



东南大学吴健雄学院
CHIEN-SHIUNG WU COLLEGE OF SEU

C++程序设计

——类与对象



内容

类与对象的基本概念

构造函数与析构函数

聚合类及其构造函数

运算符重载与友元

静态成员

字符串与字符串类

名字空间域和类域*（自学）



类与对象的基本概念

使用过对象吗？

```
cin >> x; cout << x;
```

```
ch = cin.get();
```

```
ifstream ifile; ifile.open("文件名" );
```

```
string st1, st2; //char st1[40], st2[40];
```

```
st1 = st2; //strcpy(st1, st2);
```

```
st1 = st1 + st2; //strcat(st1,st2);
```

再观察一组：

```
Book b; cout << b; //结构体
```

```
Array a; cin >> a; //数组
```

```
Complex c1, c2, c3; //复数
```

```
c3 = c1 + c2;
```

是否希望支持这样操作？



对象

```
Array a;  cin >> a;
```

```
Complex c1, c2, c3;  c3 = c1 + c2;
```

对象是经过**封装**（定义）的数据, 使用起来就像标准数据一样简单

封装（Encapsulation）是OOP（Object-Oriented Prog.）设计的最基本技术

什么是OOP? ——对应POP（Procedure-Oriented Prog.）

为什么OOP? ——复杂系统难以流程化描述。考虑一个应用程序，
或一个游戏程序

怎么封装对象?



究竟什么是对象？

- 某个数据
- 某个问题
- ...

你所面对的一切一切

怎么描述对象？

- 长什么样
- 能干什么

封装（定义）对象类型

属性（样子）—— 数据

功能（作用）—— 函数

从使用对象的角度看封装需求

以复数对象为例，是否理解以下使用：

```
Complex c1, c2, c3;
```

```
c1.set(3, 5);    //设置c1的值为3+5i
```

```
c2.set(2, -4)    //设置c2为2-4i
```

```
C3 = c1.add(c2); //c1和c2相加
```

```
c3.output();     //输出c3
```

```
double x, y;
```

```
c3.get(x, y);    //获取c3的实部和虚部
```

如何支持复数这样的使用？

封装复数类（新类型，物以类同）

```
class Complex
{    double re, im;    //样子
public:    //功能
    void set(double a, double b);
    void get(double &a, double &b);
    Complex add(Complex c);
    void output();
};    //分号不可少
```



复数类的完整封装

```
class Complex
{
    double re, im;
public:
    void set(double a, double b)
    {   re = a;   im = b; }
    void get(double &a, double &b)
    {   a = re;   b = im; }
    Complex add(Complex c);
    void output();
};    //分号不可少
```

成员函数可以在类内声明，类外定义

```
Complex Complex:: add(Complex c)
{   Complex s;
    s.re = re + c.re;   s.im = im + c.im;
    return s;
}

void Complex:: output()
{
    cout << re << ' + ' << im << ' i ' ;
}
```



语法问题

access specifier (访问权限) :

- public: 在类之外可以访问
- private: 缺省, 在类之外不能访问
- protected: 介于公有和私有之间前两者之间, 派生类可访问



Q&A

1. 类和对象是什么关系？
2. 复数对象使用的例子中，“面向对象”体现在何处？
3. 结构体和类，有什么异同？
4. 两个复数相加这一功能，定义为一般函数和定义为类成员函数，在函数形式上有何差别？
5. 数据成员可否放在 public 当中？
6. get 函数的作用是什么？跟 output 函数区别是什么？



再说封装

1. 封装的2层含义

- 类 = 数据 + 操作
- 对数据的私有化限制

2. 发挥接口 (interface) 作用的成员及其必要性 (如get、set)

- 因数据成员私有，须通过接口函数，间接访问数据成员
- 数据私有的意义：简化和安全（对象属性与外界隔离，如钟表）

3. 封装类（对象）时如何入手

- 通用类 —— 应完备、周全、通用，如字符串、复数
- 非通用类 —— 满足特定需求即可，如为信息管理设计对象



封装类的例子——超市商品信息管理

```
Cgoods cgoods1;           //某种商品
int amount;
double price, totalValue;
cgoods1.Register( "toothpast_ZhongHua", 1000, 6.5);  //设置某种商品基本信息
cgoods1.CountTotal();    //计算该商品总价
Cgoods1.display();       //输出该商品完整信息
price = Cgoods1.getPrice(); //读出单价
amount = Cgoods1.getAmount(); //读出数量
totalValue = Cgoods1.getTotal(); //读出总价
```



```
class Cgoods
```

```
{
```

```
    char Name[40];
```

```
    int  Amount;
```

```
    float Price;
```

```
    float Total_value;
```

```
public :
```

```
    void RegisterGoods(char[], int, float); //修改数据, 接口
```

```
    void CountTotal(); //计算总价
```

```
    int  getAmount(); //读商品数量, 接口
```

```
    float getPrice(); //读商品单价, 接口
```

```
    float getTotal (); //读商品总价, 接口
```

```
    void display(); //输出商品信息
```

```
};
```

```
//定义类
```

```
//或string Name, 商品名
```

```
//商品数量
```

```
//单价
```

```
//总价
```

// 在类外定义部分成员函数

```
void Cgoods :: RegisterGoods(char name[], int m, float p)
```

```
{    strcpy(Name, strlen(name)+1, name);
```

```
    Amount = m;
```

```
    Price = p;
```

```
}
```

```
void Cgoods :: CountTotal()
```

```
{
```

```
    Total_value = Amount * Price;
```

```
}
```

```
int Cgoods :: getAmount()
```

```
{
```

```
    return Amount;
```

```
}
```

```
cgoods1.Register( "toothpast_ZhongHua" , 1000, 6.5);
```

```
cgoods1.CountTotal();
```

```
amount = Cgoods1.getAmount();
```



再看一例：复数类

```
class Complex
{
    double re, im;
public:
    void set(double, double);           //修改, 接口
    void get(double &, double &);      //读, 接口
    Complex add(Complex);              //加法
    void output();                     //输出
};   //分号不可少
```

```
void Complex::set(double r, double i)
{
    re = r; im = i;
}

void Complex::get(double &r, double &i)
{
    r = re; i = im;
}

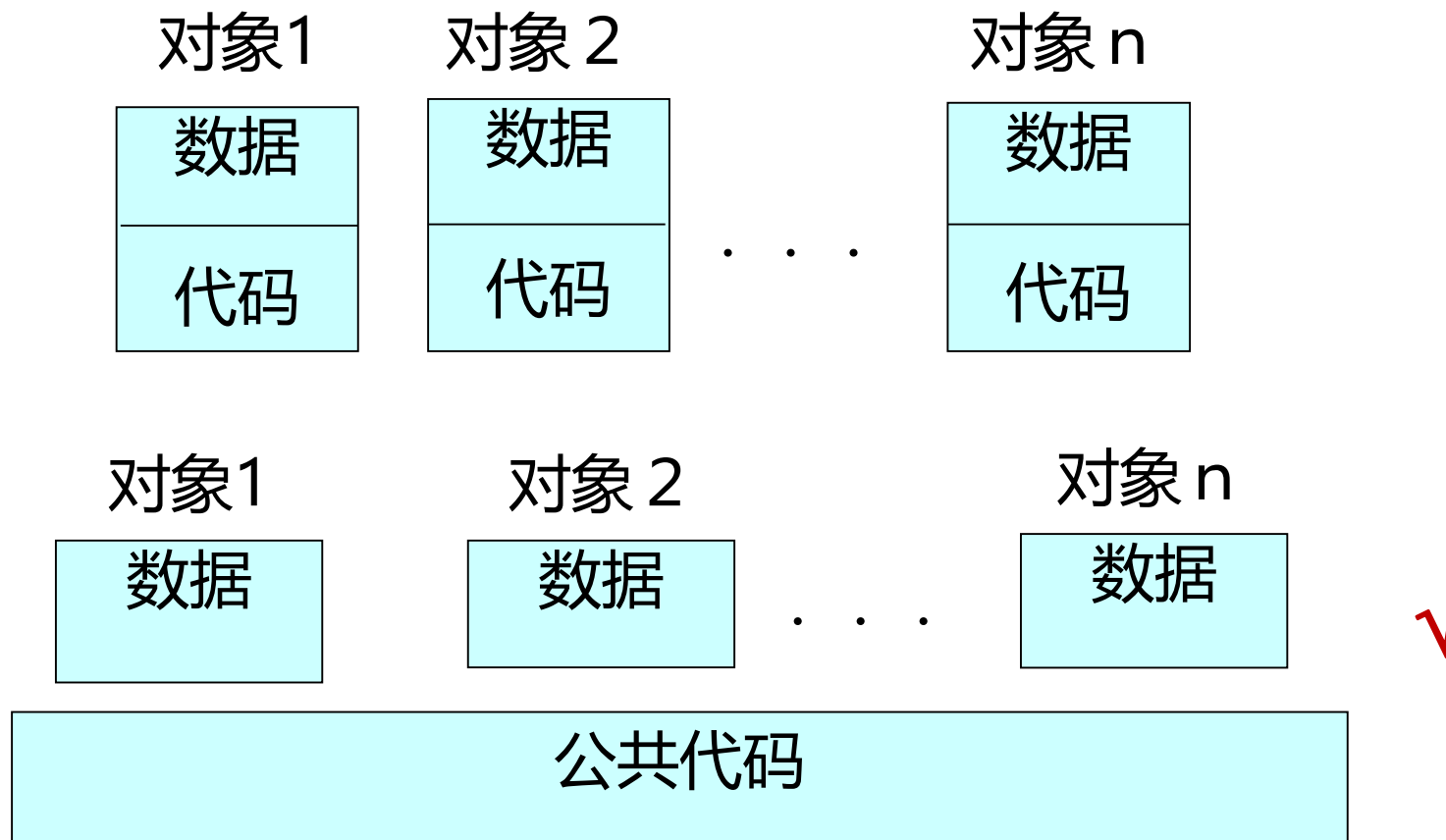
Complex Complex::add(Complex c)
{
    Complex s;
    s.re = re + c.re; s.im = im + c.im;
    return s;
}

void Complex::output()
{
    cout << re << '+' << im << 'i' ;
}
```

复数对象的使用:

```
Complex c1, c2, c3;
c1.set(2, 4); c2.set(3, -5);
c3 = c1.add(c2);
double a, b; c3.get(a,b);
```

对象的存储形式





构造函数与析构函数

定义类时，会自动产生一个缺省的构造函数和析构函数

```
class Complex
{
    double re, im;
public:
    Complex() {};           //构造函数
    ~Complex() {};         //析构函数
    void get(double &a, double &b);
    void set(double a, double b);
    Complex add(Complex c);
    void output();
};
```

特殊在哪里？ 为什么提它？

做什么用？

什么时候用？



构造/析构函数做什么用

构造函数：**创建对象**，获得对象空间

析构函数：**撤销对象**所占空间

```
int main()  
{  
    Complex c1, c2, c3;           //3次调用构造函数  
    return 0;                     //3次调用析构函数  
}
```

构造/析构函数什么时候用？

创建对象时**自动调用**构造函数，对象生命期结束时**自动调用**析构函数



像是隐形人，为什么提它？

创建对象时还有**其他需要**，例如初始化对象；或者，
缺省析构函数不能达成撤销对象的**全部要求**；
——就需要**显式定义**构造/析构函数



利用构造初始化对象

Complex :: Complex(double r, double i) //带参构造

{ re = r; im = i; }

能否支持创建对象：Complex c1(2, 3), c2(1, 6), c3(5), Complex c4;

如何应对各种初始化要求？

Complex c1(1,2), c2, c3(4);

——重载构造函数 **注意：定义带参构造则会覆盖缺省构造函数**

——带缺省参数值的构造函数（万金油）



```
class Complex
```

```
{
```

```
    double re,im;
```

```
public:           //重载应对各种初始化
```

```
    Complex() { };    //必须有
```

```
    Complex(double r, double i)
```

```
    {   re = r;   im = i;   }
```

```
    Complex(double r)
```

```
    {   re = r;   im = 0;   }
```

```
    //其他成员略
```

```
};
```

```
Complex c1, c2(3, 4), c3(4);
```

//带缺省参数值的构造，以一当百

```
Complex(double r = 0, double i = 0)
```

```
{   re = r;   im = i;   }
```

有一种形式的初始化带缺省参数值的构造函数无法支持

拷贝构造函数——拷贝初始化

```
Complex c1,c2(1,2), c3(c2);
```

用对象初始化对象（对象拷贝），由另一个缺省构造函数支持

```
Complex :: Complex(Complex & c)    //拷贝构造函数，必须是引用形参  
{ re = c.re; im = c.im; }
```

缺省拷贝构造函数通常不需显式定义。



到底封装类的时候要不要定义构造和析构？何时需要定义？

- 一般应定义构造函数（重载或带缺省参数值），支持对象初始化；
- 一般不需定义拷贝构造函数和析构函数，除非类中含有指针成员。



聚合类及其构造函数

聚合类：含有对象成员 —— 用对象构造新对象的方法

例：用点构造线段

```
class Point
{
    double x,y;
public:
    Point(double = 0,double = 0);
    void Set(double, double);
    void Get(double &, double &);
    double dis(Point d);
    double dis(double x, double y);
    void show();
};
```

```
class Segment
{
    Point d1,d2;
    double len;           //非必须成员
public:
    Segment(double = 0, double = 0 ,double = 0, double = 0);
    void SetD1(double, double);           //设置端点1值
    void SetD2(double, double);           //设置端点2值
    void GetD1(double &, double &);       //获取端点1值
    void GetD2(double &, double &);       //获取端点1值
    double calLen();                       //计算线段长度
    double GetLen();                       //获取长度
    void show();
};
```



```
Point :: Point (double a, double b)
{   x = a; y = b; }
void Point :: Set(double a, double b)
{   x = a; y = b; }
void Point :: Get(double & a, double & b)
{   a = x; b = y; }
double Point :: dis( Point d)
{   double distance;
    distance = (x - d.x) * (x - d.x) +
               (y - d.y) * (y - d.y);
    return sqrt(distance);
}
double Point :: dis(double a, double b)
{   double distance;
    distance = (x - a) * (x - a) +
               (y - b) * (y - b);
    return sqrt(distance);
}
void Point :: show()
{   cout << '(' << x << ", " << y << ')';
```

```
Segment :: Segment(double a1, double b1 ,double a2, double b2)
: d1(a1, b1), d2(a2, b2) {   len = callen();   }
void Segment :: SetD1(double a, double b)
{   d1.Set(a, b); len = callen();   }
void Segment :: SetD2(double a, double b)
{   d2.Set(a, b); len= callen();   }
void Segment :: GetD1(double & a, double & b)
{   d1.Get(a, b);   }
void Segment :: GetD2(double & a, double & b)
{   d2.Get(a, b);   }
double Segment :: callen()
{   return (d1.dis(d2));   }
double Segment :: GetLen()
{   return len;   }
void Segment :: show()
{   d1.show(); d2. show();   cout << ", length = " << len;   }
```


小结与回顾

类和对象（类型和实例变量的关系，物以类聚）；封装（属性和功能，私有成员）；

封装类的要件（构造，接口，其他功能（通用，适用））；

封装类的形式（声明，定义，作用域运算符）；

构造/析构函数（特殊性，何时、如何被调用，哪些情况下需显示定义）；

聚合类（概念，构造函数特殊性）。

“面向对象” 的进一步理解（成员函数的参数个数，**本对象**的存在）

封装类时有一个**缺省成员**：指向本对象的指针this；

```
void Complex :: set(double re, double im)
```

```
{  this->re = re;  this->im = im;  }
```

```
Complex c1;  c1.set(2, 3);
```

```
Class Complex  
{  double re, im;  
    Complex *this;  
    ...  
};
```



运算符重载与友元

为什么重载运算符？

标准数据类型可直接使用运算符（为什么？）

自定义数据类型可否使用那些运算符？

希望将运算符的应用延伸到自定义数据类型，如复数 $c1 + c2$

重载：在自定义的类型中重载（定义）运算符的功能

方式：

- 成员函数
- 友元函数（教材4.6节）

以复数为例，成员重载+，友元重载-

```
class Complex
```

```
{
```

```
    double re, im;
```

```
public:
```

```
    //其他成员略
```

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

```
    friend Complex operator - (Complex, Complex);
```

```
};
```

c2 + c3的解读: c2.operator+ (c3);

c2 - c1的解读: operator- (c2, c1);

Complex add (Complex);

c3 = c1.add(c2);

//用成员重载 +

//用友元重载 -

两种重载方式形式有何不同?

```
Complex Complex :: operaor +(Complex c)
```

```
{ Complex s; s.re = re + c.re; s.im = im + c.im; return s; }
```

```
Complex operaor - (Complex c1, Complex c2)
```

```
{ Complex d; d.re = c1.re - c2.re; d.im = c1.im - c2.im; return d; }
```



运语法问题：成员函数的几种参数形式

Complex **Complex** :: operator+(Complex c);

Complex **Complex** :: operator+(Complex & c); //有必要使用引用形参吗?

Complex **Complex** :: operator+(const Complex & c); //只读形参

此处引用形参**只为**提高空间效率

只读形参为了提高参数安全性

简单了解友元

- 友元声明在类中, 用friend修饰;
- 友元**不是成员**, 不受访问权限限制, 可定义在类的任意部分;
- 友元提供了特权, 可在类外以**对象.成员**的方式访问类中私有成员;
- 友元对封装有一定“破坏”, 应用并不很多, 运算符重载是其中一个;
- 友元类: A类的所有成员都是B类的友元, A类就是B类的友元类 (见教材)

```
class Complex
{
    double re, im;
public:
    //其他成员略
    friend Complex operator - (const
        Complex &, const Complex &);
};

Complex operaor - (const Complex & c1,
    const Complex & c2)
{
    Complex d;
    d.re = c1.re - c2.re;
    d.im = c1.im - c2.im;
    return d;
}
```

几个稍有特殊的运算符重载—— =、 ++、 >>/<<、 类型转换

= 运算符 —— 系统有缺省定义，用于支持 `c2 = c1;`

`Complex & Complex::operator = (const Complex & c)`

```
{ re = c.re; im = c.im; return *this; }
```

- 只能用成员函数重载，不能用友元重载; 但 += 可以用友元重载
- 通常不需重载 =，除非含有指针成员

++ 运算符 —— 运算符前置与后置的区分: ++c1、c1++

成员重载

Complex Complex :: operator++() //前置

Complex Complex :: operator++(int) //后置

友元重载

Complex operator++(Complex & c) //前置

Complex operator++(Complex & c, int) //后置

此处int不是类型说明，只是一个标记，以示区别

成员重载++前置的代码

```
Complex Complex :: operator++()
{
    re++; im++;
    Complex c(re, im);
    return c;
}
```

成员重载++后置的代码

```
Complex Complex :: operator++(int)
{
    Complex c(re, im);
    re++; im++;
    return c;
}
```

```
{
    re++; im++;
    return *this;
}
```

```
{
    Complex t = *this;
    re++; im++;
    return t;
}
```

如何用友元重载?



<<和>>运算符 —— 实现对象的输入输出 (cha9)

支持: `cin >> c1 >> c2; cout << c1 << " " << c2 << endl;`

```
class Complex
{   double re, im;
public:
```

//其它成员略

```
friend istream& operator >>(istream &is, complex &c)
```

//重载 >>

```
{   is >> c.re >> c.im;   return is; }
```

```
friend ostream& operator <<(ostream &os, const complex &c)
```

//重载 <<

```
{   os << c.re << "+i" << c.im;   return os; }
```

```
};
```

只能友元重载，为什么？

类型转换——形式特殊，且只能用成员重载

```
Complex :: operator double()      //复数转变为实数  
{ return ( sqrt (re*re + im*im) ); }
```

几点说明

绝大多数运算符，既可用成员重载，也可用友元重载
一个运算符，任一种方式重载一次即可



静态成员

由关键字static 修饰说明的成员

静态成员为所有对象共享，只有一份存于公用内存中

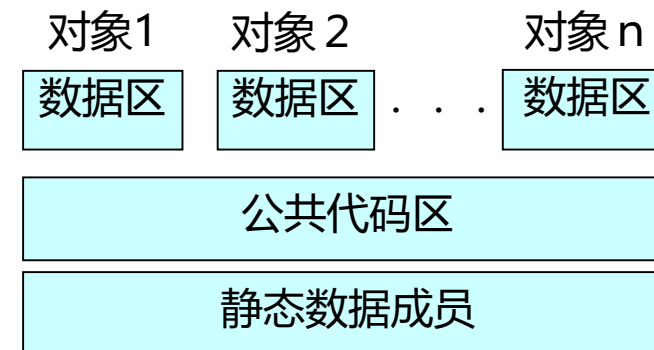


静态数据成员

```
class Ctest{  
    static int count;    //私有成员  
public:  
    Ctest()  
    {    count++;  
        cout << "对象数=" << count << '\n';  
    }  
    ~Ctest()  
    {    count--;  
        cout << "对象数=" << count << '\n';  
    }  
};  
int Ctest :: count = 0;    //静态成员要单独定义
```

```
int main(void)
```

```
{  
    Ctest a[3];  
    return 0;  
}
```



C风格字符串与字符串类

C风格字符串：字符数组，常常用指针操作

```
char str[40] = "hi" ;  
char *pstr1 = "hello"; //pstr1指向常量串  
char *pstr2 = str;      //pstr2指向str  
char *pstr3;            //pstr1和pstr3潜在危险
```

注意pstr1、 pstr2、 pstr3各自的状态

```
cin >> pstr1 >> pstr2 >> pstr3;
```

这些输入会出错吗？

常用字符串操作库函数： **#include<cstring>**

```
char *strcpy(char *cs, const char * ct);    //串拷贝
```

```
char *strcpy_s(char *cs, int len, const char * ct);
```

```
char *strcat(char *cs, const char *ct);      //串拼接
```

```
char *strca_s(char *cs, int len, const char *ct);
```

```
int strcmp(const char *cs, const char *ct);   //串比较
```

```
int strlen(const char *cs);                  //求串长
```



C++还支持串对象

```
string str1;  
cin >> str1; cout << str1 << endl;  
string str2("OK");    //带参构造  
string str3 (str2);    //拷贝构造  
str1 = str2;           //串复制  
str1 += str2;          //串拼接  
str1 = str2 + str3;    //串拼接  
if(str2 > str3) cout << ">"; //串比较  
char ch = str[i];  
//重载下标运算符, 有边界检查
```

C++的string类——定义在头文件string中

3个构造函数：缺省构造、参数构造、拷贝构造；

运算符重载：=、+、+=、==、!=、>、<、<=、>=、<<、>>、[];

友元操作：getline(cin, str, ch); //输入以ch结束
getline(cin, str); //输入以回车结束

其他的串操作：str.length(); str.empty();
str.substr(pos, length1); //返回子串
str1.insert(pos, str2) ;
str2.remove(pos, length1);
str2.find(str1, pos);
str2.find(str1);

【例5.12】——利用string类，判断字符串是否为回文。

如:Madam,I'm Adam. No x in Nixon.

```
string s;  
cin >> s;  
if (is_pal(s)) ...;  
else ...;  
  
bool is_pal(string str)  
{  
    string str2;  
    str2 = remove_punct(str, ",.\\'"); //去标点  
    str2 = to_lower(str2);             //变小写  
    if(str2 == reverse(str2))          //判对称  
        return true;  
    else return false;  
}  
  
string to_lower(string s) { //代码略; }  
string reverse(string s) { //代码略; }
```



string remove_punct(string s, string punct)

```
{    string no_punct;    //不含标点的串，初始化为空串
    for(int i = 0; i < s.length(); i++)
    {    string s_ch = s.substr(i, 1);    //逐个提取被判串中单字符子串
        int pos = punct.find(s_ch);    //子串是否在标点串中
        if(pos < 0 || pos >= punct.length()) no_punct += s_ch;
    }
    return no_punct;
}
```



理解string类——封装myString类

字符串如何表达 —— 数据成员

支持哪些功能和运算 —— 函数成员