41 OPERATORS and OPERATOR OVERLOADING in C++

1. 运算符

什么是*operator*(运算符)？基本上就是一种我们用来代替函数执行某种事情的符号。所以这里说的不仅是数学运算符，也会有其它常用的运算符，比如前面提过的*dereference*(解引用)运算符，箭头运算符，`+=`运算符，取址运算符，左移运算符(`<<`)

还有一些其它的你可能根本就没把它当成运算符的东西，比如`new`和`delete`,它们实际也是运算符。

除此之外，逗号也可以是运算符，括号也可以是。

2. 运算符重载

*overload*重载这个术语本质就是给运算符重载赋予新的含义，或者添加参数，或者创建
允许在程序中国定义或更改运算符的行为。

不过说到底，运算符就是*function*，就是函数。
与其写出函数名`add`，你只用写一个`+`这样的运算符就行，在很多情况下这真的有助于让你的代码更干净整洁，可读性更好。

运算符重载的使用应该非常少，而且只是在完全有意义的情况下使用。

C++

```cpp
#include <iostream>

struct Vector2
{
    float x, y;

    Vector2(float x,float y)
        : x(x),y(y) {}

    Vector2 Add(const Vector2& other) const
    {
        return Vector2(x + other.x, y + other.y);
    }

    Vector2 Multiply(const Vector2& other) const
    {
        return  Vector2(x * other.x, y * other.y);
    }
};

int main()
{
    Vector2 position(4.0f, 4.0f);
    Vector2 speed(0.5f, 1.5f);
    Vector2 powerup(1.1f, 1.1f);

    Vector2 result = position.Add(speed.Multiply(powerup));

    std::cin.get();
}
```

这里`result`看起来有点难读，如果在Java这样的语言中这是你唯一的方法。不过再C++中我们有运算符重载，这意味着我们可以利用这些运算符，并定义我们自己的运算符。

```cpp
    Vector2 Add(const Vector2& other) const
    {
        return Vector2(x + other.x, y + other.y);
    }

    Vector2 operator+(const Vector2& other) const
    {
        return Add(other);
    }

    Vector2 Multiply(const Vector2& other) const
    {
        return  Vector2(x * other.x, y * other.y);
    }

    Vector2 operator*(const Vector2& other)const
    {
        return Multiply(other);
    }
};

int main()
{
    Vector2 position(4.0f, 4.0f);
    Vector2 speed(0.5f, 1.5f);
    Vector2 powerup(1.1f, 1.1f);

    Vector2 result = position + speed * powerup;

    std::cin.get();
}
```

3. 左移运算符

```cpp
std::cout <<
```

我们这里不能直接输出，因为这个运算符没有重载

```cpp
Vector2 result1 = position.Add((speed.Multiply(powerup)));
Vector2 result2 = position + speed * powerup;

std::cout << result2 << std::endl;
```

C++

```cpp
std::ostream& operator<<(std::ostream& stream,const Vector2& other)
{
    stream << other.x << "," << other.y << std::endl;
    return  stream;
}
```

C++

```cpp
struct Vector2
{
    float x, y;

    Vector2(float x,float y)
        : x(x),y(y) {}

    Vector2 Add(const Vector2& other) const
    {
        return Vector2(x + other.x, y + other.y);
    }

    Vector2 operator+(const Vector2& other) const
    {
        return Add(other);
    }

    Vector2 operator*(const Vector2& other)const
    {
        return Multiply(other);
    }

    Vector2 Multiply(const Vector2& other) const
    {
        return  Vector2(x * other.x, y * other.y);
    }

    bool operator==(const Vector2& other) const
    {
        return x == other.x && y == other.y;
    }

    bool operator!=(const Vector2& other) const
    {
        return !(*this == other);
    }
};

std::ostream& operator<<(std::ostream& stream,const Vector2& other)
{
    stream << other.x << "," << other.y << std::endl;
    return  stream;
}



int main()
{
    Vector2 position(4.0f, 4.0f);
    Vector2 speed(0.5f, 1.5f);
    Vector2 powerup(1.1f, 1.1f);

    Vector2 result1 = position.Add((speed.Multiply(powerup)));
```

```
    Vector2 result2 = position + speed * powerup;

    if(result1==result2)
        std::cout << result2 << std::endl; // 输出
    if(result1≠result2)
        std::cout << result2.x << "," << result2.y << std::endl; // 不输出
    std::cin.get();
}
```