OOP Exercise 4+

应承峻 3170103456

1) you can overload an operator as members or non-members, Which should you choose?

大部分运算符既可以以成员函数进行重载，也可以以非成员函数的形式进行重载。但是如果一个运算符比如（+，-，*，/，%，<，<= ，==， !=， > ，>=等）既可以以成员函数的形式也可以以非成员函数的形式重载，那么它最好以非成员函数的形式重载，因为这样可以实现自动类型的转换。

举一个简单的例子：

一般情况下使用成员函数的形式对 Rational 类中的+号进行重载，能够实现 Rational 类中的两个对象的相加：

```
Rational Rational::operator+(const Rational& secondRational) const {
    return add(secondRational);
}
```

但是如果让一个 Rational 的对象和一个 int 或 double 型的数据相加，则会编译报错：

```
Rational r1(4 , 2);
double ans = r1 + 2.3;
```

此时，我们可以定义一个运算符函数，它能够将一个 Rational 对象进行自动类型转换，转换成 int 或者 double 型的值。

```
Rational::operator double() {
    return doubleValue();
}
```

现在就实现了从对象到基本数据类型的转换，但是现在如果要计算 `r = 2.3 + r1` 就会出错，因为左边的运算对象是+运算符的调用者，它必须是一个 Rational 对象才能实现自动转换，而 2.3 并不是一个 Rational 对象，因此需要做下面两个操作：

Step1：定义构造函数

```
Rational(int numerator);
```

Step2：将+运算符定义为非成员函数

```
Rational operator+(const Rational& r1 , const Rational& r1) {
    return r1.add(r2);
}
```

从上述例子可看出，对于既可以以成员函数的形式也可以以非成员函数的形式重载的运算符，采用非成员函数重载的形式能够更方便的进行显式或者隐式的类型转换。

2) Features of operator as members and operator as non-members

重载运算符为成员函数的特征：有一个 this 指针指向自身对象，对于单目运算符来说不需要任何参数，而对于双目运算符来说只需要一个参数。例如：

```
Rational r2(2 , 3);
cout << "r2[0] is " << r2[0] << endl;
cout << "r2[1] is " << r2[1] << endl;
```

上述代码实际上与下面的代码是等价的：

```
cout << "r2[0] is " << r2.operator[](0) << endl;
cout << "r2[1] is " << r2.operator[](1) << endl;
```

可以看出，上述的双目运算符只传入了一个参数。因此可以发现，重载运算符为成员函

数时的第一个参数必须是对象，否则将会报错。

重载运算符作为非成员函数的特征：没有 this 指针，对于任何运算符来说，都需要将各个算子按顺序传入函数，例如在前面提到的：

```
Rational operator+(const Rational& r1 , const Rational& r1) {
    return r1.add(r2);
}
```

参与运算的两个算子 r1 和 r2 都按照顺序被传入。但是，在上述例子中没有涉及到私有数据的改变，因此如果该运算符涉及到私有数据的改变，则必须将非成员函数作为该类的友元，而这将会带来暴露私有数据的问题。

3)what's mean about "memberwise assignment"?

memberwise assignment 指的是逐个成员拷贝，它与 bitwise assignment 逐位成员拷贝相对应。逐个成员拷贝指的是对成员的内容进行拷贝，而逐位成员拷贝指的是对成员的地址进行拷贝。

例如，在下面的例子当中，当没有对赋值运算符“=”做重载时，执行 course2 = course1 时发生的将是逐位成员拷贝，使得 course2 中对于 course1 的 students 数据仅仅只是做了地址的拷贝，即 course2 和 course1 全部指向了储存学生的地址，所以下面的例子输出的是两行 “Lisa Ma”

```
Course course1("Java Programming" , 10);
Course course2("C++ Programming" , 14);
course2 = course1;
course1.addStudent("Peter Pan");
course2.addStudent("Lisa Ma");
cout << course1.getStudents()[0] << endl;
cout << course2.getStudents()[0] << endl;
```

当对赋值运算符“=”做重载时，代码如下：

```
const Course& Course::operator=(const Course& course) {
    if (this != &course) { //如果是自我赋值则不需要做任何事
        courseName = course.courseName;
        numberOfStudents = course.numberOfStudents;
        capacity = course.capacity;
        delete[] this->students; //删除旧的数组
        students = new string[capacity]; //重新进行数组拷贝
        for (int i = 0; i < numberOfStudents; i++) {
            students[i] = course.students[i];
        }
    }
    return *this; //返回拷贝的对象
}
```

此时程序的输出结果是”Peter Pan”和”Lisa Ma”，说明执行 course2 = course1 时发生的将是逐个成员拷贝，即 course2 中拷贝的是 course1 的成员数据而不是地址。