第七章 运算符重载

模块7.1：基本概念

主讲教师：同济大学计算机科学与技术学院 陈宇飞
同济大学计算机科学与技术学院 龚晓亮

目录

- 运算符重载的方法
- 运算符重载的规则
- 成员函数和友元函数

1.1 运算符重载的方法

- 引例：今天早上在某账户上花费了2小时35分钟，下午又花费了2小时40分钟，则总共花了多少时间呢？

➢与加法概念吻合，但要相加的单位（小时与分钟的混合）与内置类型不匹配

解决方法0：在Time类中使用方法Sum()来处理加法

```
//mytime0.h -- Time class before
               operator overloading
#ifndef MYTIME0_H_
#define MYTIME0_H_
class Time {
private:
    int hours;
    int minutes;
public:
    Time();
    Time(int h, int m = 0);
    void AddMin(int m);
    void AddHr(int h);
    void Reset(int h = 0, int m = 0);
    Time Sum(const Time& t)const;
    void Show()const;
};
#endif
```

```
//mytime0.cpp -- implementing Time methods
#include<iostream>
#include"mytime0.h"
Time::Time() {…}
Time::Time(int h, int m) {…}
void Time::AddMin(int m) {…}
void Time::AddHr(int h) {…}
void Time::Reset(int h, int m) {…}
Time Time::Sum(const Time& t)const { //具体实现
   Time sum;
   sum.minutes = minutes + t.minutes;
   sum.hours = hours + t.hours + sum.minutes / 60;
   sum.minutes %= 60;
   return sum;
}
void Time::Show()const{…}
```

1.1 运算符重载的方法

- 解决方法0：在Time类中使用方法Sum()来处理加法

```
Time Time::Sum(const Time& t)const { //具体实现
  Time sum;
  sum.minutes = minutes + t.minutes;
  sum.hours = hours + t.hours + sum.minutes / 60;
  sum.minutes %= 60;
  return sum;
}
```

➢参数是引用：可以按值传递Time对象，但传递引用速度更快，占用内存更少

➢返回值不能是引用：返回对象sum将创建对象的副本，函数结束删除sum之前构造它的拷贝，供调用函数使用此拷贝。如果返回Time &，引用的是局部变量sum对象，函数结束时将被删除，引用将指向不存在的对象

```
//mytime0.h -- Time class before operator overloading
class Time {
    …
    Time Sum(const Time& t)const;
};
```

头文件
类的声明

```
//mytime0.cpp -- implementing Time methods
#include"mytime0.h"
Time Time::Sum(const Time& t)const {…}
…
```

源程序文件
函数的实现

```
//usetime0.cpp -- using the first draft of the Time class
//compile usetime0.cpp and mytime0.cpp together
#include"mytime0.h"
int main()
{
    Time coding(2, 40);
    Time fixing(5, 55);
    Time total;
    total = coding.Sum(fixing);  //想用更直观的+形式怎么办?
    ...
}
```

源程序文件
调用函数

1.1 运算符重载的方法

- 引例：今天早上在某账户上花费了2小时35分钟，下午又花费了2小时40分钟，则总共花了多少时间呢？

➢与加法概念吻合，但要相加的单位（小时与分钟的混合）与内置类型不匹配

解决方法0：在Time类中使用方法Sum()来处理加法

解决方法1：添加加法运算符来处理加法

```
//mytime1.h -- Time class before
               operator overloading
#ifndef MYTIME1_H_
#define MYTIME1_H_
class Time {
private:
    int hours;
    int minutes;
public:
    Time();
    Time(int h, int m = 0);
    void AddMin(int m);
    void AddHr(int h);
    void Reset(int h = 0, int m = 0);
    Time operator+(const Time& t)const;
    void Show()const;
};
#endif
```

```
//mytime1.cpp -- implementing Time methods
#include<iostream>
#include"mytime1.h"
Time::Time() {…}
Time::Time(int h, int m) {…}
void Time::AddMin(int m) {…}
void Time::AddHr(int h) {…}
void Time::Reset(int h, int m) {…}
Time Time::operator+ (const Time& t)const {
   Time sum;
   sum.minutes = minutes + t.minutes;
   sum.hours = hours + t.hours + sum.minutes / 60;
   sum.minutes %= 60;
   return sum;
}
void Time::Show()const{…}
```

1.1 运算符重载的方法

- 解决方法1：添加加法运算符来处理加法

```
Time Time::operator+ (const Time& t)const { //具体实现
  Time sum;
  sum.minutes = minutes + t.minutes;
  sum.hours = hours + t.hours + sum.minutes / 60;
  sum.minutes %= 60;
  return sum;
}
```

➢与sum()一样：参数是引用，返回值是对象

➢operator+()由Time对象调用，第二个Time对象作为参数，并返回一个Time对象

```
total = coding.operator+(fixing);  //function notation
total = coding + fixing;           //operator notation
```

```
//mytime1.h -- Time class before operator overloading
class Time {
    …
    Time operator+(const Time& t)const;
};
```

头文件
类的声明

```
//mytime1.cpp -- implementing Time methods
#include"mytime1.h"
Time Time::operator+ (const Time& t)const {…}
…
```

源程序文件
函数的实现

```
//usetime1.cpp -- using the second draft of the Time class
//compile usetime1.cpp and mytime1.cpp together
#include"mytime1.h"
int main()
{   Time coding(2, 40);
    Time fixing(5, 55);
    Time morefixing(3, 28);
    Time total;
    total = coding + fixing;  //operator notation
    total = morefixing.operator+(total);  //function notation
    ...
}
```

源程序文件
调用函数

1.1 运算符重载的方法

- 运算符重载的形式：

```
返回类型 operator 运算符(形参表)
{
    重载函数实现
}
```

- 用operator 运算符来表示对应运算符的函数

operator + ⇔ +

operator * ⇔ *

1.1 运算符重载的方法

- 对象 运算符 另一个值(可以不是对象、可以无)转换为函数调用：

  对象.operator运算符(另一个值)

```
Time t1, t2, t3, t4;
t4 = t1 + t2 + t3;
```

➡ `t4 = t1.operator+(t2 + t3);`

➡ `t4 = t1.operator+(t2.operator+(t3));`

返回Time对象，值t2+t3

返回Time对象，值t1+t2+t3

1.1 运算符重载的方法

- 运算符被重载后，原来用于其它数据类型上的功能仍然被保留(重载)，系统根据重载函数的规则匹配

```
int a, b, c;
Time A, B, C;
c = a + b;          //use int addition
C = A + B;          //use addition defined for Time objects
```

目录

- 运算符重载的方法
- 运算符重载的规则
- 成员函数和友元函数

2.1 运算符重载的规则

1. 重载运算符的两侧至少有一个是类对象（防止用户为标准类型重载运算符）
2. 不能违反运算符原来的句法规则
   - 不能改变操作对象的个数（例：不能将求模%重载成使用一个操作数）
   - 不能改变优先级
   - 不能改变结合性
3. 不能创建新运算符（例：不能定义operator **()函数来表示求幂）
4. 不能重载的运算符（primer书：sizeof　::　?:　.　.*　等）
5. 可重载的运算符（primer书 表11.1）
   - 只能通过成员函数进行重载：

     =：赋值运算符　　()：函数调用运算符

     []：下标运算符　　->：通过指针访问类成员的运算符

2.1 运算符重载的规则

6. 不允许带默认参数（操作数是不可缺少的）
7. 应当使重载运算符的功能与标准相同/相似(建议)
8. =和&系统缺省做了重载，=是对应内存拷贝，&取地址

例：关于=赋值（回顾）

```
test t1("hello"), t2;
t2=t1;    //整体内存拷贝
```

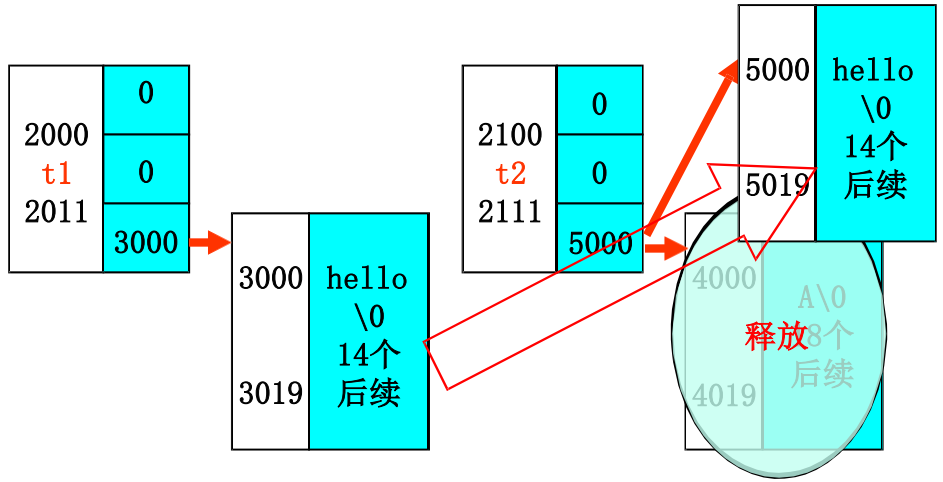
- 若对象数据成员是指针及动态分配的数据，则可能导致不可预料的后果甚至报错
- 解决方法：重载运算符=

对象的赋值与复制 （回顾）

- 对象的赋值
  - 有动态内存申请

```
int main()
{   test t1("hello"), t2;
    t1.display();   hello
    t2.display();   A
    t2=t1;
    t2.display();   hello
    t1.set("china");
    t1.display();   china
    t2.display();   hello
}
```

```
//解决方法：运算符重载！！
test &test::operator=(const test &t)
{   a = t.a;   b = t.b;
    delete c;                  //释放原空间
    c=new char[20];            //申请新空间
    strcpy(c, t.c);
    return *this;              //返回对象自身
}
```



2.1 运算符重载的规则

思考右侧运算符重载程序：

```
……//略
test &test::operator=(const test &t)
{  a = t.a;    b = t.b;
   delete c; //释放原空间
   c=new char[20]; //申请新空间
   strcpy(c, t.c);
   return *this; //返回对象自身
}
```

1）返回类型能否为test，而不是引用？

- 分析：返回值应符合运算符的语义

  t2 = t1理解为t2.operator=(t1)

  = 的语义希望执行后t2被改变

  若返回test，则返回时会调用复制构造函数，返回的就是临时对象而不是t2自身

- 对比 Time类+: Time operator+(const Time& t)const;

  coding + fixing理解为coding.operator+(fixing)

  + 的语义不能改变coding，应该返回临时对象，所以返回值是Time而不是Time&

2.1 运算符重载的规则

思考右侧运算符重载程序：

```
……//略
test &test::operator=(const test &t)
{  a = t.a;    b = t.b;
   delete c; //释放原空间
   c=new char[20]; //申请新空间
   strcpy(c, t.c);
   return *this; //返回对象自身
}
```

2）返回类型能否为void?

- 分析：返回值应符合运算符的语义

  t2 = t1理解为t2.operator=(t1)

  = 的语义希望执行后t2被改变

  若返回void，this指针指向t2，对本题而言正确

- 进一步思考：t3 = (t2 = t1);

  t2 = t1：赋值表达式的值等于左值

  t3 = (t2 = t1)：没有找到接受void类型的右操作数的运算符（或没有可接受的转换），连续赋值时程序错误

2.1 运算符重载的规则

思考右侧运算符重载程序：

```
……//略
test &test::operator=(const test &t)
{  a = t.a;    b = t.b;
   delete c; //释放原空间
   c=new char[20]; //申请新空间
   strcpy(c, t.c);
   return *this; //返回对象自身
}
```

3）为什么要先释放再申请新空间？

- 分析：

  申请/释放都是20字节，内存不保证相同

  对本题而言可以仍旧使用原来已申请的空间，可删除这两条语句

- 进一步思考：若要求test类按需申请，不浪费空间

  必须释放原空间再申请新空间

```
delete c;
c=new char[strlen(t.c)+1];
```

2.1 运算符重载的规则

- 复制构造函数和重载=的区别：

	复制构造函数	重载赋值运算符
系统缺省	有，对应内存拷贝	有，对应内存拷贝
必须定义的时机	含动态内存申请时	含动态内存申请时
调用时机	定义时用对象初始化 函数形参为对象，返回值为对象	执行语句中的=操作
调用时处理	对象生成时调用，此时不可能调用其它形式的构造函数	=操作时调用，在=前对象已生成，即已调用过某种形式的构造函数(包括复制构造函数)

//复制构造函数和重载=的区别

```
class test {
 ...
 public:
  test(const test &t);
  test &operator=(const test &t);
 };
test::test(const test &s)
{    a=s.a;
     b=s.b;
     c=new char[20];  //申请新空间
     strcpy(c, s.c);
}
```

```
test &test::operator=(const test &t)
{  a = t.a;
   b = t.b;
   delete c; //释放原空间（复制构造函数不需要）
   c=new char[20]; //申请新空间
   strcpy(c, t.c);
   return *this;//返回对象自身（复制构造函数不需要）
}
int main()
{   test t1("hello"),t2(t1); //复制构造函数
    t2 = t1; //=运算
}
```

目录

- 运算符重载的方法
- 运算符重载的规则
- 成员函数和友元函数

3.1 成员函数和友元函数

- 回顾引例：今天早上在某账户上花费了2小时35分钟，下午又花费了2小时40分钟，则总共花了多少时间呢？

➢与加法概念吻合，但要相加的单位（小时与分钟的混合）与内置类型不匹配

解决方法0：在Time类中使用方法Sum()来处理加法

解决方法1：添加加法运算符来处理加法（成员函数）

➢其它重载运算符（成员函数）：

```
Time operator-(const Time& t)const;
Time operator*(double n)const;
```

//具体实现详见primer书：mytime2.h　mytime2.cpp　usetime2.cpp

➢进一步思考：乘法运算符重载的合理性

3.1 成员函数和友元函数

➢进一步思考：乘法运算符重载的合理性

```
Time operator*(double n)const;
A = B * 2.75;  //合理，左操作数是对象
A = 2.75 * B;  //不合理
```

➢上述二者从概念上应该相同

解决方法0：非成员函数

```
Time operator*(double m, const Time & t);  //无法访问类的私有数据
```

解决方法1：友元函数（非成员函数）

```
friend Time operator*(double m, const Time & t);
```

3.1 成员函数和友元函数

- 成员函数与友元函数的区别（结合后续“案例应用”理解）

	成员函数	友元函数
单目运算符	空参数	一个参数(必须是对象)
双目运算符	一个参数 (可不是对象)	两个参数 (一个必须是对象,一个可不是)

- 两个操作数都是对象：没区别
- 一个操作数是对象：若希望 2.75 * B正确，则需要重载实现 double * Time，且该方式只能通过友元函数实现
- 建议对单目运算符采用成员函数方式，双目运算符采用友元函数方式
- C++规定，某些运算符必须是成员函数形式(赋值=，下标[]，函数())，某些运算符必须是友元函数形式(流插入<<，流提取>>，类型转换　类型(值))，可能因编译系统不同而不同

总结

- 运算符重载的方法（掌握）
- 运算符重载的规则（熟悉）
- 成员函数和友元函数（熟悉）