12.7-运算符重载

为运算符赋予与原运算含义不同的运算方式

可以重载的运算符几乎包含了C++的全部运算符集，C++语言规定，大多数运算符都可以重载，

**例外**的是：限定符 **.** 和 :: 条件运算符？: 取长度运算符sizeof

运算符的重载通过定义运算符的**重载函数**实现，因此运算符的重载也类似于一个特殊函数的定义过程

运算符重载函数的**声明**：

返回值类型 operator 运算符 (<参数表>);

1.函数名可以看作是：operator 重载的运算符

2.operator是关键字

3.返回值类型表示使用重载后的运算符进行运算得到结果的数据类型

说明：

1.算术运算符，逻辑运算符，位运算符等一般只能用于基本数据类型的运算，通过运算苻重载，使它们可以用于某些“用户自定义的数据类型”（结构体、类等），这是重载的主要目的。

2.单目运算符++和--均有“前缀”和“后缀”两种用法，而不同用法重载函数的函数名又完全相同，因此为了区分前缀++和后缀++，C++语言规定，在后缀++的重载函数的原型参数表中增加一个int 型的无名参数，则其原型为：

<类型> operator ++ (...,int)

代码示例：

现有两个分数a和b，通过运算符重载实现他们的加法运算a+b，并求值。

1 注意最终的结果应该是最简式，整数要以整数形式体现。

2 只考虑正数范围，不需要进行输入检验

输入

依次输入四个数，分别是a的分子、a的分母、b的分子和b的分母。

输出

输出最终的计算结果 分数用“/”表示，假分数直接表示，整数表示成整数的形式

样例输入

3 4 3 4

样例输出

3/2

#include<iostream>

using namespace std;

struct frac

{

int nume; //分子

int deno; //分母

};

int gys(int a,int b)

{

//求最大公约数

return a%b==0 ? b : gys(b,a%b);

}

int gbs(int a,int b)

{

//求最小公倍数

return a\*b/gys(a,b);

}

frac Reduction(frac &r)

{

//约分

int tmp=gys(r.nume,r.deno);

r.nume/=tmp;

r.deno/=tmp;

return r;

}

frac operator+(frac a,frac b)

{

frac res;

res.deno=gbs(a.deno,b.deno);

res.nume=a.nume\*(res.deno/a.deno)+b.nume\*(res.deno/b.deno);

Reduction(res);

return res;

}

void input(frac &r)

{

cin>>r.nume>>r.deno;

}

void output(frac r)

{

if(r.deno==1)

cout<<r.nume<<endl;

else

cout<<r.nume<<"/"<<r.deno<<endl;

}

int main()

{

frac a,b,r1,r2;

input(a);

input(b);

r1=a+b;

output(r1);

r2=operator+(a,b);

output(r2);

return 0;

}

运算符重载函数的**调用**可有两种方式：

1.与原运算符相同的调用方式，如上例中的a+b。

2.一般函数调用方式，如operator+(a,b)

被重载的运算符的调用方式，**优先级和运算顺序都与原运算符一致**，其运算分量的个数也不可改变。

/\*

运算符重载主要用于用类的形式定义的用户定义类型，例如，复数类型，集合类型，向量类型等等，通过运算符重载把人们习惯的运算符引入到计算操作之中，会收到很好的效果。

\*/