计算机程序设计语言（VC++）

第 6 章

类和对象

张晓如，华伟

《C++程序设计基础教程》
人民邮电出版社，2018.05

本章内容

6.1 面向对象程序设计

- 面向过程的程序设计以数据为中心。
  - 用数据描述问题，用函数解决问题；
  - 数据和函数是独立的。
- 面向对象的程序设计以事件为中心。
  - 用类描述问题，用对象解决问题；
  - 事件 = 数据 + 数据处理（函数）。

- 继承（派生）性
- 多态性

- 封装（隐藏）性

【例 6-1】计算矩形的周长和面积。

- 矩形的边长分别为 a 和 b 。
- 函数 circum 求周长。
- 函数 area 求面积。
- 用面向过程和面向对象的方法实现程序。

6.1 面向对象程序设计

【源程序代码】（面向过程的程序设计）

```
int circum(int a, int b) { // 求周长
    return 2*(a+b);
}
int area(int a, int b){    // 求面积
    return a*b;
}
int main( ){
    int a1, b1;
    cout<<" 请输入边长： "; cin>>a1>>b1;
    cout<<" 矩形周长： "<<circum(a1, b1)<<endl;
    cout<<" 矩形面积： "<<area(a1, b1)<<endl;
    return 0;
}
```

数据及其处理
没有必然联系

程序运行结果
请输入边长： 3 5
矩形周长： 16
矩形面积： 15

程序运行结果
请输入边长： -3
5
矩形周长： 4
矩形面积： -15

6.1 面向对象程序设计

【源程序代码】（面向对象的程序设计）

- 类定义
  - 封装数据及其处理
  - 隐藏成员

```
class REC{  int a, b;
public:
    REC(int t1, int t2){a=t1; b=t2;}
    int circum() {return 2*(a+b); } //  周长
    int area(){return a*b;} //  面积
};
int main( ){  int a1, b1;
    cout<<" 请输入边长 :"; cin>>a1>>b1;
    REC r1(a1, b1);
    cout<<" 矩形周长 :  "<<r1.circum( )<<endl;
    cout<<" 矩形面积 :  "<<r1.area( )<<endl;
    return 0;
}
```

程序运行结果

```
请输入边长 :  3  5
矩形周长 :  16
矩形面积 :  15
```

6.1 面向对象程序设计

提取必要信息

- **面向对象程序设计**的基础
  - **类**：对某一类问题的**抽象**描述；
  - **对象**：对象是实例，表示具体的问题。

类：抽象概念	对象：实例
普遍性	特殊性
数据类型	变量

矩形：	
边长（ a 、 b ）	a=3 b=5
周长（ c ）	c=16
面积（ s ）	s=15

6.2 类

类的定义的基本格式

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

- 定义类的关键字是 class 。
- 类名为合法标识符。
- 成员在类体“{}”中说明或定义，语句结束符“;”不能少。
- 访问权限
  - 公有成员 public ；
  - 保护成员 protected ；
  - 私有成员 private ；
  - 缺省权限 private 。

6.2 类

```
class A{
    int a=0;  // 语法错误
protected:
    int c;
public:
    A (int m, int n) {
        a=m; b=n; c=m+n;
    }
    void print( ) ;
private:
    auto int b;  // 语法错误
};
void A::print( ) { cout<<a<<'\t'<<b<<'\t'<<c<<endl; }
```

- 类 A 中定义了：
  - 私有成员 a 、 b ；
  - 护成员 c ；
  - 公有成员构造和 print 函数。
- 定义类时，访问权限的顺序和次数均没有限制。
- 数据成员不能初始化。
- 除用 static 说明静态成员外，不能用 auto 、 register 、 extern 说明成员的存储类型。

6.2 类

```
class A{
public:
    A ( ) { a=0; }
    void print( ) ;
    int a;
};
void A::print( ) {
    cout<<a<<endl;
}
```

```
int main{
    A t;
    t.a=5;
    t.print();
    }
```

- 成员函数
  - 类中定义：通常为短小函数，具有内联特性；
  - 类中说明，类外定义
    函数类型 函数名 (形参列表);
    函数类型 类名 :: 函数名 (形参列表)
    { } // 函数体
- 类作用域
  - 类中处处可见，无须先定义后使用；
  - 类中说明或定义的成员不能在类（含派生类）外使用；
  - 类外必须通过对象才能使用成员。

6.3 对象

6.3.1 对象定义与使用

1. 定义对象

- 定义对象的基本格式为：
  类名 对象名（实参列表）；
  如（设类 A 已定义）：
  A a1(1,2);
  A a2(2,3), a3(3,4);
  - 对象的实参与构造函数形参一致；
  - 构造函数无参时，对象也不能有参数，如：A t ；
  - 对象的数据成员具有独立的存储空间。
- 此外，还可以在定义类 的同时定义对象，或定义类时不列出类名而直接定义对象。

成员 对象	a	b	c
a1	1	2	3
a2	2	3	5
a3	3	4	7

6.3 对象

2. 使用对象

- 对象的使用是面向成员的，即除整体赋值外，通常不能将对象作为整体使用，而只能引用对象的成员
  - 引用非静态数据成员的基本形式
    对象名 . 数据成员名
  - 引用非静态成员函数的基本形式
    对象名 . 成员函数名（实参列表）

  如：

  ```
  a1.print(); a2.print();
  a3.print;   // 逻辑错误
  cout<<a1; // 语法错误
  ```

不输出 3 4 7

程序运行结果

```
1   2   3
2   3   5
```

- 在友元函数和派生类外，只能直接访问公有成员，如：

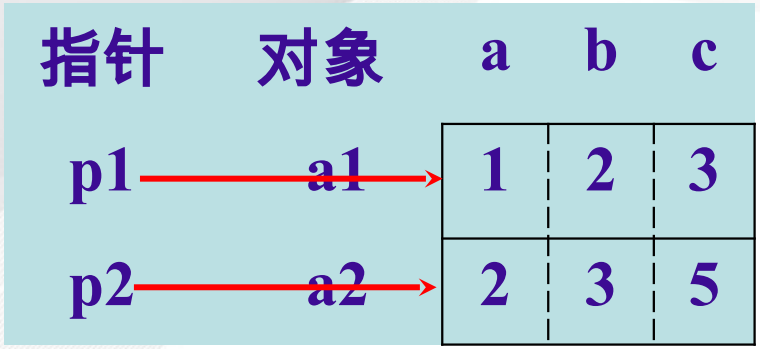
```
cin>>a1.a>>a1.b>>a1.c;  // 语法错误
```

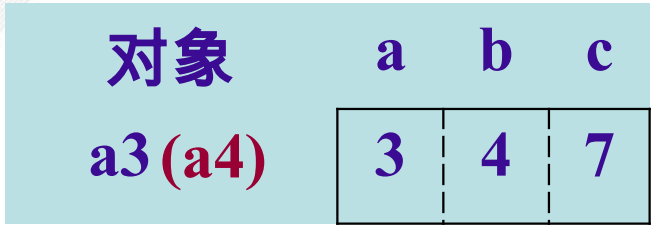
6.3 对象

6.3.2 对象的指针及引用

1. 对象指针

- 定义指向对象的指针格式为：
  类名 * 指针变量名；
  如（设类 A 已定义）：
  A a1(1,2), a2(2,3), *p1=&a1,*p2;
  p2=&a2;





- 通过指针引用成员的基本形式
  - 引用数据成员：指针名 -> 数据成员名
  - 引用成员函数：指针名 -> 成员函数名（实参列表）

2. 对象引用

- 定义引用对象格式：类名 & 引用对象名 = 被引用对象名；
- 引用对象是被引用对象的别名，如： A a3(3,4),&a4=a3 。

6.3 对象

【例 6-2】对象的指针及引用示例。

【源程序代码】

```
class B{
    int a;
public:
    int b;
    B(int t1=0, int t2=0) {
        a=t1; b=t2;
    }
    void print( ) {
        cout<<a<<'\t'<<b<<endl;
    }
};
```

```
int main( ) {
    B b1(1,2),&b2=b1;
    b2.b=3;    b1.print( );
    B *p1=&b1;
    p1->b=5; b2.print( );
    B *p2=new B;
    p2->print( ); (*p2).print( );
    delete p2; // 释放动态对象
    return 0;
}
```

程序运行结果

```
1  3
1  5
0  0
0  0
```

6.3 对象

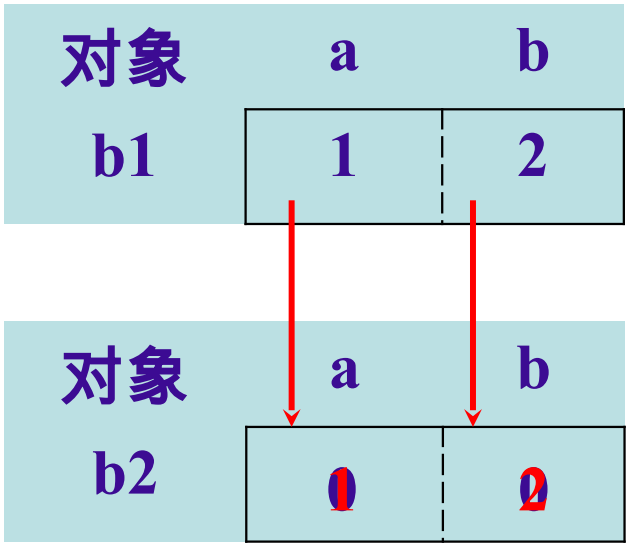
6.3.3 对象赋值

对象作为整体使用，通常仅限于赋值。

- 同类型对象之间相互赋值。
- 派生类对象向基类对象赋值。
- 赋值通过预先重载的赋值运算符实现。
- 将对象的所有数据成员逐个拷贝。

如：

```
B b1(1,2),b2;
b2=b1;  //b2.a=b1.a,b2.b=b1.b;
```



【例 6-3】对象之间的赋值示例。

6.3 对象

数据成员相同的
不同类的对象不可以
相互赋值

【源程序代码】

```
class C{ int a, b, c;
public:
    C(int t1, int t2, int t3){ a=t1; b=t2; c=t3;}
    void print( ){ cout<<a<<'\t'<<b<<'\t'<<c<<endl; }
};
int main( ) {  C c1(1, 2, 3), c2(0, 0, 0);
    cout<<" 赋值前： \n";
    c1.print( );    c2.print( );
    c2=c1;
    cout<<" 赋值后： \n";
    c1.print( );   c2.print( );
    return 0;
}
```

程序运行结果

```
赋值前：
1   2   3
0   0   0
赋值后：
1   2   3
1   2   3
```

6.4 类成员的访问控制

1. 访问权限

- 公有成员可以在任何地方直接访问；
- 保护成员可以在自身类和派生类中直接访问；
- 私有成员只能在自身类和友元函数中直接访问。

2. 访问方式

- 直接访问：类中直接引用成员，类外通过对象直接引用成员。如：a 或 t.a（设 a 为类 A 的成员，t 为类 A 的对象）。
- 间接访问：类中通过公有成员函数间接引用成员，类外通过对象的公有成员函数直接引用成员。 如：get() 或 t.get()（设 get() 为公有成员函数，引用成员 a ）。

【例 6-4】定义一个复数类，实现复数相加。

6.4 类成员的访问控制

【源程序代码】

```
class F{
    int a, b;
public:
    F(int t1=0, int t2=0) {
        a=t1; b=t2;
    }
    void set(int t1, int t2){
        a=t1; b=t2;
    }
    void print( ) {cout<<a<<'+'<<b<<'i'<<endl;}
    int geta( ) { return a; }
    int getb( ) { return b; }
};
```

```
int main( ) {
    F f1(1, 2), f2(3, 4), f3;
    int a1=f1.geta( )+f2.geta( );
    int b1=f1.b+f2.b; // 语法错误
    f3.set(a1,b1);
    f3.print( );
    return 0;
}
```

程序运行结果

```
4+6i
```

6.5 构造函数与析构函数

- 构造函数用于对象初始化（数据成员）赋值。
- 构造函数是**没有函数类型**、函数**名称**与类名**相同**的特殊成员函数。
- 根据参数类型（函数重载），构造函数实现不同对象初始化
  - **普通构造函数**：以基本数据、数组等初始化对象；
  - **默认构造函数**：初始化无参对象；
  - **拷贝构造函数**：用已有对象初始化新对象。
- 一个类至少有两个构造函数：拷贝构造函数、普通构造函数（默认构造函数）。
- 析构函数是**没有函数类型**、函数名称为“~类名”、**没有参数**的特殊成员函数，用于释放对象。
- 析构函数不能重载。
- 构造函数和析构函数由系统自动调用，通常为公有访问权限。

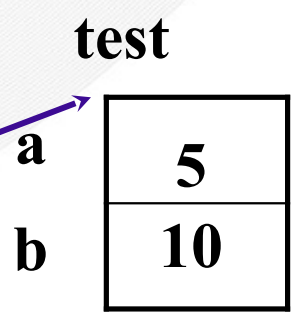
6.5 构造函数与析构函数

6.5.1 普通构造函数

形参	实参
普通变量	基本数据（变量、常量）
指针变量	数组、地址

如（普通变量）：

```
class A{
    int a,b;
public:
    A(int x,int y){
        a=x;b=y;
    }
};
```




```
int main(){
    int n=5;
    A test(n,10);
}
```

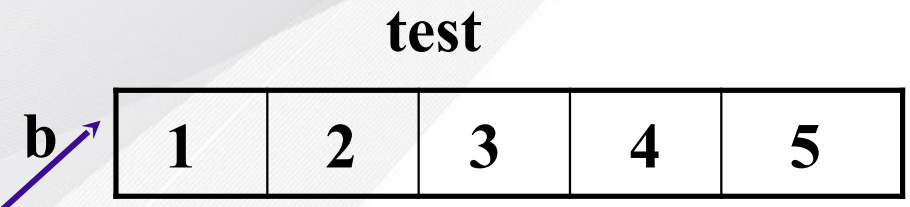
6.5 构造函数与析构函数

6.5.1 普通构造函数

如（指针变量）：

```
class B{
    int b[5];
public:
    B(int *p,int n){
        for(int i=0;i<n;i++)
            b[i]=p[i];
    }
};
```

b=p; // 语法错误




```
int main(){
    int t[5]={1,2,3,4,5};
    B test(t,5);
}
```

- 构造函数没有函数类型，既不是 void ，也不是 int ；
- 构造函数不能用 return 语句返回值。

6.5 构造函数与析构函数

6.5.2 默认构造函数

- 系统自动产生

```
class A{
    int a,b;
public:
    A( ){  }
};
```

- 定义（无参）

```
class B{
    int a,b;
public:
    B(){
        a=0;b=0;
    }
};
```

- 定义（都有默认值）

```
class C{
    int a,b;
public:
    C(int x=0,int y=0){
        a=x;b=y;
    }
};
```

```
int main(){
    A t1; B t2; C t3,t4(1,2);
}
```

	t1	t2	t3	t4
a	无	0	0	1
b	无	0	0	2

6.5 构造函数与析构函数

6.5.2 默认构造函数

- 只有在未定义该类构造函数时才自动产生默认构造函数；
- 不是每个类都有默认的构造函数；
- 一个类最多有一个默认的同类构造函数；
- 定义对象时，必须有形参与对象实参一致的构造函数。

```
class A{
    int a;
public:
    A(int x)
    {
        a=x;
    }
};
```

```
class B{
    int a;
public:
    B(){ a=0; }
    B(int x=1){
        a=x;
    }
};
```

```
int main(){
    A t1;       // 语法错误
    B t2(2);
    B t3;       // 调用时
                // 语法错误
    B t3();     // 函数说明
}
```

6.5 构造函数与析构函数

6.5.3 拷贝构造函数

- 形参为该类型对象引用，实参为对象；
- 通常无需定义，由默认的拷贝构造函数将实参对象的数据成员逐一赋值给新对象；
- 当数据成员为指针（动态内存），或要求新对象与实参对象的数据成员不同时必须定义拷贝构造函数。

```
class A{
    int a,b;
public:
    A(A&t){
        a=t.a;
        b=t.b;
    }
};
```

```
class B{
    int a,b;
public:
    B(B &t){
        a=t.a+2;
        b=t.b*2;
    }
};
```

```
int main(){
    A t2(t1);      // 设 t1(1,2)
    B t4=t3;       // 设 t3(1,2)
}
```

则 t2 的 a 、 b 分别为 1 、 2 ，
t4 的 a 、 b 分别为 2 、 4

- 注意拷贝构造的形式

6.5 构造函数与析构函数

【例 6-5】拷贝构造函数（指针成员）示例。

【源程序代码】

```
class K{ char *s;
public:
    K(char *p) {
        int n=strlen(p);
        s=new char[n+1];
        strcpy(s,p);
    }
    K(K &t) ;
    ~K( ) { delete [ ]s; }
    void print( ){cout<<s<<endl;}
};
```

6.5 构造函数与析构函数

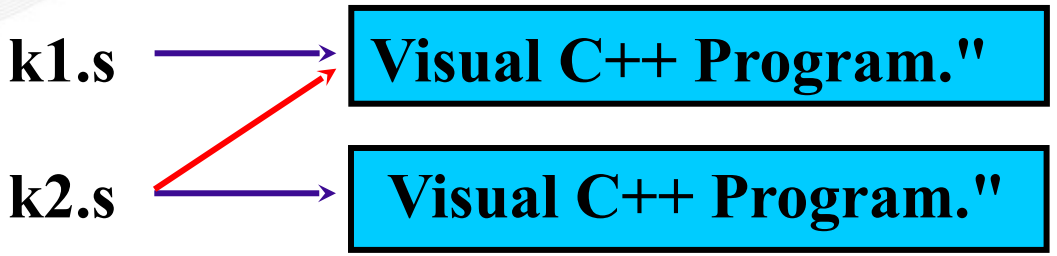
【源程序代码】

```
K::K(K &t) {
    int n=strlen(t.s);
    s=new char[n+1];
    strcpy(s,t.s);
}
```

```
K::K(K &t) {
    s=t.s;;
}
```

内存引用错误

```
int main( ) {
    K k1("Visual C++ Program.");
    k1.print( );
    K k2(k1);
    k2.print( );
    return 0;
}
```



6.5 构造函数与析构函数

6.5.4 构造函数与成员初始化列表

- 当类的数据成员是构造类型时，通常不能在函数体中赋值；
- 常量、引用和对象等特殊成员在头部用列表初始化。

```
class A{
    int a;
public:
    A(int x){
        a=x;
    }
};
```

```
class B{
    int b,&c;
    const int d;
    A a1;
public:
    B(int,int,int);
};
```

```
B::B(int x,int y,int z){
    b=x;
    c=b;      // 语法错误
    d=z;      // 语法错误
    a1=t;     // 语法错误
    a1.a=t;   // 语法错误
}
```

```
B::B(int x,int y,int z,int t):c(b),d(y),a1(z){
    b=x;
}
```

6.5 构造函数与析构函数

【例 6-6】成员初始化

成员初始化顺序：
- 先头部列表，后函数体；
- 列表按成员说明顺序初始化。

【源程序代码】

```
class M{  int a;
public:
      M(int
      voi
};
class N{
    int a,b;  const int c;
    int &d;  M m1;
public:
      N(int t):m1(++t),d(b),c(++t),a(++t) { b=2*t; }
      void print( ) ;
};
```

成员说明顺序决定列表中成员初始化顺序

列表决定成员的值，与初始化顺序无关

6.5 构造函数与析构函数

【例 6-6】成员初始化列表的使用示例。

【源程序代码】

```
void N::print( ) {
    cout<<"m1.a =";m1.show( );
    cout<<"a = "<<a<<endl;
    cout<<"b = "<<b<<endl;
    cout<<"c = "<<c<<endl;
    cout<<"d = "<<d<<endl;
}
int main( ) {
    N n1(1);
    n1.print( ); return 0;
}
```

cout<<m1;
cout<<m1.a;
?

程序运行结果

```
m1.a = 4
a = 2
b = 8
c = 3
d = 8
```

6.5 构造函数与析构函数

6.5.5 析构造函数

- 析构函数通常由系统自动产生，自动调用；
- 数据成员使用了动态内存时，必须定义析构函数将其释放。

【例 6-7】析构函数使用示例。

【源程序代码】

```
class G{
    char *s;
public:
G(char *p);
    ~G( );
    void print( ) { cout<<s<<endl; }
};
```

6.5 构造函数与析构函数

【源程序代码】

```
G::G(char *p){
    s=new char[strlen(p)+1];  // 分配动态内存
    strcpy(s,p);
    cout<<" 调用了构造函数 \n";
}
G::~G( ){
    delete [ ]s;              // 释放动态内存
    cout<<" 调用了析构函数 \n";
}
int main( ) {
    G g1("Visual C++ Program.");
    g1.print( );   return 0;
}
```

程序运行结果

```
调用了构造函数
Visual C++
  Program.
调用了析构函数
```

6.6 this 指针

- 非静态成员函数中指向自身（当前对象，即正在调用成员函数的对象）的指针

```
class Q{
public:
    int a;
    void f( )
    {
        cout<<this->a;
    }
};
```

this

```
int main(){
    Q q1,q2;
    q1.a=5,q2.a=10;
    q1.f();   // 当前对象为 q1
    q2.f();   // 当前对象为 q2
    return 0;
}
```

- 系统自动产生，隐式使用的常量指针；
- 可以显式使用。

【例 6-8】 this 指针使用示例。

6.6 this 指针

【源程序代码】

```
class Q{  int a, b;
public:
    Q(int t, int b){
        a=t; this->b=b;
    }
    void print( ){
        cout<<a<<'\t'<<b<<endl;
    }
    Q add(int a, Q &t){
        this->a=this->a+a+t.a;
        b=b+a+t.b;
        return *this ;
    }
};
```

```
int main( ) {
    Q q1(1,2), q2(0,0);
    q1.print( ); q2.print( );
    q1= q2 . add(5,  q1 );
    q1.print( ); q2.print( );
    return 0;
}
```

this → q2

this->a=this->a+a+t.a; → q2.a=q2.a+a+q1.a;

return *this ; → q2

程序运行结果

```
1    2
0    0
6    7
6    7
```

6.7 静态成员

- 静态成员实现类的不同对象之间的成员共享。

6.7.1 静态数据成员

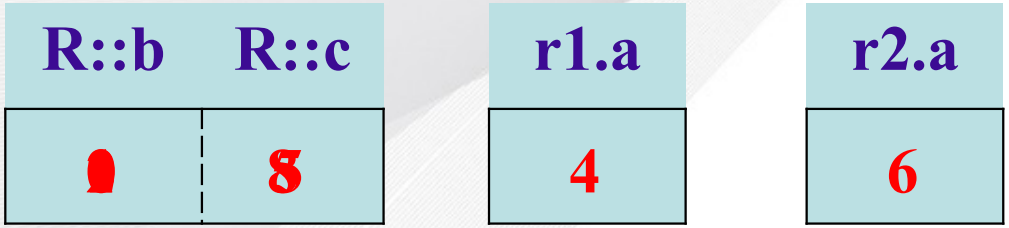
- 类的静态数据成员不属于特定的对象，而是属于类；
- 静态数据成员必须类中说明、类外定义
  - static 数据类型 成员名； // 类中说明
  - 数据类型 类名 :: 成员名 = 初始值； // 类外定义

或

  - 数据类型 类名 :: 成员名 (初始值); // 类外定义
- 静态数据成员具有默认的初始值 0 ；
- 静态数据成员使用
  - 类中：类名 :: 成员名、成员名（ this-> 成员名 ）；
  - 类外：类名 :: 成员名、对象名 . 成员名 。

6.7 静态成员

```
class R{
public:
    int a;
    static int b;
    static int c;
    R(int n)
    {
        a=n;
        b++;
    }
};
int R::b;
int R::c=5;
```




```
int main(){
    cout<<R::a;    // 语法错误
    cout<<R::b;    // 输出 0
    R r1(4),r2(6);
    cout<<r1.a<<r1.b<<r1.c;    // 输出 425
    r1.c=8;
    cout<<r2.a<<r2.b<<R::c;    // 输出 628
    return 0;
}
```

6.7 静态成员

【例 6-9】类的静态数据成员的定义与使用示例。

【源程序代码】

```
class R{
    int a;
    static int b, c; // 类中说明
public:
    R(int t){ a=t; }
    void add( ){  a++; b++; c++;}
    void print( ){
        cout<<a<<'\t'<<b<<'\t'<<R::c<<endl;
    }
};
int R::b, R::c(5); // 类外定义
```

```
int main( ) {
    R r1(0), r2(3);
    r1.print( ); r2.print( );
    r1.add( );
    r1.print( ); r2.print( );
    return 0;
}
```

程序运行结果

```
0    0    5
3    0    5
1    1    6
3    1    6
```

6.7 静态成员

6.7.2 静态成员函数

- 用关键字 static 说明的成员函数，没有 this 指针；
- 类中直接定义

  static 函数类型 函数名 (形参列表){ }
- 类中说明、类外定义
  - static 函数类型 函数名 (形参列表); // 类中说明
  - 函数类型 类名 :: 函数名 (形参列表)){ } // 类外定义
- 静态函数中使用成员
  - 静态成员：成员名，即类名 :: 成员名；
  - 非静态成员：等同类外使用，如对象名 . 成员名 。
- 类外使用静态成员函数，与使用静态数据成员相似，如类名 :: 静态成员函数名 (实参表)、对象名 . 静态成员函数名 (实参表)。

6.7 静态成员

【例 6-10】类的静态成员函数定义与使用示例。

【源程序代码】

```
class S{
    int a;
    static int b,c;
public:
    S(int t) { a=t; }
    static void add(  S &t  ){
        t.a++;//  语法错误
        b++;  S::c++;
    }
    void print( ){cout<<a<<'\t'<<b <<'\t'<<c<<endl;}
};
int S::b=5, S::c=10;
```

```
int main( ) {
    S s1(0);        s1.print( );
    s1.add(s1);  //S::add(s1);
    s1.print( );
    return 0;
}
```

程序运行结果

```
0    5    10
1    6    11
```

6.8 常成员与常对象

6.8.1 常成员

类的常成员包括**常数据成员**和**常成员函数**。

1. 常数据成员

- 常数据成员的概念

  值为**常量**的数据成员。

- 常数据成员的定义

  const 数据类型 数据成员名；

  数据类型 const 数据成员名；

- 常数据成员的初始化

  只能在构造函数**头部列表初始化**。

- 常数据成员的使用

  只能**读**不能**写**。

6.8 常成员与常对象

【例】常数据成员

```
class CLASS{
    const int a;
    int b,c;
public:
    CLASS(int t1,int t2,int t3) :a(t1),b(t2){ c=t3; }
    void set(int t1,int t2,int t3)
    {
        a=t1;        // 语法错误
        b=t2,c=t3;
    }
    void print( ){cout<<a<<'\t'<<b <<'\t'<<c<<endl;}
};
```

6.8 常成员与常对象

6.8.1 常成员

2. 常成员函数

- 常成员函数的概念

  只能用来读数据成员，不能用来写数据成员的成员函数。

- 常成员函数的定义
  - 定义格式

  函数类型 成员函数名（形参列表） const ；
  - 定义时注意： const 应置于形参列表的后面，而不能置于函数类型或函数名的前面。
- 常成员函数的使用

  常成员函数与非常成员函数构成重载关系。

6.8 常成员与常对象

【例】常成员函数

```
class MyClass{
    int a;
public:
    MyClass(int t) :a(t){ }
    void set(int t)const
    {
        a=t;    // 语法错误
    }
    void print( )const{    cout<<a<<endl; }
    void print( )  {    cout<<a*a<<endl;    }
};
```

6.8 常成员与常对象

6.8.2 常对象

- 常对象的概念：数据成员的值不能改变的对象。
- 常对象的定义
  - 定义格式

  类名 const 对象名（实参列表）；

  const 类名 对象名（实参列表）；
  - 说明：常对象的数据成员不一定是常成员 。
- 常对象与常成员函数的使用
  - 非常对象可以调用常成员函数和非常成员函数；
  - 常对象只能调用常成员函数，不能调用非常成员函数；
  - 重载时，常对象调用常成员函数，非常对象调用非常成员函数。

6.8 常成员与常对象

【例】常对象

```
class MyClass{     int a;     const int b;
public:
     MyClass(int t1,int t2) :a(t1),b(t2){  }
     void set(int t)   {     a=t;     }
     void print( )const    { cout<<a+b<<endl; }
};
int main(void)
{    MyClass test1(1,2);  const MyClass test2(3,4);
     test1.set(5);  test1.print();
     test2.set(6);          // 语法错误
     test2.print();
     return 0;
}
```

6.8 常成员与常对象

【例】常对象与常成员函数

```
class MyClass{
    int a,b;
public:
    MyClass(int t1,int t2) :a(t1),b(t2){ }
    void print( )const {   cout<<a<<'\t'<<b<<endl;    }
    void print( ) {cout<<b<<'\t'<<a<<endl; }
};
int main(void)
{   const MyClass test1(1,2);
    MyClass test2(3,4);
    test1.print();  test2.print();
    return 0;
}
```

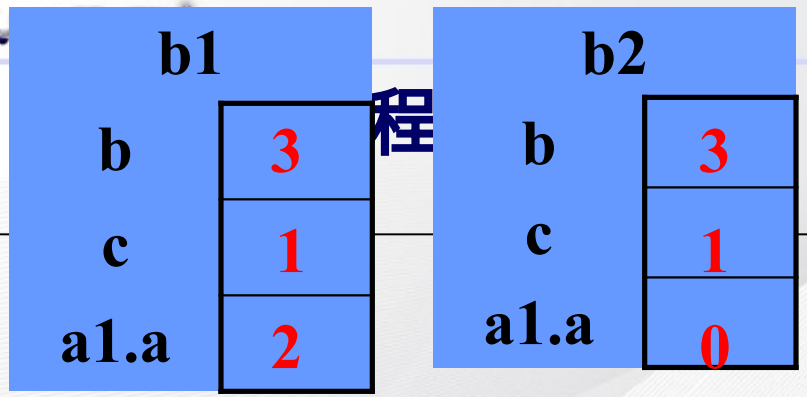
程序运行结果

```
1       2
4       3
```

6.9 程序举例

【例 6-11 】写出下列程序的运行结果（构造函数和析构函数调用）。

- 先产生（调用构造函数）的对象后撤销（调用析构函数），后产生的对象先撤销。
- 调用构造函数
  - 产生无参对象，调用默认构造函数；
  - 以普通参数产生对象，调用普通构造函数；
  - 以已有对象产生新对象，调用拷贝构造函数。
- 调用析构函数
  - 释放空间，包括对象成员占据的内存；
  - 释放对象成员要调用对象的析构函数
- 释放对象时，构造函数与析构函数调用顺序通常相反。



【源程序代码】

```
class A{  int a;
public:
    A() { a=0; cout<<"A()"<<endl; }
    A(int t) { a=t; cout<<"A(int)"<<endl; }
    ~A(){ cout<<a<<','<<"~A()"<<endl; }
    int get( ){ return a; }
};
class B{
    int b,c;
    A a1;
public:
    B(B&t)  : A ( )                 { b=t.b;  c=t.c;  cout<<"B(&)"<<endl; }
    B(int t): a1(t++) , c(t++)      { b=t++; cout<<"B(int)"<<endl; }
    ~B(){ cout<<a1.get()<<','<<b<<','<<c<<','<<"~B()"<<endl; }
};
```

释放 释放 b1

```
int main( ) {
    B b1(1);
    B b2(b1);
    return 0;
}
```

程序运行结果

```
A(int)
B(int)
A()
B(&)
0,3,1,~B()
0,~A()
2,3,1,~B()
2,~A()
```

6.9 程序举例

【例 6-12 】创建一个 Triangle 类，这个类将直角三角形的两条直角边作为私有数据成员，要求设计构造函数及两个成员函数，分别实现初始化数据、求斜边长度以及求三角形面积的功能。

- 定义三角形类 Triangle
  - 私有数据成员 a 、 b ，表示两条直角边；
  - 构造函数初始化数据成员：
    Triangle(float x,float y);
  - 成员函数 f1 求斜边长度：
    float f1();
  - 成员函数 f2 求面积：
    float f2();
- 主函数输入数据初始化三角形，并输出其斜边和面积。

6.9 程序举例

【源程序代码】

```
#include <iostream>
#include <cmath>
using namespace std;
class Triangle{
    float a,b;
public:
    Triangle(float x=0 , float y=0) { a=x; b=y; }
    float f1(){
        return sqrt(a*a+b*b);
    }
    float f2(){
        return a*b/2;
    }
};
```

6.9 程序举例

【源程序代码】

```
int main( ){
    float s,t;
    cout<<" 输入直角三角形的两条直角边： ";
    cin>>s>>t;
    Triangle aa(s,t);
    cout<<" 直角三角形斜边长度为 :"<<aa.f1()<<'\n';
    cout<<" 直角三角形的面积为 :"<<aa.f2()<<'\n';
    return 0;
}
```

程序运行结果

输入直角三角形的两条直角边： 3 4

直角三角形斜边长度为： 5

直角三角形的面积为： 5

6.9 程序举例

【例 6-13 】定义一个数组类，实现将二维数组各行和各列元素排序、全体元素按内存顺序排序等功能。

- 定义数组类 Array
  - 私有数据成员： int a[4][5];
  - 公有构造函数： Array(int t[][5],int n) ；通过循环语句初始化数组
  - 功能函数： void fun1( ); 实现行排序
  - 功能函数： void fun2( ); 实现列排序
  - 功能函数： void fun3( ); 实现内存顺序排序
  - 输出函数： void print( ) ；循环语句输出数据成员
- 主函数测试类
  - 定义测试数据：随机产生的二维数组；
  - 定义对象：实参与构造函数形参一致；
  - 通过对象调用功能函数：注意调用顺序和函数实参。

6.9 程序举例

- 行排序算法
  - 将每行元素作为5个（5列）元素的一维数组（数组名a[i]），采用直接选择排序法对其排序，元素位置为k；
  - 通过循环语句对4个（4行）这样的一维数组排序。
- 列排序算法
  - 将每列元素作为4个（4行）元素一维数组，排序过程中列位置j不变；
  - 通过循环语句对5个（5列）这样的一维数组排序。
- 全部元素排序算法
  - 将二维数组作为20个（4行5列）元素的一维数组；
  - 定义元素指针p指向二维数组的首元素（a[0][0]），上述一维数组即为p。
- 对行排序将影响每列中的元素，通过对象赋值恢复原数组。

6.9 程序举例

【源程序代码】

```
#include<iostream>
#include<cstdlib>
using namespace std;
class Array{
    int a[4][5];
public:
    Array(int t[][5],int n) ;
    void print( ) ;
    void fun1( ); // 行排序
    void fun2( ); // 列排序
    void fun3( ); // 全部排序
};
```

```
Array::Array (int t[][5],int n) {
    for(int i=0; i<n; i++)
        for(int j=0; j<5; j++)
            a[i][j]=t[i][j];
}
```

```
void Array::print ( ) {
    for(int i=0; i<4; i++){
        for(int j=0; j<5; j++)
            cout<<a[i][j]<<'\t';
        cout<<endl;
    }
}
```

6.9 程序举例

【源程序代码】

```
void Array::fun1( ) {
   for(int i=0; i<4; i++)
      for(int k=0; k<4; k++)
         for(int j=k+1; j<5; j++)
            if(a[i][k]>a[i][j]){
               int t=a[i][k];
               a[i][k]=a[i][j];
               a[i][j]=t;
            }
}
```

```
void Array::fun2( ) {
   for(int j=0; j<5; j++)
      for(int i=0; i<3; i++)
         for(int k=i+1; k<4; k++)
            if(a[i][j]>a[k][j]){
               int t=a[i][j];
               a[i][j]=a[k][j];
               a[k][j]=t;
            }
}
```

6.9 程序举例

【源程序代码】

```
void Array::fun3( ) {
   int *p=&a[0][0];
   int n=4*5;
   for(int i=0; i<n-1; i++)
      for(int j=i+1; j<n; j++)
         if(*(p+i)>*(p+j)){
            int t=*(p+i);
            *(p+i)=*(p+j);
            *(p+j)=t;
         }
}
```

```
int main( ) {
    int data[4][5];
    for(int i=0; i<4; i++)
        for(int j=0; j<5; j++)
            data[i][j]=rand( )%100;
    Array a1(data,4), a2(data,4);
    a1.print( );
    a1.fun1( ); a1.print( );
    a1=a2;
    a1.fun2( ); a1.print( );
    a1=a2;
    a1.fun3( ); a1.print( );
    return 0;
}
```

6.9 程序举例

【原数组】

41	67	34	0	69
24	78	58	62	64
5	45	81	27	61
91	95	42	27	36

【行排序】

0	34	41	67	69
24	58	62	64	78
5	27	45	61	81
27	36	42	91	95

【列排序】

5	45	34	0	36
24	67	42	27	61
41	78	58	27	64
91	95	81	62	69

【全部元素排序】

0	5	24	27	27
34	36	41	42	45
58	61	62	64	67
69	78	81	91	95

6.9 程序举例

【例 6-14】定义一个类，该类可以将一组数据按给定的行列表示成一个二维数组。

- 定义数组类 Array
  - 私有数据成员： int *p; 表示动态的一维数组
  - 私有数据成员： int m, n; 分别表示二维数组行数和列数
  - 公有构造函数： Array(int *t, int a, int b) ；为一维数组分配动态内存，并用参数初始化数据成员
  - 析构函数： ~Array( ) ；撤销动态内存
  - 功能函数： int get(int i, int j) ；取数组下标为 [i][j] 的元素
  - 输出函数： void print( ) ；循环二维数组
- 主函数测试类
  - 定义测试数据：输入动态的一维数组；
  - 定义对象：实参与构造函数形参一致；
  - 通过对象调用相关成员函数：注意调用顺序和函数实参

6.9 程序举例

【源程序代码】

```
#include<iostream>
using namespace std;
class Array{
    int *p;
    int m, n;
public:
    Array(int *t, int a, int b) ;
    ~Array( ) { delete []p; }
    int get(int i, int j){
        return *(p+i*n+j);
    }
    void print( ) ;
};
```

```
Array::Array(int *t, int a, int b) {
    m=a; n=b;
    p=new int[m*n];
    for(int i=0; i<m*n; i++)
        *(p+i)=*(t+i);
}
```

```
void Array::print ( ) {
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++)
            cout<<get(i,j)<<'\t';
        cout<<'\n';
    }
}
```

6.9 程序举例

【源程序代码】

```
int main( ) {
    int *data, i0, j0;
    cout<<" 请输入行数和列数： ";
    cin>>i0>>j0;
    data=new int[i0*j0];
    cout<<" 请输入元素： ";
    for(int i=0; i<i0*j0; i++)
    cin>>*(data+i);
    Array a1(data, i0, j0);
    a1.print( );
    delete [ ]data;
    return 0;
}
```

程序运行结果

```
请输入行数和列数： 3 5
请输入元素： 1 2 3 4 5
  6
7 8 9 10 11 12 13 14
  15
1   2   3   4   5
6   7   8   9   10
11  12  13  14  15
```

6.10 习题

1．为什么实现拷贝构造函数的参数应该使用引用类型？

2．在类的成员函数中，如何返回调用该函数的当前对象？

3．假设 A 为一个类，下列语句序列执行后共调用了几次类 A 的构造函数？

A a1, a2[3], *pa, *pb[3];

4．当类中含有引用成员、常量成员、对象成员时，其构造函数是何形式？

6.10 习题

5．定义一个 Point 类表示平面上的一个点，再定义一个 Rectangle 类表示平面上的矩形，用 Point 类的对象作为 Rectangle 类的成员描述平面上矩形的顶点坐标。要求类 Point 中有相应的成员函数可以读取点的坐标值，类 Rectangle 含有一个函数用于计算并输出矩形的顶点坐标及面积。在主函数中对类 Rectangle 进行测试。

6．定义一个求 n! 的类，要求其成员数据包括 n 和 n!，成员函数分别实现设定 n 的值、计算 n! 以及输出成员数据。编写一个完整的程序对类进行测试。