面向对象方法与C++程序设计

第2章　　类与对象

大连理工大学
主讲人-赵小薇

数组



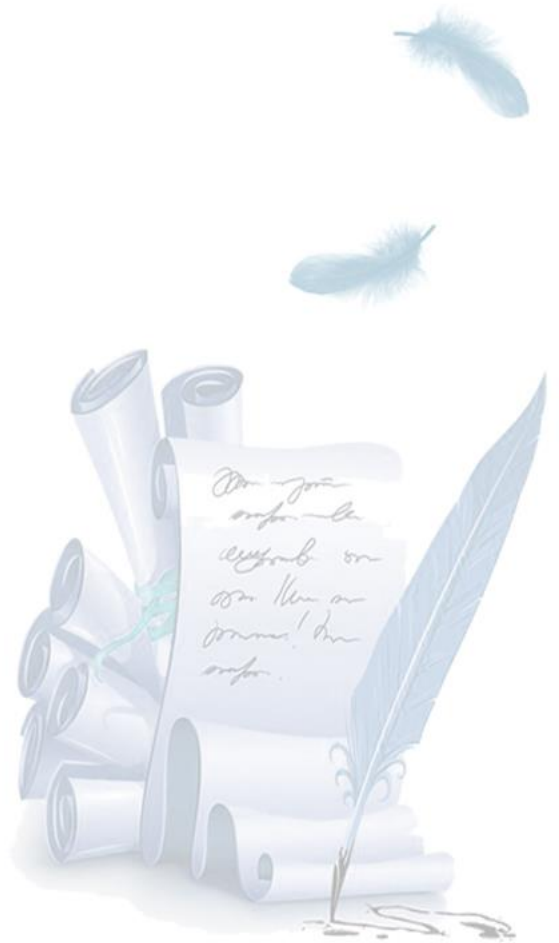
```
class Student{
private:
        char name[20];
        int age;
        float score;
public:
    Student(int s){
        strcpy(name,"no name");
        age=20;
        score=s;}
};
```

数组定义：

```
Student
stud[3]={60,70,78};
```

构造函数只有一个参数
数组可以这样初始化；

可以不初始化吗？



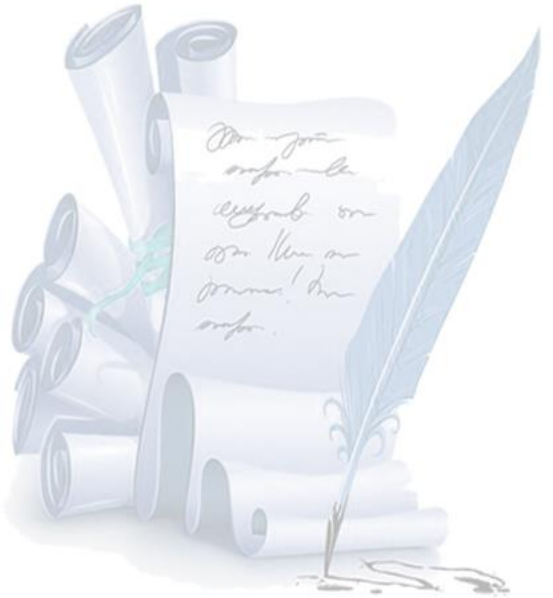
对象数组



数组元素不仅可以由简单类型组成（例如，整型数组的每一个元素都相当于一个整型变量），也可以由对象组成（对象数组的每一个元素都是同类型的对象）。

数组定义：

```
Student stud[3]={ Student("zhang san",20,60),
Student("Li si",19,70), Student("Wang wu",18,78)  };
```

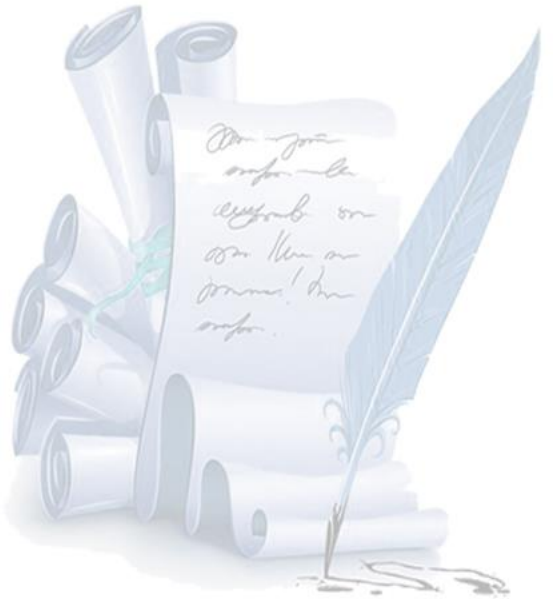


举例



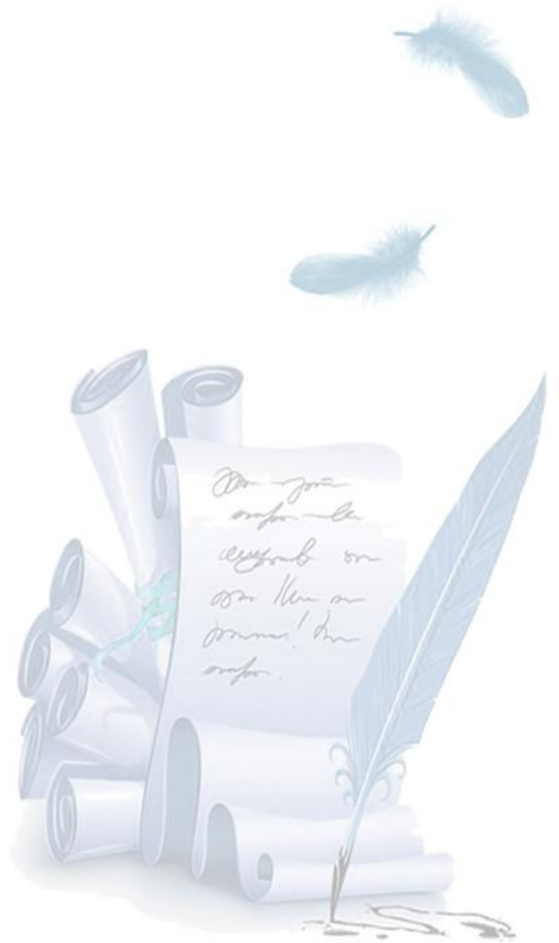
计算一组学生的总成绩和平均成绩

```
class Student{
private:
    char name[20];   int age;       float score;
public:
    Student(char * n, int a, int s){
        strcpy(name,n);         age=a;
        score=s;    sum+=score;          }
    static float sum;              //声明静态数据成员
    static float getAverage();     //声明静态函数成员
    void display();                //输出总和与平均值
};
float Student::getAverage(){return sum/3;} //实现时不需要static
float Student::sum=0;              //初始化静态数据成员
void Student::display(){           //成员函数引用静态成员
   cout<<"sum="<<sum<<"   average="<<getAverage()<<endl;}
```



```
int main(){
  //静态成员函数、静态成员数据不定义对象用类可以访问
  cout<<“sum=”<<Student::sum<<“average=”<<Student::getAverage()<<endl;
  Student stud[3]={ Student(“zhang san”,20,60), Student(“Li si”,19,70),
Student(“Wang wu”,18,78)  };
  //静态成员函数、静态成员数据定义对象后，通过类访问
  cout<<“sum=”<<Student::sum<<“average=”<<Student::getAverage()<<endl;
  //静态成员函数、静态成员数据定义对象后，通过对象访问
  cout<<"sum="<<stud[0].sum<<"average="<<stud[0].getAverage()<<endl;
  //成员函数
  stud[0].display();
  return 0;
}
```

```
sum=0   average=0
sum=208   average=69.3333
sum=208   average=69.3333
sum=208   average=69.3333
```



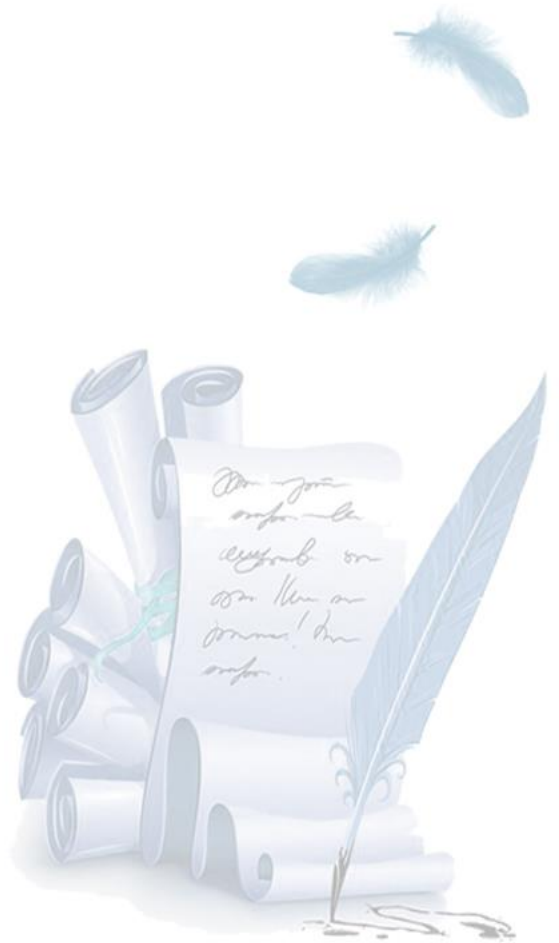
案例分析

父亲给儿子钱类的设计



Father
- name: char[16] - money: int
+Father(n:char *,m: int) +getName(): char * +receive(m:int): void +pay(m:int): int +print():void

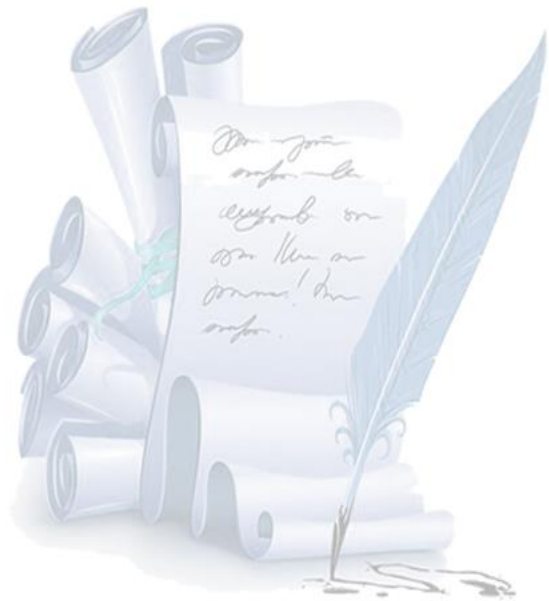
Son
- name: char[16] - money: int -father: Father *
+Son(p:Father *,n: char *,m: int) +getName(): char * +receive(m:int): void +pay(m:int): int +print():void +getFather(): Father *



父亲给儿子钱类的设计

```
int Father::pay(int m){
  if(m<=0)
     return 0;
  if(money>=m) {
     money=money-m;
    return m;
  }
  else   return 0;
}
```

```
 Father f1("李四",10000);
Son s1(&f1," 李小四",100);
s1.receive(f1.pay(1000));
f1.print();   //name: 李四 money:9000
s1.print();//name: 李小四money:1100
```



父亲给儿子钱类的设计

上述设计，任何人都可以调用f1的pay函数从李四那里支取钱。

Father
- name: char[16] - money: int
+Father(n:char *,m: int) +getName(): char * +receive(m:int): void -pay(m:int): int +manage(role :Son *,m :int): int +print():void

```
int Father::manage(Son *role,int m) {
    if(strcmp(role->getFather()->getName(),name)==0) //支付对象的父亲是自己
    return pay(m);
  else          return 0;    }
```

```
Father f1("李四",10000);
Son s1(&f1," 李小四",100);
s1.receive(f1.manage(&s1,1000));
f1.print();  //name: 李四  money:9000
s1.print();//name: 李小四money:1100
```

案例分析



1．面向对象的设计首先要找出参与任务的对象并将其抽象为类，然后由类对象协同完成任务。
2．类的设计应该忠实于实际对象，尤其是功能的权限范围，如本例的pay函数就定义为私有的。
3．存取函数对类中的私有数据成员提供对外的接口，用来设置和返回私有成员数据的值，通常称为getter和setter函数，命名方式通常为get（set）＋私有成员的名称。

问题发现：
1. 父子两个类会有很多同样的数据成员和函数成员，程序有很多冗余；
2. 还有Father类的manage函数只能给自己的儿子付款，如果想给自己的妻子、朋友付款怎么办呢？
* 这些问题将在继承和多态章节给出解决方案。