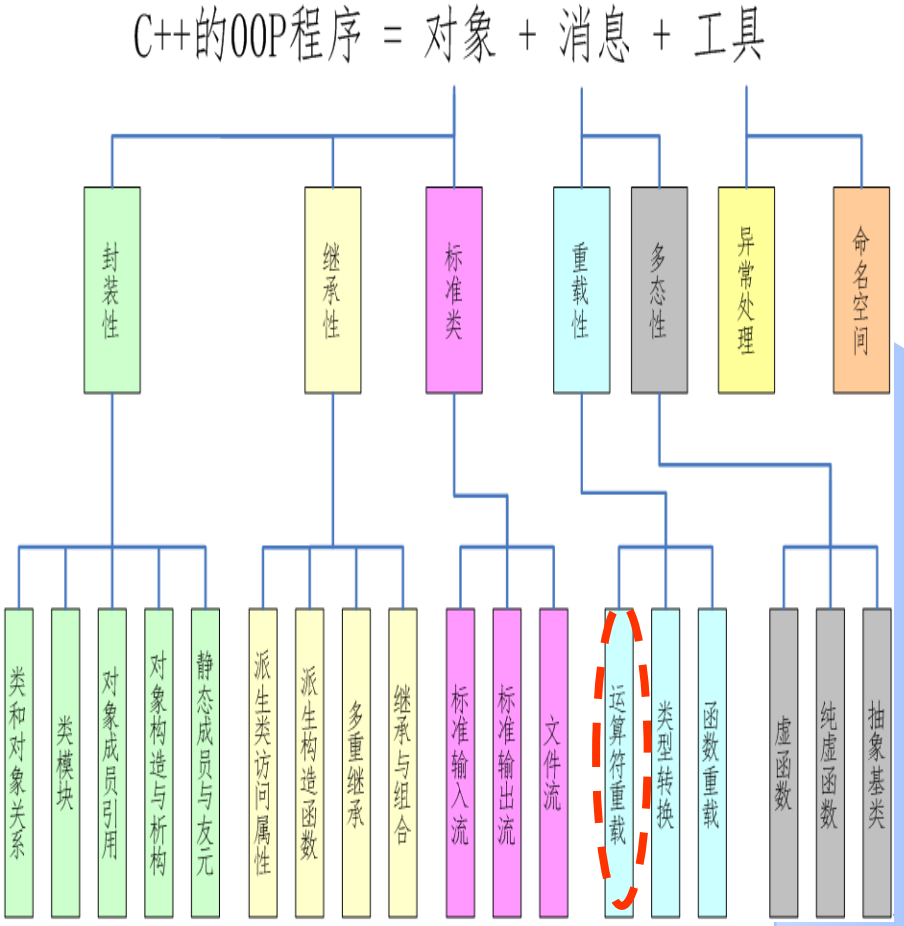
Lab of
New Generation Network
Techonology & Applications

# 第4讲　运算符重载

4.1 运算符重载概念

4.2 重载运算符的规则

4.3 友元及友元重载函数



# 引言

- 下列哪些表达式是正确的？为什么
  - int a,b,c; c=a+b;
  - char c1,c2,c3;c3=c1+c2;
  - char c1[4]=“abc”,c2[4]=“cde”,c3[10];
    c3=c1+c2;
    strcpy(c3,c1);strcat(c3,c2);
  - string s1=“abc”,s2=“cde”,s3; s3=s1+s2;?
  - struct student st1,st2,st3; st3=st1+st2;?
  - class student st1,st2,st3; st3=st1+st2;?

- 结论:在C++中的运算符只能对基本类型的数据进行操作；

  这些运算符本质上是由编译器提供的底层函数来实现

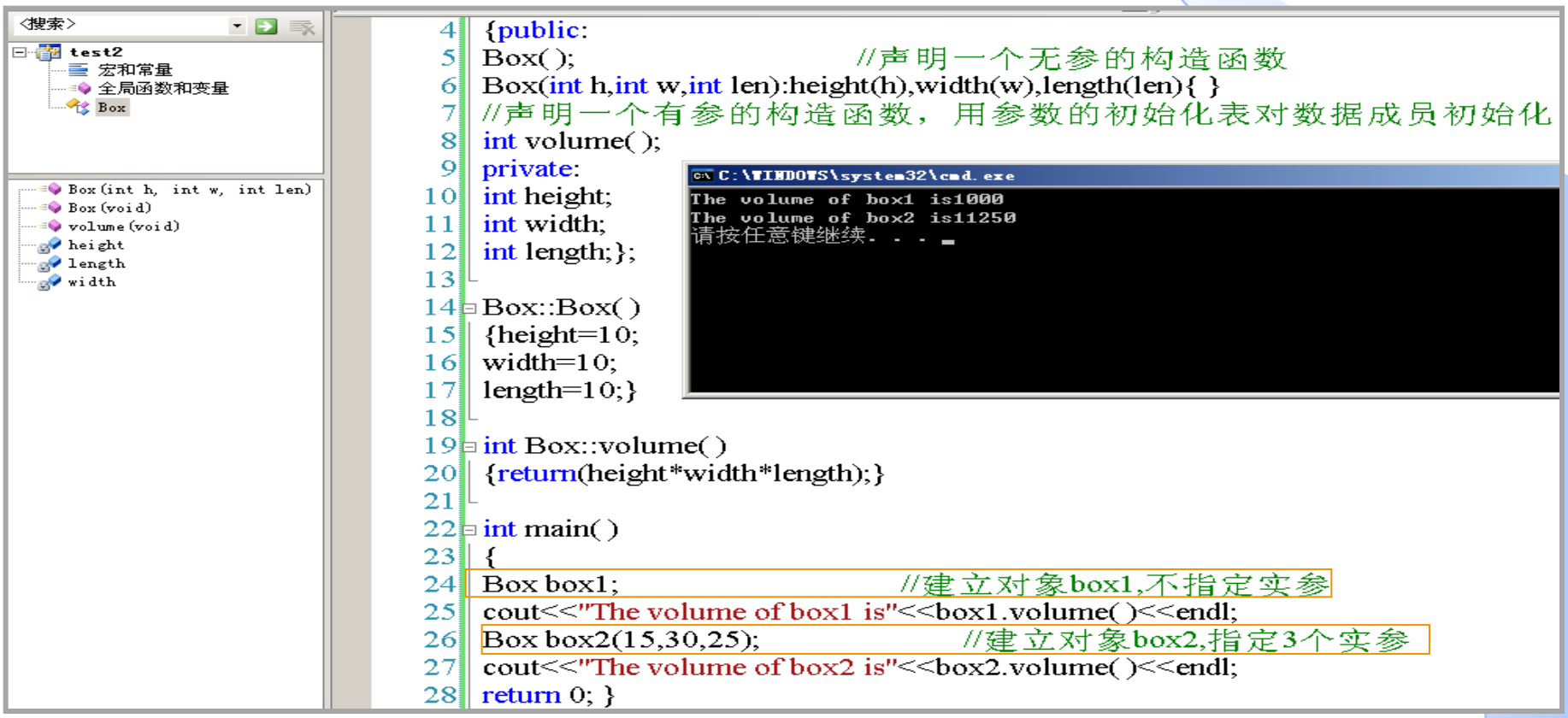
- 问题:对象是否可以使用这些运算符来操作呢？

# 4.1 运算符重载概念

- 重载概念：用户根据自己需要对C++已有运算符或函数的功能重新赋予新的含义，使之“一词多义”；

✓函数重载：同一函数名可以实现不同的功能；

✓运算符重载：同一运算符能实现不同类型数据操作；如box1+box2



# 生活中的重载概念



杯具一词重载

## 小故事--中文词汇“重载”的威力

某天吃饭时一人说去方便一下，日本人不解。旁人告诉他“方便”就是“上厕所”。敬酒时，一人对日本人说:希望下次去日本时能给予方便，日本人纳闷不敢问。酒席上，又电视台美女主持人提出，在她方便时会安排日本人做专访谈谈如啥要霸占钓鱼岛？日本人愕然：怎么能在你方便时谈呢？美女主持人说，那在你方便时，请你谈。日本人晕倒！醒来后，美女主持人又对他说，要不你我都方便时，再谈。又一次晕倒，再没有醒来。大家方便时笑一笑吧 。

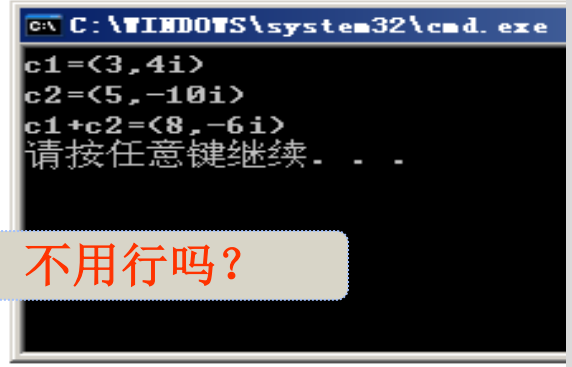
# 4.1 运算符重载概念

```
1  #include <iostream>
2  using namespace std;
3  class Complex                          //定义Complex类
4  {public:
5  Complex( ){real=0;imag=0;}             //定义构造函数
6  Complex(double r,double i){real=r;imag=i;}    //构造函数重载
7  Complex complex_add(Complex &c2);      //声明复数相加函数
8  void display( );                       //声明输出函数
9   private:
10 double real;                           //实部
11 double imag; };                        //虚部
12
13 Complex Complex::complex_add(Complex &c2)
14 {Complex c;
15 c.real=real+c2.real;
16 c.imag=imag+c2.imag;
17 return c;}
18
19 void Complex::display( )               //定义输出函数
20 {cout<<"("<<real<<","<<imag<<"i)"<<endl;}
21
22 int main( )
23 {Complex c1(3,4),c2(5,-10),c3;         //定义3个复数对象
24 c3=c1.complex_add(c2);                 //调用复数相加函数
25 cout<<"c1="; c1.display( );            //输出c1的值
26 cout<<"c2="; c2.display( );            //输出c2的值
27 cout<<"c1+c2="; c3.display( );         //输出c3的值
28 return 0;}
```

c.real=this->real+c2.real

c.real=c1.real+c2.real

为什么用对象引用，不用行吗？



# 4.1 运算符重载概念

- 运算符重载:通过定义一个函数来赋予一个运算符新功能。

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；

- 重载运算符一般格式：

  函数类型 operator 运算符名称（形参表列）

  {对运算符重载处理的函数体}

- 例：将“+”用于Complex类加法运算，函数原型：

  Complex operator+ (Complex& c1,Complex& c2);

```
1  #include <iostream>
2  using namespace std;
3  class Complex
4  {public:
5  Complex( ){real=0;imag=0;}
6  Complex(double r,double i){real=r;imag=i;}
7  Complex operator+(Complex &c2); //声明重载运算符的函数
8  void display( );
9   private:
10 double real;
11 double imag;
12 };
13 Complex Complex::operator+(Complex &c2)
14 { Complex c;
15 c.real=real+c2.real;
16 c.imag=imag+c2.imag;
17 return c;}
18
19 void Complex::display( )
20 { cout<<"("<<real<<","<<imag<<"i)"<<endl;}
21
22 int main( )
23 { Complex c1(3,4),c2(5,-10),c3;
24 c3=c1+c2;                    //运算符+用于复数运算
25 cout<<"c1=";c1.display( );
26 cout<<"c2=";c2.display( );
27 cout<<"c1+c2=";c3.display( );
28 return 0;
29 }
```

//定义重载运算符的函数

自动窗口

| 名称 | 值 | 类型 |
|---|---|---|
| c1 | {real=3.0000000000000000 imag=4.00( | Complex |
| imag | 4.0000000000000000 | double |
| real | 3.0000000000000000 | double |
| c2 | {real=5.0000000000000000 imag=-10.0 | Complex |
| imag | -10.000000000000000 | double |
| real | 5.0000000000000000 | double |
| c3 | {real=8.0000000000000000 imag=-6.00 | Complex |
| imag | -6.0000000000000000 | double |
| real | 8.0000000000000000 | double |

自动窗口 | 局部变量 | 线程 | 模块 | 监视 1

# 4.1 运算符重载概念

小结：

1） 例2中的operator+函数取代了例1中的complex_add函数，只是函数名不同，函数体和函数返回值类型都是相同；

2） main函数中，以“c3=c1+c2;”取代了例1中的“c3=c1.complex_add(c2);” 编译系统将程序中式c1+c2解释为c1.operator+(c2)

3）重载意义：c++提供的运算符只能用于标准数据类型的运算。但可通过重载，用于新的数据类型（类对象）

4）运算符被重载后，其原有的功能仍然保留。根据运算对象来确定是使用原有还是重载功能。

# 4.2运算符重载规则

## (1) C++允许重载的运算符

表 10.1 C++允许重载的运算符

| | |
|---|---|
| 双目算术运算符 | +(加),-(减),*(乘),/(除),%(取模) |
| 关系运算符 | ==(等于),!=(不等于),<(小于),>(大于),<=(小于等于),>=(大于等于) |
| 逻辑运算符 | \|\|(逻辑或),&&(逻辑与),!(逻辑非) |
| 单目运算符 | +(正),-(负),*(指针),&(取地址) |
| 自增自减运算符 | ++(自增),--(自减) |
| 位运算符 | \|(按位或),&(按位与),~(按位取反),^(按位异或),<<(左移),>>(右移) |
| 赋值运算符 | =,+=,-=,*=,/=,%=,&=,\|=,^=,<<=,>>= |
| 空间申请与释放 | new,delete,new[],delete[] |
| 其他运算符 | ()(函数调用),->(成员访问),->*(成员指针访问),,(逗号),[](下标) |

## (2) 不能重载的运算符

. (成员访问运算符)

* (成员指针访问运算符)

:: (域运算符)

sizeof(长度运算符)

?: (条件运算符)

- 对于双目运算符：

| 运算符类别 | 运算符及其原功能 |
| --- | --- |
| 双目运算符 | +(加)，-(减)，*(乘)，/(除)，%(取模) |
| 关系运算符 | ==(等于)，!=(不等于)，<(小于)，>(大于)，<br><=(小于等于)，>=(大于等于) |
| 位运算符 | \|(按位或)，&(按位与)，~(按位取反)，^(按位异或) |
| 赋值运算符 | =，+=，-=，*=，/=，%=，&=，\|= |
| 其它运算符 | ,(逗号) |

- 运算符函数声明为成员函数：

参数和返回值可以使用引用和const保护

```
函数类型 operator 运算符@ (类型名);
```

```
Complex & operator + (Complex & c2); //Complex + Complex
Complex & operator * (float c2);      //Complex + float
Complex & operator = (Complex & c2); //Complex = Complex
Complex & operator > (Complex & c2); //Complex > Complex
……
```

```
1.  class Complex {
2.      double re, im;
3.  public:
4.      Complex(double r=0, double i=0) { re=r; im=i; }

5.      Complex operator+(const Complex& C2)
6.      { return Complex(re+C2.re, im+C2.im); }

7.      Complex operator+(double x)
8.      { return Complex(re+x, im); }
9.      ...
10. };

11. void main()
12. {   Complex c1(3,4), c2(5,10), c3;
13.     c3 = c1 + c2;
14.     c3 = c1 + 2.5;
15.     c3 = c1 + 1;
16.     c3 = 1 + c1;    // 错误,没有相匹配的运算符函数
17. }
```

# 4. 2运算符重载规则

- 对于普通的单目运算符：

| 运算符类别 | 运算符及其原功能 |
| --- | --- |
| 逻辑运算符 | !(逻辑非) |
| 单目运算符 | +(正)，-(负)，*(指针取内容)，&(取地址) |
| 自增自减运算符 | ++(前置自增)，--(前置自减) |
| 其它运算符 | ()(函数调用) |

- 运算符函数声明为成员函数：

```
函数类型 operator 运算符@ ();
```

```
Complex & operator ! ();   //!Complex
Complex & operator + ();   //+Complex
Complex & operator ++ ();  //++Complex
Complex & operator () ();  //Complex()
……
```

注意："+"号在这里是单目运算符，不适用于双目运算

# 4.2运算符重载规则

```
1.  class Date {
2.      int d, m, y;
3.  public:
4.      Date(int dd=0, int mm=0, int yy=0);
5.      void addDay();
6.      Date operator ++(); //声明前置运算符++
7.      void Print() { cout << d << "," << m << ","
8.                           << y << endl; }
9.      ...
10. };
11. Date Date::operator ++() //定义前置运算符++
12. {   add_day();   //因为是前置++,所以先日期自增,然后返回当前值
13.     return *this;
14. }
15. void main()
16. {   Date today(1,1,1900);
17.     for (int i=1; i<32; i++) {
18.         (++today).Print();   //使用前置运算符
19.     }
20. }
```

- 对于后置的单目运算符，在声明上要和前置有所区分：

| 运算符类别 | 运算符及其原功能 |
| --- | --- |
| 自增自减运算符 | ++(后置自增)，--(后置自减) |

**类型名 operator @(int)**

```
Complex & operator ++ (int); //Complex++
Complex & operator -- (int); //Complex--
……
```

为了和前置++、--运算符区分，增加了一个int参数

- 前置单目运算符（@操作数）

**函数类型 类名::operator @()**

使用示例：`Date today(1,1,1900); ++today.Print();`

- 后置单目运算符（操作数@）

**函数类型 类名::operator @(int)**

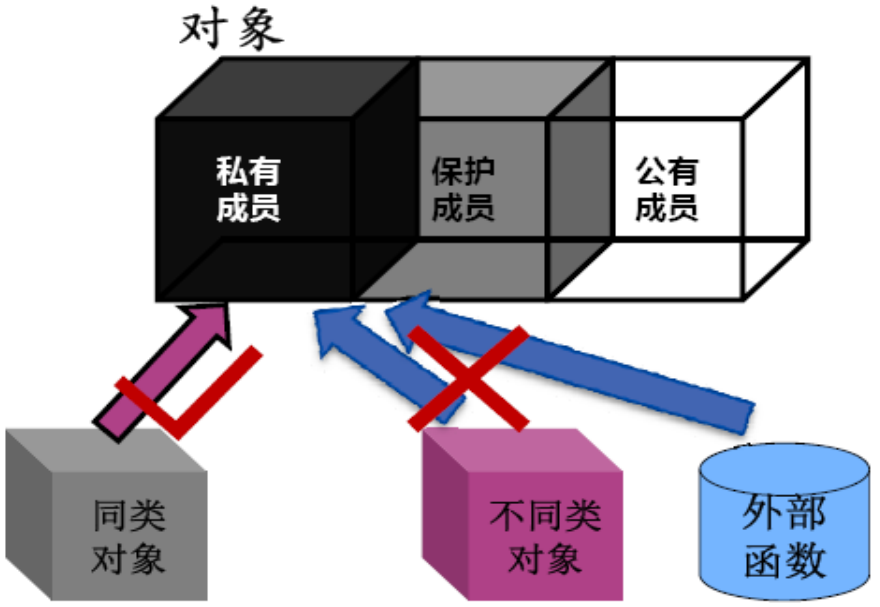
使用示例：`Date today(1,1,1900); today++.Print();`

```
1.  class Date {
2.      int d, m, y;
3.  public:
4.      Date(int dd=0, int mm=0, int yy=0);
5.      void addDay();
6.      Date operator ++(int); //声明后置运算符++，参数名不用写
7.      void Print() { cout<<d<<",">>m<<",">>y<<endl; }
8.      ...
9.  };
10. Date Date::operator ++(int) //定义后置运算符++
11. {   Date tmp(*this);
12.     add_day();
13.     return tmp; //因为是后置++,所以先返回当前值,然后日期自增
14. }
15. void main()
16. {   Date today(1,1,1900);
17.     for (int i=1; i<32; i++) {
18.         (today++).Print();  //使用前置运算符
19.     }
20. }
```

# 4.3 友元及友元重载函数

- 类的成员访问权限

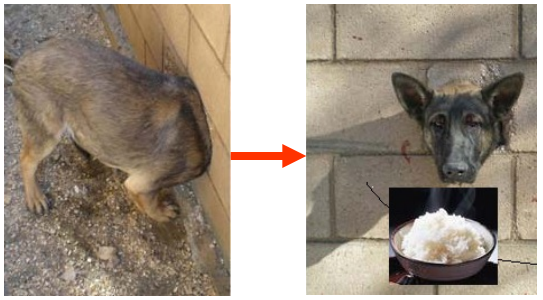
```
class 类名称
{
public:
    公有成员
protected:
    保护成员
private:
    私有成员
};
```




- 问题：类的私有成员如果需要被类外函数或不同类的对象访问？怎么办？

# 4.3 友元及友元重载函数

1. 将外部函数声明为类的友元



```
class Date {
    int d, m, y;
public:
    friend void Print(const Date& D); //外部函数
    ...
};
void Print(const Date& D)
{   //注意:友元函数是外部函数,必须通过对象来访问各成员
    cout << D.d << "," << D.m << "," << D.y << endl;
}
```

- 运算符重载为友元函数（类外）

所有操作数均作为函数参数

```
friend 函数类型 operator 运算符名称(参数表列);
```

```
Complex operator+(Complex& C1, Complex& C2)
{ ... }
c1 + c2 → operator+(c1, c2)
```

- 重载为成员函数与友元函数的比较

| | 重载为成员函数 | 重载为友元函数 |
|---|---|---|
| this指针 | 有this指针 | 没有this指针 |
| 双目运算操作数 | 要求第一个操作数必须为本类对象 | 不要求第一个操作数必须为本类对象 |
| 对封装的影响 | 不会破坏类的封装 | 会部分破坏类的封装 |
| 不适用的场合 | >>、<<运算符不能重载为成员函数 | =、[]、()不能重载为友元函数 |

```
1  #include <iostream>
2  using namespace std;
3  class Complex
4  {public:
5  Complex( ){real=0;imag=0;}
6  Complex(double r,double i){real=r;imag=i;}
7  friend Complex operator + (Complex &c1,Complex &c2);//重载函数作为友元函数
8  void display( );
9   private:
10 double real;
11 double imag;};
12
13 Complex operator + (Complex &c1,Complex &c2)        //定义作为友元函数的重载函数
14 {return Complex(c1.real+c2.real, c1.imag+c2.imag);}
15
16 void Complex::display( )
17 {cout<<"("<<real<<","<<imag<<"i)"<<endl;}
18 int main( )
19 {Complex c1(3,4),c2(5,-10),c3;
20 c3=c1+c2;
21 cout<<"c1="; c1.display( );
22 cout<<"c2="; c2.display( );
23 cout<<"c1+c2 ="; c3.display( );}
24
```

```
C:\WINDOWS\system32\cmd.exe
c1=(3,4i)
c2=(5,-10i)
c1+c2=(8,-6i)
请按任意键继续. . .
```

监视 1

| 名称 | 值 | 类型 |
|---|---|---|
| imag | 4.0000000000000000 | double |
| real | 3.0000000000000000 | double |
| c2 | {real=5.0000000000000000 imag=-10.00000 | Complex |
| imag | -10.000000000000000 | double |
| real | 5.0000000000000000 | double |
| c3 | {real=8.0000000000000000 imag=-6.000000 | Complex |
| imag | -6.0000000000000000 | double |
| real | 8.0000000000000000 | double |

自动窗口 | 局部变量 | 线程 | 模块 | 监视 1

# 4.3 友元及友元重载函数

2. 将其它类的成员函数声明为类的友元

```
class Date {
    friend Student::Print(); //其它类的成员函数
    ...
};
class Student {
    Date birthday;
public:
    void Print();
};
void Student::Print()
{
    cout << birthday.d << "," << birthday.m << ","
         << birthday.y << endl;
```

```
#include <iostream>
using namespace std;
class Date;                  //对Date类的提前引用声明
class Time                   //定义Time类
{public:
Time(int,int,int);
void display(Date &);    //display是成员函数，形参是Date类对象的引用
 private:
int hour;
int minute;
int sec;};
class Date                          //声明Date类
{public:
Date(int,int,int);
friend void Time::display(Date &);   //声明Time中的display函数为友元成员函数
 private:
int month;
int day;
int year;};
Time::Time(int h,int m,int s)    //类Time的构造函数
{hour=h;
minute=m;
sec=s;}
void Time::display(Date &d)       //display的作用是输出年、月、日和时、分、秒
{cout<<d.month<<"/"<<d.day<<"/"<<d.year<<endl;  //引用Date类对象中的私有数据
cout<<hour<<":"<<minute<<":"<<sec<<endl; }       //引用本类对象中的私有数据
Date::Date(int m,int d,int y)         //类Date的构造函数
{month=m;
day=d;
year=y;}
int main( )
{Time t1(10,13,56);            //定义Time类对象t1
Date d1(12,25,2004);           //定义Date类对象d1
t1.display(d1);               //调用t1中的display函数，实参是Date类对象d1
return 0;}
```

```
e:\vc-execise\test2\debug\test2.exe
12/25/2004
10:13:56
```

局部变量

| 名称 | 值 | 类型 |
|---|---|---|
| d1 | {month=12 day=25 y | Date |
| day | 25 | int |
| month | 12 | int |
| year | 2004 | int |
| t1 | {hour=10 minute=13 | Time |
| hour | 10 | int |
| minute | 13 | int |
| sec | 56 | int |

# 4.3 友元及友元重载函数

3．将其它类声明为类的友元

```
class Date
{
    friend Student; //外部类
    ...
};
class Student {
    Date birthday;
public:
    void Print();
    void Copy();
    ...
};
```

```
void Student::Print()
{
    //可以访问birthday的私有成员
}
void Student::Copy()
{
    //可以访问birthday的私有成员
}
```

- 如需对“友元”更多了解，请阅读“课外阅读材料：C++ 友元函数详解”。

# 运算符重载规则小结

1) 重载不能改变运算符操作数的个数、优先级别、结合性；

2) 重载运算符的函数不能有默认的参数；

3) 重载运算符必须和用户定义的自定义类型的对象一起使用，其参数至少应有一个是类对象(或类对象的引用）；

4) 用于类对象的运算符必须重载，但有两个例外，运算符“=”和“&”不必用户重载；

5) 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；

6) 运算符重载函数可以是类的成员函数或类的友元函数；

7) 友元的关系是单向的而不是双向的；友元的关系不能传递。

# 第4次作业（必做题）

1. 完成类中友元函数（申明形式如红线）的定义形式

```
1.  class Date {
2.      int d, m, y;
3.  public:
4.      Date(int dd=0, int mm=0, int yy=0);
5.      void addDay();
6.      friend Date operator ++(Date& D);          //声明前置++
7.      friend Date operator ++(Date& D, int); //声明后置++
8.      void Print() { cout << d << "," << m << ","
9.                           << y << endl; }
10.     ...
11. };
```

2. C++在C基础上增加一个新的数据类型—字符串类型（string），其实本质上它是C++在标准库中申明了一个字符类。请完成string类申明，并采用类的成员函数重载==，>，< ，+，= 5种运算符。然后，在5子棋盘前3次需求基础上，使用上述某些运算符，通过比较玩家名称（英文表示）来决出谁先手（名字大者先下棋）。

- **也可自己设计需求，将上述5种运算符都应用在五子棋程序中。**

# 第4次作业（选做题）

第1道题：阅读附录资料—友元，思考如下问题，分析为什么？

1）友元函数是否可以申明在类中的private?;

2）类中申明的友元函数是否有this指针？；

3）如何使得友元函数获得操作对象？；

4）为什么说，使用友元函数破坏了类的封装性，而使用友元类增强了类的封装性？运用类封装性和友元类的原理来分析？