**我们这里说我们因该谨慎使用隐式转化**

**为什么string转换成char\*使用c\_str 这不是一个巧合**

**根本问题是当你在不需要使用转换函数时，这些的函数缺却会被调用运行。结果，这些**

**不正确的程序会做出一些令人恼火的事情，而你又很难判断出原因。**

class Rational {

public:

...

operator double() const; **// 转换 Rational 类成**

}; **// double 类型**

int mian（）{

Rational r(1, 2);

cout << r;} **//这里我们如果没有定义Rational的重载<<函数那么我们这里的本意是想输出Rational有理数，但是我们的标准库没有匹配到相关函数，那么我们编译器就会将Rational隐式转换成double类型，就会输出2分之一的浮点数。**

**我们可以通过显示定义类型转换来解决这个问题**

class Rational {

public: ...

double asDouble() const; **//转变 Rational**

}; **// 成 double**

int mian（）{

cout << r.asDouble();}

**还有一种构造函数的隐式转换**

template<class T>

class Array {

public:

Array(int size); **//Array可以向int转换**

};

**解决方法就是添加explicit**

explicit Array(int size);  **//杜绝隐式转换**

前置递增和后置递增区别

当处理用户定义的类型时，尽可能地使用前缀increment，因为它的效率较高。

**不要重载一些运算符**

**因为你写的operator提供的参数无法保证他们的计算顺序**

**短路求值法 if ((index < lowerBound) || (index > upperBound))**

如果左边没有成功那么就不会计算右边的式子所以叫做 短路求值法

if (expression1.operator&&(expression2))

但是函数调用法与短路求值法是绝对不同的。首先当函数被调用时，需要运算其所有参数，所以调用函数 functions operator&& 和 operator||时，两个参数都需要计算，换言之，没有采用短路计算法。第二是 C++语言规范没有定义函数参数的计算顺序，所以没有办法知道表达式 1 与表达式 2 哪一个先计算。完全可能与具有从左参数到右参数计算顺序的短路计算法相反。

你重载&&或||，就没有办法提供给程序员他们所期望和使用的行为特性，所以不要重载&&和||逗号运算符也不需要你提供重载运算符。