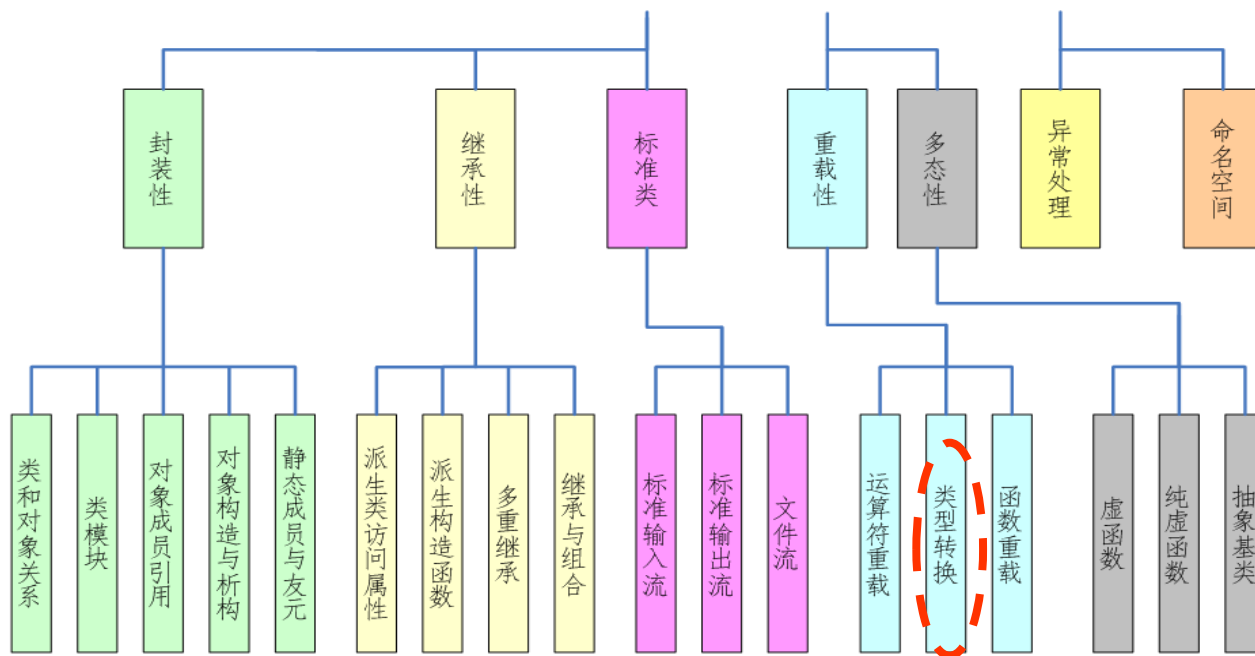


第5讲 运算符重载(下)

5.1 运算符重载实例

5.2 不同类型数据间的转换

C++的OOP程序 = 对象 + 消息 + 工具



<< >>的来世今生

`cout<<2<<3;`会输出23

`cout<<(2<<3);`会输出16

- << >>是左/右移运算符，左/右移运算符是用来将一个数的各二进制位全部左/右移若干位,右/左补0。高位左移后溢出舍弃。例如：将a的二进制数左移2位，右补0。若a=15，即二进制数00001111，左移2位得00111100，即十进制数60。
- 语法格式：需要移位的数字 << 移位的次数。

>> (右移): $x \gg y == x / (2^y)$

`cout<<(8>>1);`会输出4//等于 $8/(2^1)$;

`cout<<(8>>2);`会输出2//等于 $8/(2^2)$;

5.1 运算符重载实例

- `>>/<<`提取/插入运算符从输入/输出流对象（`cin/cout`）中提取/插入数据；双目，左数`cin/cout`；右数为标准类型变量（在 `iostream` 类对 `>>/<<`进行了重载），也可是字符串变量（字符数组或指针）；提取数据时，以空白符（空格、enter、tab）作为数据分隔符，因此提取字符串数据时不能提取空白字符。

```
ostream & ostream::operator<<(int n)
{
    .... // 输出n整型的代码
    return *this;
}

ostream & ostream::operator<<(const char * s)
{
    .... // 输出s字符串的代码
    return *this;
}
```

```
#include<iostream>
using namespace std;
int main()
{
    int s;
    //在键盘输入内容前, cin还没有用到,
    //输完内容之后才把内容以字符串流的形式传给cin流对象
    //cin就变成字符串流, 使用提取运算符 ">>" 从设备键盘取得数据,
    //读取类型符合时, 就把读取的数据赋值给s;
    cin>>s;
    cout<<endl;
    cout<<s<<endl;
    return 0;
}
```

`std::cout << 1;` 语句, 等价于 `cout.operator<<(1);`

`std::cout << "hello";` 语句, 等价于 `cout.operator<<("hello");`

5.1 运算符重载实例

问题：在使用“cout<<”时为啥要#include <iostream>;

□ C++提供输入流类istream和输出流类ostream;cin和cout分别是istream类和ostream类的对象，且是全局对象。

□ 流插入运算符“<<”和流提取运算符“>>”是C++在istream类和ostream类中进行了重载，使之用来输出和输入C++标准类型数据。

□ 用户自定义类型数据（类），是否可以用“<<”和“>>”进行输出和输入呢？当然可以，需要再次重载；

□ 对“<<”和“>>”重载的函数形式如下：

```
istream & operator >> (istream &, 自定义类&);
```

```
ostream & operator << (ostream &, 自定义类&);
```

```

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);           //运算符“+”重载为成员函数
8      friend ostream& operator << (ostream&,Complex&); //运算符“<<”重载为友元函数
9  private:
10     double real;
11     double imag;};
12
13  Complex Complex::operator + (Complex &c2)       //定义运算符“+”重载函数
14  {return Complex(real+c2.real,imag+c2.imag);}
15  ostream& operator << (ostream& output,Complex& c) //定义运算符“<<”重载函数
16  {output<<"("<<c.real<<"+"<<c.imag<<"i)"<<endl;
17  return output;}
18
19  int main()
20  {Complex c1(2,4),c2(6,10),c3;
21   c3=c1+c2;
22   cout<<c3;
23   return 0;}

```

重载流插入运算符 “<<”

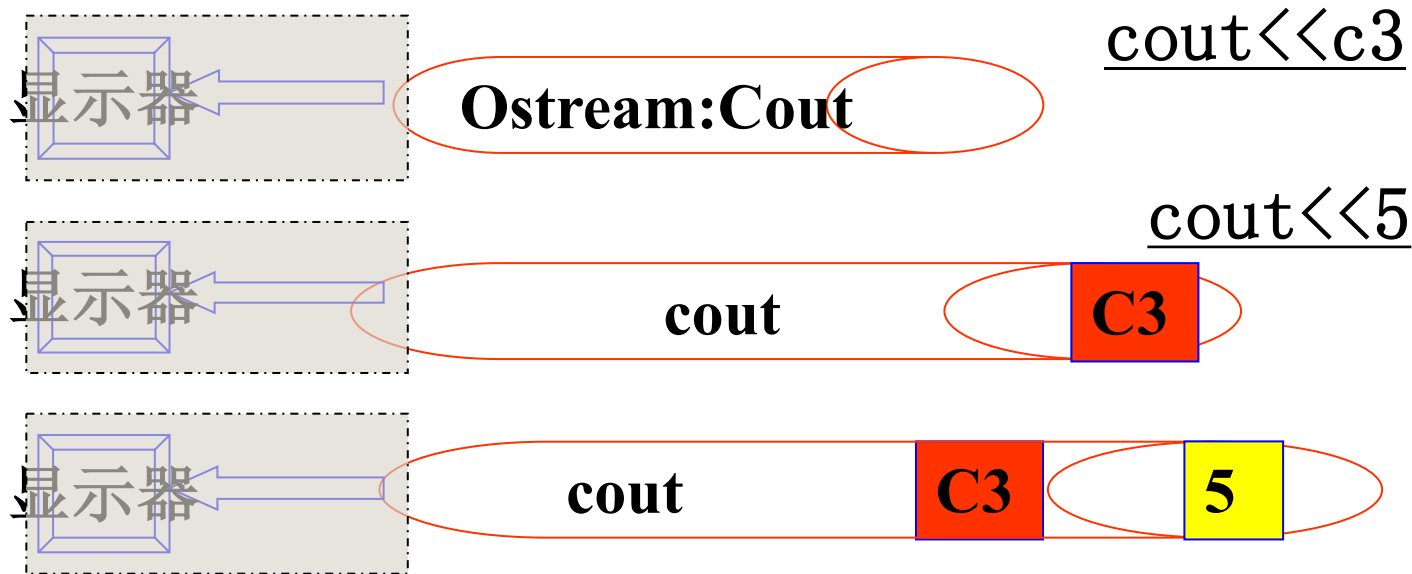
C:\WINDOWS\system32\cmd.exe

<8+14i>

请按任意键继续. . .

(1) 重载函数中的return output的作用是什么？在本程序中output的实参是哪个？

(2) 如何区分什么情况下的“<<”是标准类型数据的流插入符，什么情况下“<<”是重载的流插入符。如 cout<<c3<<5<<endl；



```

1  #include <iostream>
2  using namespace std;
3  class Complex
4  {public:
5  friend ostream& operator << (ostream&,Complex&); //声明重载运算符"<<"
6  friend istream& operator >> (istream&,Complex&); //声明重载运算符">>"
7  private:
8  double real;
9  double imag;};
10 ostream& operator << (ostream& output,Complex& c)
11 {output<<"("<<c.real;
12 if(c.imag>=0) output<<"+"; //虚部为正数时，在虚部前加"+"号
13 output<<c.imag<<"i)"<<endl; //虚部为负数时，在虚部前不加"+"号
14 return output;
15 }
16
17 istream& operator >> (istream& input,Complex& c) //定义重载运算符">>"
18 {cout<<"input real part and imaginary part of complex number:";
19 input>>c.real>>c.imag;
20 return input;}
21
22 int main( )
23 {Complex c1,c2;
24 cin>>c1>>c2;
25 cout<<"c1="<<c1<<endl;
26 cout<<"c2="<<c2<<endl;
27 return 0;}

```

重载流提取运算符 ">>"

C:\WINDOWS\system32\cmd.exe

```

input real part and imaginary part of complex number:3 4
input real part and imaginary part of complex number:3 -4
c1=(3+4i)
c2=(3-4i)
请按任意键继续. . .

```

```

#include <iostream>
#include <assert.h>
using namespace std;
class Vector
{
    int * data; int size;
public:
    Vector();
    Vector(int * pData, int n);
    ~Vector();
    int SetSize(int n);
    int GetSize() const;
    int & operator [] (int n);};
Vector::Vector()
{
    data = NULL;
    size = 0;}
Vector::Vector(int * pData, int n)
{
    data = pData;
    size = n;}
Vector::~~Vector()
{
    delete []data;}
int Vector::GetSize() const
{
    return size;}
int Vector::SetSize(int n)
{
    delete []data;
    data = new int[n];
    size = n;
    return size;}

```

□ 下标运算符[]是C++常用运算符。但[]没有定义下标越界的检测功能。而下标越界的错误很容易产生。因此，通过运算符重载，对下标运算符增加一个越界检测功能

```

int & Vector::operator[] (int n)
{
    assert(n >= 0 && n < size);
    return data[n];
}

int main()
{
    Vector v;
    int size = 10;
    v.SetSize(size);
    for(int i = 0; i < size; i++)
        v[i] = i;
    cout<<v[3]<<endl;
    cout<<v[10]<<endl;
    return 0;
}

```


assert (CRT)

[Example](#) [See Also](#) [Send Feedback](#)

Evaluates an expression and, when the result is **false**, prints a diagnostic message and aborts the program.

```
void assert(  
    int expression  
);
```

Parameters

expression

Expression (including pointers) that evaluates to nonzero or 0.

```
// crt_assert.c
// compile with: /c
#include <stdio.h>
#include <assert.h>
#include <string.h>

void analyze_string( char *string );    // Prototype

int main( void )
{
    char  test1[] = "abc", *test2 = NULL, test3[] = "";

    printf ( "Analyzing string '%s'\n", test1 ); fflush( stdout );
    analyze_string( test1 );
    printf ( "Analyzing string '%s'\n", test2 ); fflush( stdout );
    analyze_string( test2 );
    printf ( "Analyzing string '%s'\n", test3 ); fflush( stdout );
    analyze_string( test3 );
}

// Tests a string to see if it is NULL,
// empty, or longer than 0 characters.
void analyze_string( char * string )
{
    assert( string != NULL );           // Cannot be NULL
    assert( *string != '\0' );         // Cannot be empty
    assert( strlen( string ) > 2 );    // Length must exceed 2
}
```

回顾：二维数据引用：`double data[3][3];`

`Data[1][2]=data[1][1]+1;`是否表达为：`Data(1,2)=data(1,1)+1`

```
#include <iostream>
#include <assert.h>
using namespace std;
class Matrix
{double *data;int row;int col;
public:
    Matrix();
    Matrix(double * pData, int r, int c);
    ~Matrix();
    double & operator()(int r, int c);};
Matrix::Matrix()
{ data = NULL;
  row = 0;
  col = 0;}

Matrix::Matrix(double * pData, int r, int c)
{ data = pData;
  row = r;
  col = c;}

Matrix::~~Matrix()
{ delete []data;}
```

```
double & Matrix::operator()(int r, int c)
{ assert(r >= 0 && r < row && c >= 0 && c < col);
  return data[col * r + c];}

int main()
{ double * pData = new double[9];
  int row = 3;
  int col = 3;
  Matrix m(pData, row, col);
  int cnt = 1;
  for(int i = 0; i < row; i++)
  {
    for(int j = 0; j < col; j++)
    {
      m(i, j) = cnt++;
      cout<<m(i, j)<<' ';
    }
    cout<<endl;
  }

  return 0;}
```

New 和Delete 的重载

内存管理运算符new、delete也可进行重载，其重载形式既可是类成员函数，也可是友元函数；且第1个参数类型必须是size_t。如果类中没有定义 new 和 delete 的重载函数，那么会自动调用缺省 new 和 delete。

New操作符：（1）分配足够的内存以容纳对象；（2）调用构造函数初始化分配内存。重载能改变只是第一步的行为：如何为对象分配RAW。

```
void * className::operator new( size_t size ){
    //TODO:
}
```

```
void className::operator delete( void *ptr){
    //TODO:
}
```

```
void * operator new(size_t unSize)
{
    printf("operator new called\n");
    return malloc(unSize);
}
```

```
void * operator new( size_t size ){
    //TODO:
}
```

```
void operator delete( void *ptr){
    //TODO:
}
```

```
void operator delete(void * pMem,size_t unSize)
{
    printf("delete1: %u\n", unSize);
    free(pMem);
}
```



```
1 #include <stdio.h>
2 #include<stdlib.h>
3 void *operator new(size_t sz)
4 {
5     printf("operator new:%d Bytes\n",sz);
6     void *m=malloc(sz);
7     if(!m)
8         puts("out of memory");
9     return m;
10 }
11 void operator delete(void *m)
12 {
13     puts("operator delete");
14     free(m);
15 }
16 class S
17 {
18     int i[100];
19 public:
20     S(){puts("S:S()");}
21     ~S(){puts("~S::~S()");}
22 };
```

```
23 int main( )
24 {
25     puts("(1)Creating & Destroying an int");
26     int *p=new int(47);
27     delete p;
28     puts("(2)Creating & Destroying an S");
29     S *s=new S;
30     delete s;
31     puts("(3)Creating & Destroying S[3]");
32     S *sa=new S[3];
33     delete []sa;
34     return 0;
35 }
```

```
(1)Creating & Destroying an int
operator new:4 Bytes
operator delete
(2)Creating & Destroying an S
operator new:400 Bytes
S:S<>
~S::~S<>
operator delete
(3)Creating & Destroying S[3]
operator new:1204 Bytes
S:S<>
S:S<>
S:S<>
~S::~S<>
~S::~S<>
~S::~S<>
operator delete
```



1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

```
1 #include<iostream>
2 using namespace std;
3
4 class Test
5 {
6     friend ostream& operator <<(ostream& out, Test& test);
7     int* pa;
8 public:
9     Test(int a = 0)
10    {
11        pa = new int(a);
12        cout << "调用构造函数:" << pa << endl;
13    }
14    ~Test()
15    {
16        cout << "调用析构函数:" << pa << endl;
17        if (pa)
18            delete pa;
19    }
```

20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

```
Test&operator=(const Test& test)//重载=Test
{
    if (pa)
        *pa = *(test.pa);
    else
        pa = new int(*(test.pa));
    return *this;
}
Test&operator=(int a)//重载=int
{
    if (pa)
        *pa = a;
    else
        pa = new int(a);
    return *this;
}
static void* operator new(size_t size)//重载operator new函数
{
    void* p = (void*)malloc(size);//调用的是malloc
    cout << "调用重载的 operator new" << endl;
    return p;
}
```

```

42 static void operator delete(void* ptest, size_t size)//
43 {
44     cout << "调用重载的operator delete" << endl;
45     free(ptest);
46 }
47 static void* operator new[](size_t size)
48 {
49     void* p = (void*)malloc(size);
50     cout << "调用重载的 operator new[]" << endl;
51     return p;
52 }
53 static void operator delete[](void* ptest, size_t size)
54 {
55     cout << "调用重载的operator delete[]" << endl;
56     free(ptest);
57 }
58 };

```



```

60 ostream& operator <<(ostream& out, Test& test)
61 {
62     out << test.pa << " " << *(test.pa);
63     return out;
64 }
65
66 int main()
67 {
68     cout << "new,delete:" << endl;
69     Test* ptest = new Test(2);
70     cout << *ptest << endl;
71     delete ptest;
72     cout << "new[],delete[]:" << endl;
73     Test* ptest1 = new Test[3];
74     for (int i = 0; i < 3; i++)
75     {
76         ptest1[i] = i + 1;
77         cout << ptest1[i] << endl;
78     }
79
80     delete[] ptest1;
81     system("pause");
82     return 0;
83 }

```

```

new,delete:
调用重载的 operator new
调用构造函数:000EB070
000EB070 2
调用析构函数: 000EB070
调用重载的operator delete
new[],delete[]:
调用重载的 operator new[]
调用构造函数:000E53C8
调用构造函数:000E5428
调用构造函数:000EB070
000E53C8 1
000E5428 2
000EB070 3
调用析构函数: 000EB070
调用析构函数: 000E5428
调用析构函数: 000E53C8
调用重载的operator delete[]
请按任意键继续. . .

```

new 先调用operator new申请内存，再调用构造函数初始化内存；delete先调用析构函数，再调用operator delete释放内存；new[]与delete[]都只会调用一次operator new[]、operator delete[]，且是从前往后构造，从后往前析

5.2 不同类型数据间的转换

◆ 标准类型数据间的转换

□ 在C++中，某些不同类型数据之间可以自动转换，例如

```
int i = 6; i = 7.5 + i;
```

□ C++还提供显式类型转换，用户可以指定将数据转换成另一类型，其形式为：类型名(数据)，如：int(89.5)；

◆ 非标准类型数据间的转换

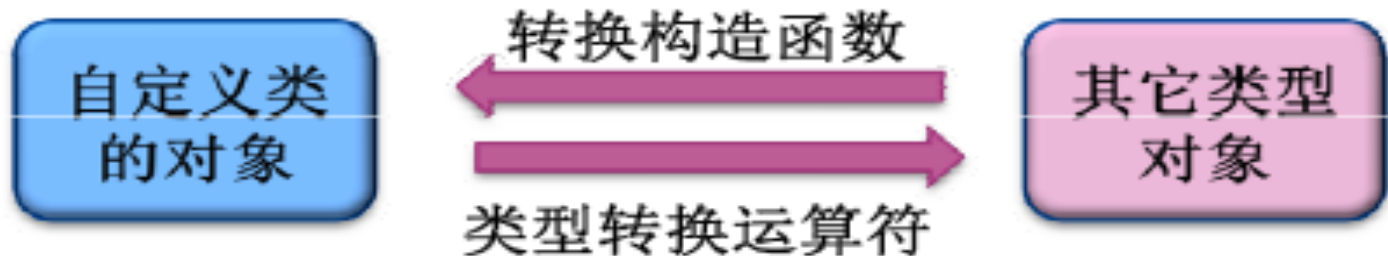
□ 对于用户自己声明的类型，即非标准类型。编译系统并不知道怎样进行转换？

□ 如何实现不同类型的类对象之间的转换呢？

```
Complex c1(3,1);  
int a = int(c1); //需要将Complex类型转换为int  
double d = 3.0;  
Complex c2 = Complex(d); //需要把double转换为Complex
```

5.2 不同类型数据间的转换

- 两种类型转换操作：



1. 转换构造函数

其它类型对象到当前类对象的自动转换

2. 类型转换运算符

当前类对象到其它类型对象的自动转换

1. 类型转换构造函数：以其他类型数据为参数来构建一个成自身类的对象；

□ 设计方法

(1) 先声明一个类；

(2) 在这个类中定义一个只有一个参数的构造函数，参数的类型是需要转换的类型，在函数体中指定转换的方法；

□ 例：`Teacher (Student&s)` ；

```
{num=s. num; strcpy (name, s. name) ; sex=s. sex; } ;
```

□ 注意：不仅可以将一个标准类型数据转换成类对象，也可以将另一个类的对象转换成转换构造函数自身类的对象。

□ 提问：对象s中的num, name, sex如果是私有成员是否可以？为什么？

□ 类型转换构造函数与有参和拷贝构造函数的区别与联系

```

1. class Complex {
2.     double re, im;
3. public:
4.     Complex() { re=0; im=0; }
5.     Complex(double r) { re=r; im=0; }
6.     Complex(double r, double i) { re=r; im=i; }
7.     friend Complex operator+(Complex C1, Complex C2);
8. };

```

```

9. Complex operator+(Complex C1, Complex C2)
10. { return Complex(C1.re+C2.re, C1.im+C2.im); }

```

```

11. void main()
12. { Complex c1(3,4), c2(5,10), c3, c4, c5;
13.     c3 = c1 + 2.5; //调用转换构造函数,相当于c3=c1+Complex(2.5)
14.     c4 = 2.5 + c2; //调用转换构造函数,相当于c4=Complex(2.5)+c2
15.     int i = 2;
16.     c5 = 2.5 + i; //调用转换构造函数,相当于c5=Complex(2.5+i)
17. }

```

注意：
这里参数类型是
Complex才能调用转
换构造函数，若是
Complex&则不能！

□ 构造函数小结:

- 默认（默认参数）构造函数;
- 带初始化参数列表构造函数;
- 拷贝构造函数; `Complex (Complex &c);`
- 类型转换构造函数; `Complex(double r) {real=r; imag=0;}`

□ 以上构造函数可同时出现在同类中，是构造函数的重载。编译系统会根据建立对象时给出的实参的个数与类型选择形参与之匹配的构造函数

提问：（1）类型转换构造与复制构造函数的区别与联系？

（2）类型转换构造函数与一个有参构造函数的区别与联系？

（3）转换构造函数能否将自身类对象转换为其他类型数据？

5.2 不同类型数据间的转换

2. 类型转换重载函数：C++提供类型转换函数(type conversion function)来解决“将一个类的对象转换成另一类型的数据”的问题

类型转换函数的一般形式：

Operator 类型名 ()
{实现转换的语句}

例：如果已声明了Complex类，可在Complex类中定义类型转换函数operator double() {return real;}

在函数名前面不能指定函数类型，函数没有参数。其返回值的类型是由函数名中指定的类型名来确定

类型转换函数只能作为成员函数，因为转换的主体是本类的对象。不能作为友元函数或普通函数

```

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      operator double( ) {return real;}
8  private:
9      double real;
10     double imag;};
11
12 int main( )
13 {Complex c1(3,4),c2(5,-10),c3;
14  double d;
15  d=2.5+c1;
16  cout<<d<<endl;
17  return 0;}

```

//类型转换函数

C:\WINDOWS\system32\cmd.exe

5.5

请按任意键继续. . .

//要求将一个double数据与Complex类数据相加

2.5+double (c1):自动转换规则，类型转换重载

例. 包含转换构造函数、运算符重载函数

```

1  #include <iostream>
2  using namespace std;
3  class Complex
4  {public:
5      Complex(){real=0;imag=0;}           //默认构造函数
6      Complex(double r){real=r;imag=0;}    //转换构造函数
7      Complex(double r,double i){real=r;imag=i;} //实现初始化的构造函数
8      friend Complex operator + (Complex c1,Complex c2); //重载运算符“+”的友元函数
9      void display();
10 private:
11     double real;
12     double imag;};
13
14  Complex operator + (Complex c1,Complex c2) //定义运算符“+”重载函数
15  {return Complex(c1.real+c2.real, c1.imag+c2.imag);}
16
17  void Complex::display()
18  {cout<<"("<<real<<","<<imag<<"i)"<<endl;}
19
20  int main()
21  {Complex c1(3,4),c2(5,-10),c3;
22   c3=c1+2.5; //复数与double数据相加
23   c3.display();
24   return 0;}

```

operator+(c1,Complex(2.5)):运算符重载调动转换构造函数



```

C:\WINDOWS\system32\cmd.exe
(5.5,4i)
请按任意键继续. . .

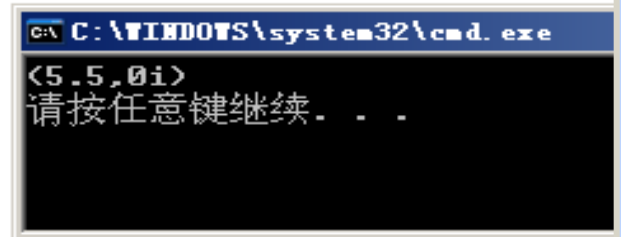
```


问题：在程序中增加类型转换函数，23语句如何执行？

```

1  #include <iostream>
2  using namespace std;
3  class Complex
4  {public:
5      Complex( ){real=0;imag=0;}
6      Complex(double r){real=r;imag=0;} //转换构造函数
7      Complex(double r,double i){real=r;imag=i;}
8      operator double( ){return real;} //类型转换函数
9      //friend Complex operator+ (Complex c1,Complex c2); //重载运算符“+”
10 void display( );
11 private:
12 double real;
13 double imag;};
14
15 //Complex operator + (Complex c1,Complex c2) //定义运算符“+”重载函数
16 //{return Complex(c1.real+c2.real, c1.imag+c2.imag);}
17
18 void Complex::display( )
19 {cout<<"("<<real<<","<<imag<<"i)"<<endl;}
20
21 int main( )
22 {Complex c1(3,4),c2(5,-10),c3;
23  c3=c1+2.5; //复数与double数据相加
24  c3.display( );
25  return 0;}

```



C:\WINDOWS\system32\cmd.exe
<5.5,0i>
请按任意键继续...

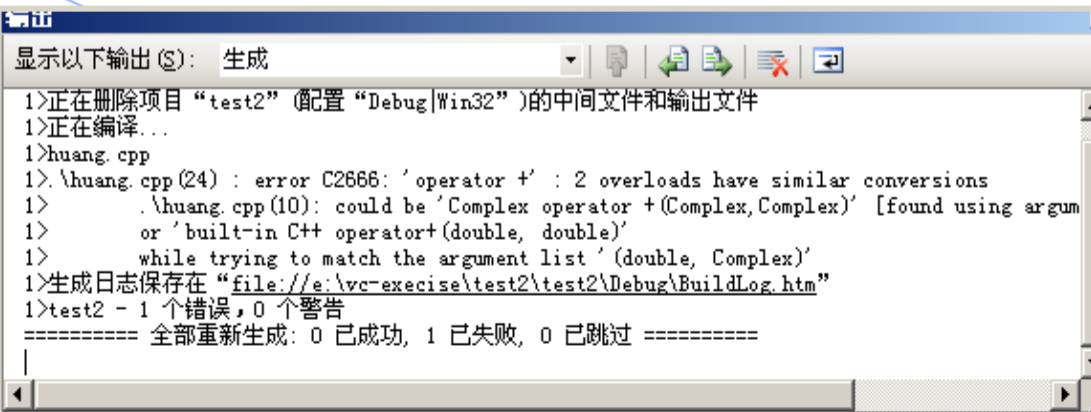
先调用类型转换,再调用构造函数, WHY?

包含转换构造函数、运算符重载函数和类型转换函数

```

1  #include <iostream>
2  using namespace std;
3  class Complex
4  {public:
5      Complex(){real=0;imag=0;}
6      Complex(double r){real=r;imag=0;}
7      Complex(double r,double i){real=r;imag=i;}
8      operator double(){return real;}
9      void display();
10     friend Complex operator + (Complex c1,Complex c2);
11
12     private:
13         double real;
14         double imag;
15     };
16     Complex operator + (Complex c1,Complex c2)
17     {return Complex(c1.real+c2.real, c1.imag+c2.imag);}
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=2.5+c1;
25       c3.display();
26       return 0;
27     }

```



5.2 不同类型数据间的转换

- 类型转换函数可以简化代码编写，利用隐式自动类型转换，将原本多个运算符函数合并成一个：

```
friend Complex operator+(Complex c1, Complex c2);
friend Complex operator+(Complex c1, double d);
friend Complex operator+(Complex c1, int i);
```



```
Complex(double d);
friend Complex operator+(Complex c1, Complex c2);
```

- 但过多地使用隐式的自动类型转换会给程序带来隐患
- 编写程序时应当尽量使用显式类型转换，少用隐式类型转换：

```
c3 = c1 + 2.5;
c4 = 2.5 + c2;
int i = 2;
c5 = 2.5 + i;
```



```
c3 = c1 + Complex(2.5);
c4 = Complex(2.5) + c2;
int i = 2;
c5 = Complex(2
```

虽然有转换构造函数会自动执行类型转换，但还是写成显式类型转换风格更好！

本讲重点

- 对象的引用做函数参数的意义
- 对<<和>>的重载的方法
- 类型转换构造函数：其他类型→本类
- 类型转换重载函数：本类→标准类型
- 根据计算表达式需要，编译系统会自动调用转换构造和重载函数来构造临时对象（变量）
- 结合类型转换函数，一般将双目运算符重载为友元函数，以满足交换率

第5次作业

作业要求同前面，在第7周末交

1. 人事管理的People(人员)类，具有的属性如下：姓名char name[11]、编号char number[7]、性别char sex[3]、生日birthday、身份证号char id[16]等。其中“出生日期”声明为一个“日期”类内嵌子对象。基本要求：(1)用重载运算符《、》实现对人员信息的录入和显示；(2)自己设计构造函数和析构函数、拷贝构造函数、内联成员函数等应用；(3)在程序中声明people类的对象数组，录入数据并显示。(4)可以扩展上述功能。
2. 参考课件中重载数组下标[]和二维数据访问()的实例。编写程序实现如下矩阵计算。要求具有数据越界检测和行列引用标记，例如(i, j)或[i, j]

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad b = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

第5次作业（选做题）

1. 定义一个Teacher类和一个Student类，两者有一部分的数据成员是相同的，例如：num, name和sex. 编写程序将一个Teacher 类对象转为Student类对象，只需将以上3个相同的数据成员移植过去。
2. 阅读教材本章之后“人事管理系统”的代码。分析该代码和前面一章代码的主要区别（增加了哪些新的功能或者知识点）。
3. 阅读“张晋珩同学的五子棋程序”，在其基础上，提出进一步完善意见和实现，根据具体情况可给予奖励分数（1-2分）。