运算符重载

定义：

对已有的运算符进行重新的定义，以适应不同的数据类型。

其实还是一个函数，只不过函数名改为了operator后面跟要重载的运算符。

参数：

运算符重载函数的参数个数取决于：

1. 运算符是一元运算符还是二元运算符
2. 运算符重载函数是全局函数还是成员函数。对于全局函数，一元运算符有一个参数，二元运算符有两个参数；对于成员函数，一元运算符没有参数，二元运算符有一个参数。

+号运算符重载代码：

#define \_CRT\_SECURE\_NO\_WARNINIGS

#include<iostream>

using namespace std;

class Person

{

public:

Person() {};

Person(int a, int b) : m\_A(a), m\_B(b)

{}

//+号重载 成员函数,将成员变量对应相加

Person operator+(Person& p)

{

Person tmp;

tmp.m\_A = this->m\_A + p.m\_A;

tmp.m\_B = this->m\_B + p.m\_B;

return tmp;

}

int m\_A;

int m\_B;

};

//全局函数，重载

//Person operator+ (Person & p1, Person & p2)

//{

// Person tmp;

// tmp.m\_A = p1.m\_A + p2.m\_A;

// tmp.m\_B = p1.m\_B + p2.m\_B;

// return tmp;

//}

void test()

{

Person p1(10, 10);

Person p2(20, 20);

Person p3 = p1 + p2; //成员函数，全局函数都可以被调用

cout << "p3.m\_A = " << p3.m\_A << endl;

cout << "p3.m\_B = " << p3.m\_B << endl;

}

int main()

{

test();

return 0;

}

左移运算符重载

首先，我们想要实现的需求是用左移运算符<<直接输出输出类，实现如下所示：  
cout << p

输出为

P的m\_A为… p的m\_B为…

要实现这种功能，就要对cout进行重载。之前说到，重载运算符分为成员函数重载和全局函数重载，但是左移运算符<<前面已经有了cout，因此成员函数重载是无法使用的，只能用全局函数重载。

同时，左移运算符还有连续输出的功能，这一点就要通过重载的全局函数返回cout引用来实现。

当需要输出的数据为类中的私有数据时，还需要将重载全局函数添加为类的友元函数。

代码如下：

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

using namespace std;

class Person

{

friend ostream& operator<<(ostream& cout, Person& p1);

public:

Person(){}

Person(int a, int b)

{

this->m\_A = a;

this->m\_B = b;

}

private:

int m\_A;

int m\_B;

};

//全局函数重载

ostream& operator<<(ostream& cout, Person& p1)

{

cout << "m\_A = " << p1.m\_A << "m\_B = " << p1.m\_B;

return cout;

}

void test()

{

Person p1(10, 10);

cout << p1 << endl;

}

int main()

{

test();

return 0;

}

前置后置++重载

如何区分前置++和后置++的重载函数？

用占位参数区分：

前置++： MyInterger& operator++(){}

后置++： MyInterger operator++(int){}

前置++ 返回的是类的引用，因为前置++，数据值已经+1了，返回的是已经+1的类，可以用引用，一遍后续操作。

后置++ 返回的是类，因为需要开辟临时变量存储未+1的值，返回的是临时变量，是不能用引用的。

代码：

#include<iostream>

Using namespace std;

class MyInteger

{

friend ostream& operator<<(ostream& cout, MyInteger);

public:

MyInteger()

{

m\_Num = 0;

}

//前置++重载

MyInteger& oeprator++()

{

this->m\_Num++;

return \*this;

}

//后置++重载

MyInteger operator++(int)

{

//临时变量保存

MyInteger tmp = \*this;

m\_Num++;

return tmp;

}

int m\_Num;

};

ostream& operator<<(ostream& cout, MyInteger& myInt)

{

cout << myInt.m\_Num;

return cout;

}