得到一个可以运行的程序，还要为设计提供验证。在编写实现代码时，你还需要不时回顾其接口设计，例如，当你发现很难将设计中一种特性的性能控制在一个预定的水平时。作为一个实现者，你也有责任测试你的实现代码以确保类提供了接口中计划实现的功能。接下来的各节将遵循上述步骤来设计`Rational`类。

总结是四个方面

- 定义私有实例变量
- 定义构造函数

- 定义类方法
- 实现类方法

token扫描器类的设计

```
string lineToPigLatin(string line){
    TokenScanner scanner(line);
    string result= "";
    while (scanner.hasMoreTokens()) {
        string word = scanner.nextToken();
        if (isalpha(word[0])) word = wordToPigLatin(word);

        result += word;
    return result;
}
```

Set the input for the token scanner to be some string or input stream.
while (*more tokens are available*) {
    Read the next token.
}

这段伪码结构直接表明了 TokenScanner 类必须支持的方法。从这个例子中，可能希望 TokenScanner 类提供以下方法：

- 一个允许用户指明记号来源的 setInput 方法。理想情况下，这个方法应该重载，使得输入可以是一个字符串或者是一个输入流。
- 一个用于判断扫描器扫描过程中是否还有待扫描记号的 hasMoreTokens 方法。
- 扫描并返回下一个记号的 nextToken 方法。

除此之外的实现在cpp文件中。