

第4章

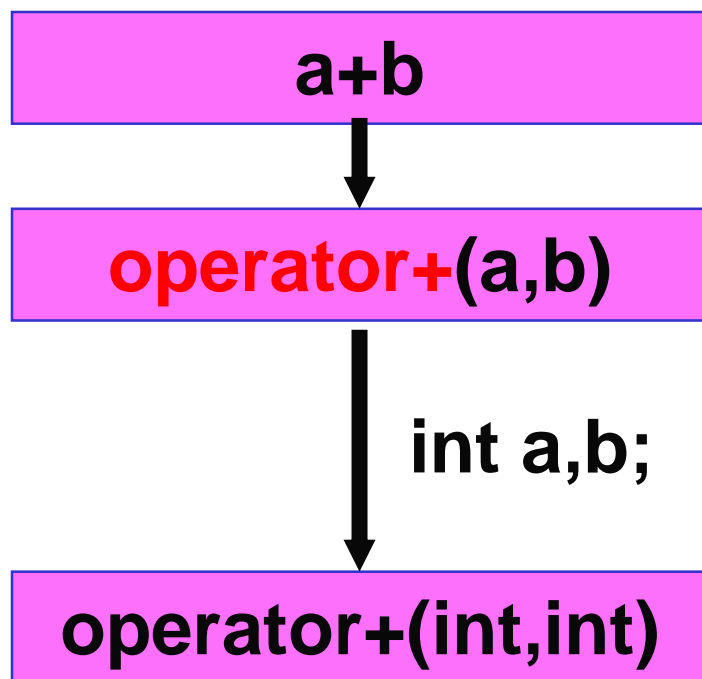
运算符重载

内容安排

- ❖ 1 运算符重载
- ❖ 2 运算符重载的规则
- ❖ 3 重载为类的成员函数
- ❖ 4 重载为类的友元函数
- ❖ 5 典型运算符的重载

1 运算符重载

❖ 在C++中，所有系统预定义的运算符都是通过运算符函数来实现的。



❖ **运算符重载**是指对已有的运算符赋予它新的含义，是通过运算重载函数来实现的，本质上也是属于函数重载，是C++实现静态多态的重要手段。

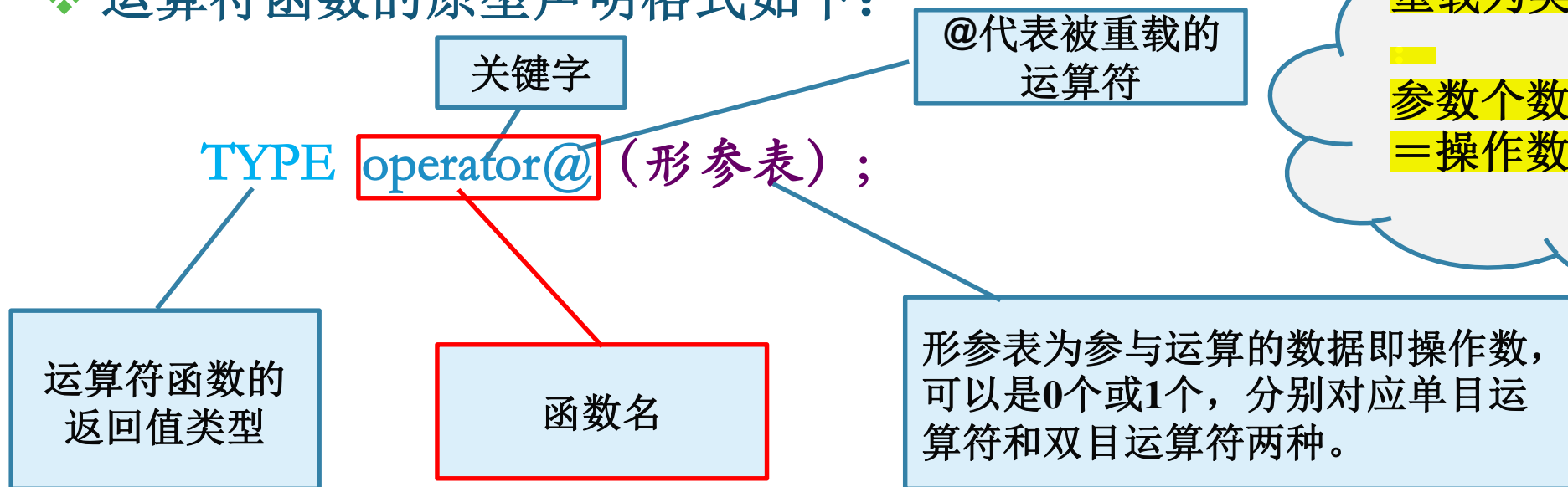
❖ 在用户自定义的新类中可以重载这些函数。

2 C++运算符重载的规则

- ❖ 不允许定义新的运算符，只能对已有预定义运算符进行重载。
- ❖ 不能改变运算符操作数（对象）的个数。
- ❖ 不能改变该运算符的优先级别和结合性。
- ❖ 运算符重载应该符合实际需要，重载的功能应该与运算符原有的功能相似。
- ❖ 大多数预定义的运算符都可以被重载（除成员访问运算符“.”、成员指针运算符“.*”、作用域运算符“::”、条件运算符“?:”、sizeof外）。
- ❖ 在类中对运算符进行重载的方式（对该类对象进行运算）：
 - 可重载为类的成员函数
 - 可重载为类的友元函数

3 重载为类的成员函数

- ❖ 在类中定义一个同名的运算符函数来重载该函数。
- ❖ 运算符函数的原型声明格式如下：



重载为类的成员函数：

参数个数

=操作数个数-1

Complex operator+(Complex&);

3 重载为类的成员函数

❖ 由于将运算符函数重载为类的成员函数，所以，操作的一方当然是当前对象：

- 如果重载**单目运算符**（例如**++**，**--**），就**不必**另设置参数，参数为**0**个；

```
Complex operator++();
```

```
Complex a;  
++a;
```

```
a.operator++();
```

- 如果是重载**双目运算符**（例如**+**，**-**，*****，**/**），**只设置一个参数**作为右侧运算量，而左侧运算量就是该对象本身。

```
Complex operator+(Complex&);
```

```
Complex a,b;  
a=a+b;
```

```
a=a.operator+(b);
```



❖ 【例4-1】 定义一个表示复数的类**Complex**，并在该类中对运算符“+”进行重载，以实现两个复数的加运算，要求将运算符重载为类的成员函数。

注意： **a+b**等价于**a.operator+(b)**

```

❖ #include <iostream>
❖ using namespace std;
❖ class Complex
❖ {
❖     protected:
❖         double r;           //实部
❖         double i;           //虚部
❖     public:
❖         Complex(double x=0.0, double y=0.0)
❖         {
❖             r = x;
❖             i = y;
❖         }
❖         Complex operator+(Complex &);
❖         void show()
❖         {
❖             cout<<"("<<r<<","<<i<<")";
❖         }
❖ };

```

函数名

两个复数相加结果仍是复数，所以函数返回值类型是Complex

双目运算符重载为成员函数，只有一个参数

❖ **Complex** **Complex::operator+(Complex & c)**

一个形参

❖ {

❖ **Complex temp;**

❖ **temp.r = this->r + c.r;**

❖ **temp.i = this->i + c.i;**

❖ **return temp;**

❖ }

❖ **int main()**

❖ {

❖ **Complex c1(10.0,20.0), c2(30.0,40.0), c3;**

❖ **c3 = c1 + c2;** **c3 = c1.operator+(c2);**

❖ **c1.show();**

❖ **cout<<"+";**

❖ **c2.show();**

❖ **cout<<"=";**

❖ **c3.show();**

❖ **cout<<endl;**

❖ **return 0;**

❖ }

返回值类型

类名

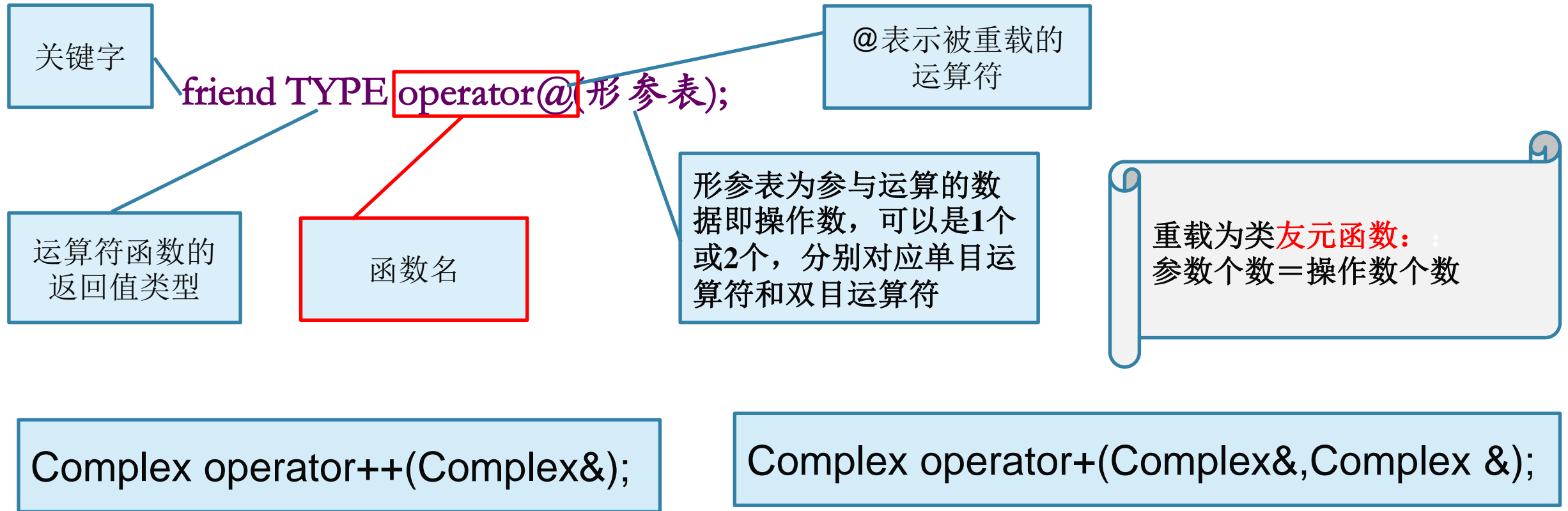
函数名

(10, 20) + (30, 40) = (40, 60)

返回

4 重载为类的友元函数

- ❖ 定义一个与某一运算符函数同名的全局函数;
- ❖ 然后再将该全局函数声明为类的友元函数, 从而实现运算符的重载。





❖ 【例4-2】 定义一个表示复数的类**Complex**，并在该类中对运算符“-”进行重载，以实现两个复数的减运算，要求将运算符重载为类的友元函数。

a-b等价于**operator -(a,b)**

```

❖ #include<iostream>
❖ using namespace std;

❖ class Complex
❖ {
❖ private:
❖     double r;
❖     double i;
❖ public:
❖     Complex(double x=0.0, double y=0.0)
❖     {
❖         r = x;
❖         i = y;
❖     }
❖     friend Complex operator-(const Complex & c1, const Complex & c2);
❖     void show()
❖     {
❖         cout<<"("<<r<<","<<i<<")";
❖     }
❖ };

```

❖ Complex operator-(const Complex & c1, const Complex & c2)

```
❖ {  
❖     Complex temp;  
❖     temp.r = c1.r - c2.r;  
❖     temp.i = c1.i - c2.i;  
❖     return temp;  
❖ }
```

```
❖ int main()
```

```
❖ {  
❖     Complex c1(10.0,20.0), c2(30.0,40.0), c3;  
❖     c3 = c1 - c2;  
❖     c1.show();  
❖     cout<<"-";  
❖     c2.show();  
❖     cout<<"=";  
❖     c3.show();  
❖     cout<<endl;  
❖     return 0;  
❖ }
```

```
Complex Complex::operator-(const Complex & c2)  
{  
    Complex temp;  
    temp.r = r - c2.r;  
    temp.i = i - c2.i;  
    return temp;  
}
```

`c3 = operator-(c1,c2);`

`c3 = c1.operator-(c2);`

$(10, 20) - (30, 40) = (-20, -20)$

`Complex operator-(const Complex & c2);`


思考：-重载为成员函数怎么实现？

总结

- (1) 当运算符重载函数重载为类的成员函数时，形参个数要比运算符操作数个数少一个；若重载为友元函数，则参数个数与操作数个数相同。
- (2) 前置单目运算符重载为类的成员函数时，不需要形参；后置单目运算符重载为类的成员函数时，函数要带有一个整型形参
- (3) 运算符函数 `operator @()` 通常被声明为类的成员函数或友元函数。其等价的函数调用形式如下表所示。

表达式↵	友元函数调用↵	成员函数调用↵
<code>a+b↵</code>	<code>operator+(a,b)↵</code>	<code>a.operator+(b)↵</code>
<code>a++↵</code>	<code>operator++(a,0)↵</code>	<code>a.operator++(0)↵</code>
<code>-a↵</code>	<code>operator-(a)↵</code>	<code>a.operator-()↵</code>

两种重载形式的比较

- 
- (1) 有些运算符只能重载为类的成员函数，如：=、()、[]、->。有些运算符只能重载为友元函数，如：<<，>>。
- (2) 一般地，单目运算符和复合赋值运算符最好重载为类的成员函数；双目运算符则最好重载为类的友元函数。
- (3) 类型转换函数只能重载为类的成员函数而不能重载为类的友元函数。比如
`operator int();`
- (4) 若一个运算符的操作需要修改对象的状态，选择重载为成员函数较合适。

5 典型运算符的重载

- ❖ (1) 赋值运算符的重载（只能重载为类的成员函数）
- ❖ (2) 单目运算符的重载
- ❖ (3) I/O运算符的重载（只能重载为类的友元函数）
- ❖ (4) 数组下标运算符[]（只能重载为类的成员函数）

(1) 赋值运算符的重载

- ❖ “=” 运算符是双目运算符，重载时，注意：
 - 赋值运算符函数必须是类的成员函数，不允许重载为友元函数；
 - 赋值运算符函数不能被派生类继承。
 - 如果不定义自己的赋值运算符函数，那么编译器会自动生成一个默认的赋值运算符函数。该函数把一个对象逐位地拷贝给要赋值的对象，可以使用默认的赋值运算符实现对象的赋值。在没有特殊处理的情况下（例如对内存的动态分配等），只使用默认的赋值运算符函数即可
 - 赋值运算符应该返回对*this的引用
- ❖ 【例4-3】创建一个字符串类String，并重载赋值运算符“=”，实现字符串之间的赋值运算。

```
❖ #include<iostream>
❖ #include<cstring>
❖ using namespace std;
❖ class String
❖ {
❖ private:
❖     char * str;
❖ public:
❖     String(char * pstr = "")
❖     {
❖         str = new char[strlen(pstr)+1];
❖         strcpy_s(str, strlen(pstr)+1,pstr);
❖     }
❖     ~String()
❖     {
❖         delete [] str;
❖     }
❖     void show()
❖     {
❖         cout<<str<<endl;
❖     }
❖     String & operator=(String & );
❖ };
```

```
❖ String & String::operator=(String & s)
❖ {
❖     delete [] str;
❖     str = new char[strlen(s.str)+1];
❖     strcpy_s(str, strlen(s.str)+1, s.str);
❖     return *this;
❖ }
❖ int main()
❖ {
❖     String s1("Hello world."), s2;
❖     s1.show();
❖     s2 = s1; s2.operator=(s1);
❖     s2.show();
❖     return 0;
❖ }
```

练习1

❖ 创建一个复数类 **Complex**，并重载赋值运算符“=”，实现复数之间的赋值运算。

(2) 单目运算符的重载

- ❖ 单目运算符可以重载为类的成员函数重载，也可以作为类的友元函数重载；
- ❖ 作为成员函数重载时没有参数，而作为友元函数重载时有一个参数。
- ❖ 【例4-4】创建一个字符串类**String**，并重载运算符“!”为其成员函数，用于判断对象中的字符串是否为空。

```
❖ #include<iostream>
❖ #include<cstring>
❖ using namespace std;
❖ class String
❖ {
❖ private:
❖     char * str;
❖ public:
❖     String(char * pstr = "")
❖     {
❖         str = new char[strlen(pstr)+1];
❖         strcpy_s(str, strlen(pstr) + 1, pstr);
❖     }
❖     ~String()
❖     {
❖         delete [] str;
❖     }
❖     void show()
❖     {
❖         cout<<str<<endl;
❖     }
❖     bool operator!();
❖ };
```

```
❖ bool String::operator!()
❖ {
❖     if(strlen(str)==0)
❖         return true;
❖     return false;
❖ }
❖ int main()
❖ {
❖     String s1("Hello world."), s2;
❖     if(!s1) 表达式调用
❖         cout<<"s1 is null."<<"\n";
❖     else
❖         s1.show();
❖     if(s2.operator!()) 通过成员函数调用
❖         cout<<"s2 is null."<<"\n";
❖     else
❖         s2.show();
❖     return 0;
❖ }
```

练习2

❖ 创建一个复数类**Complex**，并重载前缀“++”运算符和后缀“++”运算符，将其重载为成员函数，实现复数的实部加1和虚部加1。

❖ 说明：

- 以成员函数方式重载前缀“++”运算符，函数原型声明如下：
类型 **operator++()**;
- 以成员函数方式重载后缀“++”运算符，函数原型声明如下：
类型 **operator++(int)**;

- ❖ 还可以将单目运算符重载为类的友元函数，这时需要有一个参数。
- ❖ 【例4-5】 创建一个字符串类**String**，并重载运算符“！”为其友元函数，用于判断对象中的字符串是否为空。

```
❖ #include<iostream>
❖ #include<cstring>
❖ using namespace std;
❖ class String
❖ {
❖ private:
❖     char * str;
❖ public:
❖     String(char * pstr = "")
❖     {
❖         str = new char[strlen(pstr)+1];
❖         strcpy_s(str, strlen(pstr)+1, pstr);
❖     }
❖     ~String()
❖     {
❖         delete [] str;
❖     }
❖     void show()
❖     {
❖         cout<<str<<endl;
❖     }
❖     friend bool operator!(String &);
❖ };
```

```
❖ bool String::operator!(String &s)
❖ {
❖     if(strlen(s.str)==0)
❖         return true;
❖     return false;
❖ }
❖ int main()
❖ {
❖     String s1("Hello world."), s2;
❖     if(!s1)
❖         cout<<"s1 is null."<<"\n";
❖     else
❖         s1.show();
❖     if(operator!(s2))
❖         cout<<"s2 is null."<<"\n";
❖     else
❖         s2.show();
❖     return 0;
❖ }
```


练习3

❖ 创建一个复数类**Complex**，并重载前缀“--”运算符和后缀“--”运算符，将其重载为友元函数，实现复数的实部减1和虚部减1。

❖ 说明：

- 以友元函数方式重载前缀“--”运算符，函数原型声明如下：
类型 `operator--(类名 &);`
- 以友元函数方式重载后缀“--”运算符，函数原型声明如下：
类型 `operator--(类名&, int);`

(3) I/O运算符的重载

- ❖ C++的I/O流库的一个重要特性就是能够支持新的数据类型的输入和输出。
- ❖ 用户可以通过对提取运算符“>>”和插入运算符“<<”进行重载，来实现对新的数据类型的数据的输入和输出操作。
- ❖ 一般来讲，输出应该尽量小地格式化，**不输出换行符**；而输入操作需要考虑出错和文件结束的可能性。
- ❖ 【例4-6】定义一个Complex类，友元重载提取运算符“>>”和插入运算符“<<”重载完成复数的输入和输出。

```
❖ #include <iostream>
❖ using namespace std;
❖ class Complex
❖ {
❖ protected:
❖     double r;//实部
❖     double i;//虚部
❖ public:
❖     Complex(double x = 0.0, double y = 0.0)
❖     {
❖         r = x;
❖         i = y;
❖     }
❖     friend istream& operator>>(istream &,Complex &);
❖     friend ostream& operator<<(ostream &, const Complex &);
❖ };
```

提取运算符
的函数返回
值类型

第1个参数
的类型

插入运算符
的函数返回
值类型

第1个参数
的类型

```

❖ istream& operator >> (istream &is, Complex &c)
❖ {
❖     is >> c.r >> c.i;
❖     return is;
❖ }
❖ ostream& operator<<(ostream &os, const Complex &c)
❖ {
❖     os << "(" << c.r << "," << c.i << ")" ;
❖     return os;
❖ }
❖ int main()
❖ {
❖     Complex c1;
❖     cout << "请输入复数的实部和虚部: ";
❖     cin >> c1;
❖     cout << c1 << endl;
❖     return 0;
❖ }

```

(4) 数组下标运算符[]

❖ 数组下标运算符“[]”，它只能被重载为成员运算符函数。

❖ 在重载时，把“[]”看作双目运算符。

❖ 成员运算符函数`operator[]()`的一般形式为：

返回类型 & `operator [] (int i)`

{

函数体

}

❖ **注意：**一定以某类型的引用返回，目的是使函数返回值可以作为左值使用

❖ **【例4-7】**定义一个**String**类，重载运算符“[]”。

```

❖ #include<iostream>
❖ #include<cstring>
❖ using namespace std;
❖ class String
❖ {
❖ private:
❖     char * str;
❖     int length;
❖ public:
❖     String(char * pstr = "")
❖     {
❖         length = strlen(pstr);
❖         str = new char[strlen(pstr) + 1];
❖         strcpy_s(str, strlen(pstr) + 1, pstr);
❖     }
❖     ~String()
❖     {
❖         delete[] str;
❖     }
❖     void show()
❖     {
❖         cout << str << endl;
❖     }
❖     char& operator[](int i);
❖ };

```

```

❖ char& String::operator[](int i)
❖ {
❖     //对下标进行检查，超出范围则报错退出程序
❖     if (i<0 || i >= length)
❖     {
❖         cout << "下标越界错误！" << endl;
❖         exit(1);
❖     }
❖     return str[i];
❖ }

❖ int main()
❖ {
❖     String s1("Hello world.");
❖     s1.show();
❖     cout << s1[1] << endl;
❖     return 0;
❖ }

```

```

Hello world.
e

```

练习4

❖ 设计一个复数类**Complex**，通过运算符重载，实现复数的输入输出、 $+$ 、 $-$ 、 $*$ 、 $/$ 、 $+=$ 、 $-=$ 、 $*=$ 、 $/=$ 、 $==$ 、 $!=$ 相关运算。