1. 运算符的重载
2. 运算符重载概述
3. 引例：a.geti()=a1.geti()+b1.geti();

//这样写并不方便，可以使用重载“+”来简化

1. 实现：void operator+(A a){

i=i+a.i;

}

--->解读：在**某个类中**重新定义+，则该类的对象可以作为+的操作数

调用时直接使用a1+a2，看上去没有赋值，

实质上是调用+函数改变了a1的值，即a1.operator+(a2);

1. 规则
   1. 至少含有一个自定义数据类型（即重载为全都是原有数据类型是不合法的）
   2. 不能违背运算符原来的语法规则
   3. 不能改变原有的优先级与结合性
   4. 不能自创运算符
   5. 重载的功能应当与原有功能相类似
   6. 不能改变操作数的个数
2. 作为成员函数与友元函数的重载

如 obj=obj+20; 可以重载为成员函数也可以是友元函数

obj=20+obj; 只可以重载为友元函数，因为无法将后者默认为调用方

1. 双目运算符的重载

（将operator+视作一个函数名）

1. 作为成员函数：只需要给出第二个操作数，因为第一个由隐含的this指针给出

表达为：obj1 + obj2;

解释为： obj1.operator+(obj2);

1. 作为非成员函数（友元函数）：需要给出两个操作对象（两个参数）

表达为：obj1 + obj2;

解释为：operator+(obj1 , obj2);

1. 区别

成员函数重载的运算符，第一操作数的类型是确定的，是隐含的，不可改变，即\*this

但是如果程序中需实现不同对象的相加，如：

obj = obj + 20; 或 obj = 20 + obj ;

则需要使用友元函数来重载

1. 单目运算符的重载
2. 前缀重载：operator++(); 无需参数

--> ++a; <=> a.operator++();

1. 后缀重载：operator++(int); 多一个第二参数，但仅用来区别前缀与后缀

--> a++; <=> a.operator++(0);

1. 返回值类型：a++的返回值是一个类型为a的引用，故而可以参与左值运算

即“a++=3;”是合法的

1. 重载的实现：

Point point::operator++(int)

{

Point temp=\*this; //先创建一个临时对象用于存储原来的值

++\*this; //直接调用之前定义的前缀自加

return temp;  
}

1. 实例应用

Ex.时钟类型的分与秒计时

time& time::operator++(){

sec++;//这两行可以一起写成“++sec>=60”

if(sec>=60){

sec-=60;

min++;

display();

}

return \*this;

};

1. 输出运算符<<的重载

//类的非成员运算符重载函数可以声明为类的友元函数

1. 应用场景：cout只能处理预定义的数据类型，当输出数据不是预定义类型时，就需要重载输出运算符“<<”
2. 实现：friend operator<<(ostream& o, point p)

{

o<<p.x<<p.y<<endl;

Return o;

}

//将o作为返回值可以实现连续输出（cout<<p<<p1）

解读为“cout<<p”是一个函数返回值，又作为后一次”<<”运算符的参数

//插入运算符的重载声明往往写在类外

1. 第一个参数为ostream，第二个参数为需要输出的参数类型
2. 当<<运算符重载且无返回值时，并不需要写明void，而是将其省略

---->>>（ 输入运算符>>同理 ）

1. 下标运算符[ ]的重载

实例：为my string 类做[]下标运算符重载进行下标超界检查

（声明）char & operator [ ] (int i);

（实现）My string :: operator[ ](int i){

If( i<length && i>=0 ) return buff[i];

}

//Explain：下标运算符重载的返回值往往为一个引用，这样可以进行左值运算

//attention：含有下标运算符的指针在析构时也需要加下标运算符[]才能释放完全

~Matrix(){delete []p};

1. 复制构造函数与赋值运算符函数重载
2. 区别：

复制构造函数：当前对象未创建，复制源对象已创建

赋值运算符函数：当前对象与复制源对象都已经创建

1. 实现：

Person& person:: operator = (person& source ){

If((this->name)!=0) //指针不为空

Delete this -> name;

This -> name = new char[size of(source.name)]; //重新分配

Strcpy(this->name, source.name);

Return \*this;

}

Attention：与前文同理，为了使得连续赋值能够进行，将返回值设置为指针

1. 类型转换运算符( )的重载

类型转换函数：将类类型转化为标准类型

Operator<标准类型名>( )

{

……

Return 标准类型名；

}