第 3 章　运算符重载

3.1　什么是运算符重载

为了实现两个 Time 类对象的加法运算，可以写出如下语句：

```
Time t1,t2;                    //定义时间类对象 t1、t2
t1= Tadd(t1,t2);               //调用函数 Tadd()计算两个时间的和
```

显然这种调用方式不直观、太烦琐，使人感到很不方便。能否也和整数的加法运算一样，直接用加号“＋”来实现时间的加法运算呢？可以把第二条语句替换为 t1＝t1＋t2；吗？答案是可以的，使用运算符重载就可以做到。

所谓重载，就是重新赋予新的含义。运算符重载是将系统中已有的运算符赋予不同的意义。使用运算符重载可以使 C++ 的代码更直观、更易懂、更灵活，使得用户自定义的数据类型以一种更方便、更简洁的方式工作。

所有预定义的运算符都是通过函数来实现的，所以也可以把运算符理解为特殊的函数，例如“＋”运算符就是由 operator＋()这个函数来实现的。其中，operator 是一个关键字，它与后面的运算符“＋”共同组成了该运算符函数的函数名。

由于运算符也是函数，所以在用户自定义的类中可以去重载这些函数。运算符重载的方法就是定义一个重载运算符的函数，在需要执行被重载的运算符时，系统就自动调用该函数，以实现相应的运算。也就是说，运算符重载是通过定义函数来实现的。运算符重载实质上是函数的重载。

运算符通常是对类中的私有成员进行操作，故重载运算符应能访问类中的私有成员，所以重载运算符一般采用成员函数或友元函数的形式。

3.2　重载运算符的规则

运算符重载的规则主要有以下几点：

(1) 重载运算符可以对运算符作出新的解释，但原有的基本语义不变。一个运算符被重载后，原有意义没有失去，只是定义了相对特定类的一个新运算符。

(2) 不改变运算符的优先级和结合性。

(3) 不改变运算符所需要的操作数，即单目运算符只能重载为单目运算符，不能将单目运算符重载为双目运算符。

(4) 不能创建新的运算符，只有系统预定义的运算符才能被重载。除“.”、“*”、“::”、“?”、sizeof 外，其他系统预定义的运算符都可以被重载。

(5) 重载运算符的函数不能有默认的参数，否则就改变了运算符参数的个数。

(6) 重载的运算符必须和用户自定义类型的对象一起使用，其参数至少应该有一个是

类对象或类对象的引用。

(7) 用于类对象的运算符一般必须重载,但有两个例外,运算符=和 &.,用户不必重载这两个运算符。

(8) 运算符重载是针对新类型的实际需求,对原有的运算符进行适当的改造。应当使重载运算符的功能类似于该运算符作用于标准类型数据时所实现的功能。

(9) 运算符重载函数可以是类的成员函数,也可以是类的友元函数。对于=、()、[]和->,运算符只能用成员函数的方式进行重载,对于<<和>>运算符必须用友元函数的方式进行重载。

3.3 运算符重载函数作为类的成员函数

可以将运算符重载函数作为类的成员函数,方法是在类中定义一个同名的运算符函数来重载该运算符。

定义运算符重载函数的格式如下:

```
函数类型 operator 运算符名称(形参表)
{
    函数体;
}
```

例如

```
Time operator+(time t)
```

其中 operator+为函数名,形参表中是运算符要求的操作数。在定义重载运算符的函数后,可以说函数 operator+重载了运算符+。

由于将运算符函数重载为类的成员函数,操作方就是当前对象,所以,如果重载单目运算符,就不必设置参数;如果重载双目运算符,就只要设置一个参数作为右侧操作数,而左侧操作数就是该对象本身。

【例 3-1】 对自定义的时间类 Time 实现"加法"操作。

```
#include<iostream.h>
class Time
{
    private:
    int hour,minute,second;                    //定义 3 个变量,分别代表小时、分钟和秒
    public:
    Time(int x=0,int y=0,int z=0)              //默认参数为 0 的构造函数
    {
        hour=x;minute=y;second=z;
    }
    Time operator+(Time &t);                   //声明重载的+运算符函数
    void show();
};
```

```
Time Time::operator+(Time &t)
{
    Time mt;
    int jinwei=0;
    //定义一个进位标志,让秒先相加,若大于 60 则进一位,依次类推
    mt.second=second+t.second;
    if(mt.second>=60)
    {
        mt.second-=60;
        jinwei=1;
    }
    mt.minute=minute+t.minute+jinwei;
    //分等于两个分相加再加上秒相加的进位
    jinwei=0;
    if(mt.minute>=60)
    {
        mt.minute-=60;
        jinwei=1;
    }
    mt.hour=hour+t.hour+jinwei;          //小时等于两个小时相加再加上分相加的进位
    return mt;
}
void Time::show()
{
    cout<<hour<<":"<<minute<<":"<<second;
}
int main()
{
    Time t1(10,10,10),t2(20,55,55),t3;
    t3=t1+t2;
    t1.show();
    cout<<"+";
    t2.show();
    cout<<"=";
    t3.show();
    cout<<endl;
    return 0;
}
```

程序的运行结果为：

```
10:10:10+20:55:55=31:6:5
```

在 VC++ 6.0 环境下，为了实现运算符重载，原来程序的第一行改为#include<iostream.h>，而且要删去第二行。

通过例 3-1 可以看出，只要将“+”运算符重载为 Time 类的成员函数，就可以直接使用

“+”运算符对 Time 类的两个对象进行加法运算，具体如何执行加法，由用户自己定义。

语句 t3＝t1＋t2；在编译时变成什么样子呢？由于已经将＋运算符重载为 Time 类的成员函数，这一语句首先编译成通过对象来调用类的成员函数的形式：t3＝t1. operator＋(t2)；，2.2.6 节介绍了 this 指针，在每一个成员函数中都包含一个 this 指针。这样在执行 t3＝t1. operator＋(t2)；语句时，C ++ 就把它处理为 t3＝t1. operator＋(&t1，t2)；，即给 operator＋()函数新增加了一个参数 &t1。

对于 Time::operator＋()函数，C++ 把它处理为如下形式：

```
Time Time::operator+(Time * this,Time &t)
{
    Time mt;
    int jinwei=0;
    mt.second=this->second+t.second;
    if(mt.second>=60)
    {
        mt.second-=60;
        jinwei=1;
    }
    mt.minute=this->minute+t.minute+jinwei;
    jinwei=0;
    if(mt.minute>=60)
    {
        mt.minute-=60;
        jinwei=1;
    }
    mt.hour=this->hour+t.hour+jinwei;
    return mt;
}
```

【例 3-2】 对自定义的时间类 Time 重载运算符“!”作为其成员函数，用于判断时间对象是否为 0。

```
#include<iostream.h>
class Time
{
    private:
    int hour,minute,second;                    //定义 3 个变量,分别代表小时、分钟和秒
    public:
    Time(int x=0,int y=0,int z=0)              //默认参数为 0 的构造函数
    {
    hour=x;minute=y;second=z;
    }
    int operator!();                           //声明重载的"!"运算符函数
};
int Time::operator!()
```

```
{
    if((hour==0)&&(minute==0) &&(second==0))  return 1;
    else  return 0;
}
int main()
{
    Time t1(10,10,10),t2;
    if(!t1)  cout<<"t1 is 0"<<endl;
    else     cout<<"t1 is not 0"<<endl;
    if(!t2)  cout<<"t2 is 0"<<endl;
    else     cout<<"t2 is not 0"<<endl;
    return 0;
}
```

程序的运行结果为：

```
t1 is not 0
t2 is 0
```

请读者自己分析一下！t1 编译时变成了什么样子，对于函数 Time::operator+()，C++ 把它处理成什么样子。

3.4 运算符重载函数作为类的友元函数

可以将运算符重载为当前类的友元函数，方法是定义一个与某一运算符函数同名的全局函数或另一个类的成员函数，并将该函数声明为当前类的友元函数。

在类的内部声明友元函数的格式如下：

```
friend 函数类型 operator 运算符(形参表);
```

在类的外部对该函数进行定义的格式如下：

```
函数类型 operator 运算符(形参表)
{
    函数体;
}
```

其中，“函数类型”表示该运算符函数返回的类型，“operator 运算符”是函数名。对于单目运算符，参数表中只包含一个参数；而对于双目运算符，参数表中要包含两个参数。

【例 3-3】 为了便于理解和对比，下面对例 3-1 进行修改，重载函数 operator+()不作为 Time 类的成员函数，而是把重载函数放在类外，作为 Time 类的友元函数。

```
#include<iostream.h>
class Time
{
    private:
    int hour,minute,second;                    //定义 3 个变量,分别代表小时、分钟和秒
```

```
    public:
    Time(int x=0,int y=0,int z=0)              //默认参数为 0 的构造函数
    {
        hour=x;minute=y;second=z;
    }
    friend Time operator+(Time &t1,Time &t2);
    //重载"+"运算符函数作为友元函数
    void show();
};
void Time::show()
{
    cout<<hour<<":"<<minute<<":"<<second;
}
int main()
{
    Time t1(10,10,10),t2(20,55,55),t3;
    t3=t1+t2;
    t1.show();
    cout<<"+";
    t2.show();
    cout<<"=";
    t3.show();
    cout<<endl;
    return 0;
}
Time operator+(Time &t1,Time &t2)
{
    Time mt;
    int jinwei=0;
    //定义一个进位标志,让秒先相加,若大于 60 则进一位,依次类推
    mt.second=t1.second+t2.second;
    if(mt.second>=60)
    {
        mt.second-=60;
        jinwei=1;
    }
    mt.minute=t1.minute+t2.minute+jinwei;
    //分等于两个分相加再加上秒相加的进位
    jinwei=0;
    if(mt.minute>=60)
    {
        mt.minute-=60;
        jinwei=1;
    }
    mt.hour=t1.hour+t2.hour+jinwei;
```

```
    //小时等于两个小时相加再加上分相加的进位
    return mt;
}
```

程序的运行结果为：

```
10:10:10+20:55:55=31:6:5
```

由于已经将＋运算符重载为 Time 类的友元函数，因此语句 t3＝t1＋t2；编译成为调用普通函数的形式：t3＝operator＋(t1,t2)；。

运算符重载函数可以是类的成员函数，也可以是类的友元函数，请读者思考二者有何区别，在参数个数和调用方式上有什么不同。

把 Time 类的定义及主函数修改如下：

```
class Time
{
    private:
    int hour,minute,second;
    public:
    Time() {hour=0;minute=0;second=0;}
    Time(int x,int y,int z) {hour=x;minute=y;second=z;}
    Time(int z) {hour=0;minute=0;second=z;}
    friend Time operator+(Time &t1,Time &t2);
};
int main()
{
    Time t1(10,10,10),t2;
    t1=t1+45;
    t2=45+t1;
    return 0;
}
```

编译系统怎样处理语句 t2＝t1＋45；呢？在编译时，系统发现运算符左侧的 t1 是 Time 类对象，右侧的 45 是一个整数。那么编译系统首先寻找有没有对＋运算符的重载，发现有 operator＋()函数，它是 Time 类的友员函数，要求两个 Time 类的形参，而现在 45 是一个整数，不符合要求。然后编译系统就去找有没有转换构造函数，发现有 Time(int z)这个转换构造函数，于是去调用 Time(int z) 这个转换构造函数，将 45 转换为 Time 类常量后，才去调用 operator＋()函数。该过程相当于执行语句 t2＝t1＋Time(45)；。同理，语句 t2＝45＋t1；相当于执行语句 t2＝Time(45)＋t1；。本段代码使用的 codeblocks 和 compiler 是 mingw32－g＋＋.exe 可以编译通过的，但 VC 不支持。

如果主函数不变，修改 Time 类的定义如下：

```
class Time
{
    private:
    int hour,minute,second;
```

```
    public:
    Time() {hour=0;minute=0;second=0;}
    Time(int x,int y,int z) {hour=x;minute=y;second=z;}
    Time(int z) {hour=0;minute=0;second=z;}
    Time operator+(Time &t);
};
```

编译系统怎样处理语句 t2＝t1＋45;呢？在编译时,编译系统首先寻找有没有对“＋”运算符的重载,发现有 operator＋()函数,它是 Time 类的成员函数,要求一个 Time 类的形参,而现在 45 是一个整数,不符合要求。然后编译系统就去找有没有转换构造函数,发现有 Time(int z)这个转换构造函数,于是将该语句编译为 t2＝t1.operator＋(45);,去调用 Time(int z) 这个转换构造函数,将 45 转换为 Time 类常量后,才去调用 t1.operator＋()函数。相当于执行语句 t2＝t1.operator＋(Time(45));。而语句 t2＝45＋t1;会编译为 t2＝45.operator＋(t1),由于 45 不是 Time 类对象不能调用 Time 类的成员函数,所以编译会出错。

成员函数重载的＋运算符不支持交换律,从这个例子中可以看出在第一个参数需要隐式转换的情形下,使用友员函数重载运算符是正确的选择。

重载单目运算符的方法与重载双目运算符的方法是类似的。但由于单目运算符只有一个操作数,因此运算符重载为友元函数时只有一个参数。

【例 3-4】 对自定义的时间类 Time 重载运算符“!”作为其友元函数,用于判断时间对象是否为 0。

```
#include<iostream.h>
class Time
{
    private:
    int hour,minute,second;                  //定义 3 个变量,分别代表小时、分钟和秒
    public:
    Time(int x=0,int y=0,int z=0)            //默认参数为 0 的构造函数
    {
        hour=x;minute=y;second=z;
    }
    friend int operator!(Time &t);           //声明重载的"!"运算符函数
};
int operator!(Time &t)
{
    if((t.hour==0)&&(t.minute==0) &&(t.second==0))  return 1;
    else  return 0;
}
int main()
{
    Time t1(10,10,10),t2;
    if(!t1)  cout<<"t1 is 0"<<endl;
    else     cout<<"t1 is not 0"<<endl;
```

```
    if(!t2)  cout<<"t2 is 0"<<endl;
    else     cout<<"t2 is not 0"<<endl;
    return 0;
}
```

程序的运行结果为：

```
t1 is not 0
t2 is 0
```

什么时候应该用成员函数重载，什么时候应该用友元函数重载，这需要视情况而定。由于友元的使用会破坏类的封装，因此从原则上说，要尽量将运算符函数作为类的成员函数。但考虑到各方面的因素，一般将单目运算符重载为成员函数，将双目运算符重载为友元函数。另外，C++ 规定，有的运算符(=、()、[]、->)必须定义为类的成员函数，有的运算符则不能定义为类的成员函数(如流插入运算符<<和流提取运算符>>、类型转换运算符)。若一个运算符的操作需要修改类对象的状态时(如=、*=和++)，应该用成员函数重载；如果运算符的左右操作数类型不同，左操作数可以是常数或其他类型的数时，希望有隐式转换，则必须用友元函数重载。

3.5 重载++和--运算符

++和--运算符属于单目运算符，但是它们有两种使用方式，即前置方式++x 和后置方式 x++，其意义是不相同的。前置方式(如++x)是先自加，返回的是修改后的对象本身。后置方式(如 x++)返回的是自加前的对象，然后对象自加。C++ 对此做了约定：在自增(自减)运算符重载函数中，增加(减少)一个 int 型形参，就是后置自增(自减)运算符函数，否则就是前置方式。

下面以自增运算符++为例说明上述两种使用方式。

(1) 前置方式++x：

```
成员函数重载：类类型 operator++()
{
    函数体;
}
友元函数重载：friend 类类型 operator++(类类型)
{
    函数体;
}
```

(2) 后置方式 x++：

```
成员函数重载：类类型 operator++(int)
{
    函数体;
}
友元函数重载：friend 类类型 operator++(类类型,int)
```

```
{
    函数体;
}
```

说明：int 在这里只是一个占位符，用来区分该函数是前置还是后置方式，并没有实际意义。显示调用时，该占位符通常用 0 表示。

【例 3-5】 对 Time 类进行自增运算，每次走一秒，为了简化代码，假设当前的秒数小于 59。要求使用多种方法实现。

方法一：重载前置和后置＋＋运算符函数作为成员函数。

```
#include<iostream.h>
class Time
{
    private:
    int hour,minute,second;                          //定义 3 个变量,分别代表小时、分钟和秒
    public:
    Time(int x=0,int y=0,int z=0)  {hour=x;minute=y;second=z;}
    Time operator++()                                //重载前置++运算符函数作为成员函数
    {second++;return * this;}
    Time operator++(int x)                           //重载后置++运算符函数作为成员函数
    {Time t;t= * this;  second++;  return t;}
    void show()
    {cout<<hour<<":"<<minute<<":"<<second<<endl;}
};
int main()
{
    Time t1(10,10,10),t2,t3;
    t1.show();
    t2=++t1;                                         //调用重载前置++运算符函数
    t1.show();
    t2.show();
    t3=t1++;                                         //调用重载后置++运算符函数
    t1.show();
    t3.show();
}
```

程序的运行结果为：

```
10:10:10    (t1 原值)
10:10:11    (执行 t2=++t1;后 t1 的值)
10:10:11    (执行 t2=++t1;后 t2 的值)
10:10:12    (执行 t3=t1++;后 t1 的值)
10:10:11    (执行 t3=t1++;后 t3 的值)
```

方法二：重载前置和后置＋＋运算符函数作为友元函数。

```
#include<iostream.h>
```