

# 第3讲 类的构成与对象的使用



# 目录

# CONTENTS



**1** 类的构成

**2** 成员函数的定义

**3** 对象的定义和使用

# 1 类的构成

声明一个关于成绩的类，如下。

```
class CScore{
```

用关键字class开始声明一个类

```
    int m_nMidtermExam;
```

数据成员

```
    int m_nFinalExam;
```

```
public:
```

```
    void SetScore(int m,int f);
```

```
    void ShowScore ();
```

成员函数

```
};
```

# 1 类的构成

在C++中，类class和结构struct的功能基本上相同，都含有数据成员以及对这些数据进行操作的成员函数。

类与结构体的主要区别是**在默认情况下**：

- 结构体中的成员都是公有的；
- 类中的成员都是私有的。

# 1 类的构成

类是一种将数据和操作合并为单一结构的类型定义，  
可表示为：

**类 = 数据 + 操作（函数）**

C++类的构成包括**类的声明**、**成员函数的定义**两个部分。

# 1.1 类的声明

类的声明包括：类的**成员声明**和类成员的**访问控制**声明。

**class** 类名

{

**private:**

**私有数据和函数**

只允许本类中的函数访问，而类外部的任何函数都不能访问

**public:**

**公共数据和函数**

可供程序中的其他函数访问，是类的对外接口

**protected:**

**保护数据和函数**

类定义的结束标志 “;” 容易被漏掉

};

# 1.1 类的声明

类的声明包括：类的**成员声明**和类成员的**访问控制**声明。

**class** 类名

{

当关键字**private**紧接着类的第1个花括号时，  
可以省略该关键字，即类成员默认是私有的。

**私有数据和函数**

**public:**

**公共数据和函数**

**protected:**

**保护数据和函数**

};

## 1.2 关于类声明的说明

- 类名后面的{ }表示类的声明范围，最后的分号表示类声明的结束；
- 建议把访问权限的成员归类放在一起，并将私有成员放在前面；
- 数据成员可以是任意数据类型；
- 不能在类的声明中给类的数据成员赋初值；
- 在类对象定义之后才能给数据成员赋初值。



例如：

```
class C {  
    private:
```

```
        char a = 'q';
```

//错误

```
        int b = 33;
```

//错误

```
    public:
```

```
        ...
```

```
};
```

此题未设置答案，请点击右侧设置按钮

类中定义的成员默认为\_\_\_\_\_访问属性

- ☐ A public
- ☐ B protected
- ☐ C private
- ☐ D friend

提交

## 2 成员函数

- ◎ 类的成员函数属于类的成员，它可以访问或调用本类的任何数据成员和其他成员函数。
- ◎ 成员函数可以被限定为私有的(private)、公有的(public)或受保护的(protected)。
- ◎ 对于私有成员函数，只能被本类的其他成员函数所调用。

## 2 成员函数

- ◎ 对于公有成员函数，可以作为类对外的接口，被外部调用。
- ◎ C++类中成员函数的定义既可放在类中，也可放在类的外面。

## 2.1 成员函数定义—外部定义

类定义中只有成员函数的**声明**，成员函数的定义在类的外部完成，其形式为：

```
返回类型 类名 :: 函数名（参数表） {  
    // 函数体  
}
```

需要在函数名的前面加上类名和作用域运算符“::”，以标明该函数是属于该类的

```

class CScore
{
private:
    int m_nMidtermExam;        //私有数据成员
    int m_nFinalExam;          //私有数据成员
public:
    void SetScore(int m,int f) ; //声明成员函数setScore()为函数
    void ShowScore ();
};

void CScore::SetScore(int m,int f) //在类外定义此函数
{
    m_nMidtermExam=m; m_nFinalExam=f;
}

void CScore::ShowScore ()
{
    cout<<"\n期中成绩: "<<m_nMidtermExam<<"\n期末成绩: "<<m_nFinalExam<<"\n总评成绩: "<<(int)(0.3*m_nMidtermExam+0.7*m_nFinalExam)<<endl;
}

```

# 说明

- SetScore()、ShowScore ()都是在类的外部定义，但它们**仍然是**类的成员函数，可以直接使用m\_nMidtermExam、m\_nFinalExam等私有数据成员。
- 在声明成员函数时，**可只给**出类型不给出参数名；而定义成员函数时，不但要说明它的类型，还要指出参数名。
- 在定义成员函数时，其返回类型要与类声明中的函数原型中的返回类型**一致**。

## 2.2 成员函数定义—内联函数

- ◎ 直接将成员函数定义在类的内部。通常是**不包含循环**的简单成员函数可以定义在类的内部。
- ◎ 定义在类内部的成员函数**自动**成为“内联函数”。
- ◎ 当程序中出现对内联函数的调用时，C++编译器直接将函数体中的代码插入到调用该函数的语句处，同时用实参来代替形参。



```
class CScore{
    private:
        int m_nMidtermExam;           //私有数据成员
        int m_nFinalExam;             //私有数据成员
    public:
        void SetScore(int m,int f)
        {
            m_nMidtermExam=m;
            m_nFinalExam=f;
        }
        void ShowScore ()
        {
            cout<<"\n期中成绩: "<<m_nMidtermExam<<"\n期末成绩: "<<m_nFinalExam<<"\n
            总评成绩: "<<(int)(0.3*m_nMidtermExam+0.7*m_nFinalExam)<<endl;
        }
};
```

## 内联成员函数

## 2.3 内联成员函数的显式定义

```
class CScore{  
    private:  
        int m_nMidtermExam;           //私有数据成员  
        int m_nFinalExam;             //私有数据成员  
    public:  
        void SetScore(int m,int f) ; //声明成员函数SetScore ()为函数  
        void ShowScore ();  
};
```

## 2.3 内联成员函数的显式定义

```
inline void CScore::SetScore(int m,int f) //在类外定义 {  
    m_nMidtermExam=m; m_nFinalExam=f;  
}
```

```
Inline void CScore ::ShowScore ()  
{  
    cout<<"\n期中成绩: "<<m_nMidtermExam<<"\n期末成绩:  
    "<<m_nFinalExam<<"\n总评成绩:  
    "<<(int)(0.3*m_nMidtermExam+0.7*m_nFinalExam)<<endl;  
}
```

必须将类的声明和内联成员函数的定义都放在**同一个文件**中, 否则编译时无法进行代码置换。

## 2.4 类成员的访问属性的设定

- ◎ 一般情况下：数据是私有的，成员函数是公有的。这样做的好处：
  - ✓ 任何有关类的错误，都是由类中的数据成员或成员函数引起的，可以**缩小错误范围**，利于及早排除错误。
  - ✓ 修改类的性质，**只需修改它的成员**。只要对象的功能保持不变，则成员函数名所形成的接口就不会变化，这种修改不会影响使用该对象的软件系统。

## 2.5 类的作用域

- 类作用域是指类定义和相应的成员函数定义的范围。
- C++认为一个类的所有成员都是一个整体的相关部分。

## 2.5 类的作用域

- ◎ 在类的作用域内，一个类的成员函数对同一类的数据成员具有**无限的访问权**。
- ◎ 而在类的外部，对该类的数据成员和成员函数的引用是要**受限制**，有时是不允许的。
- ◎ 在类的外部不能直接访问数据成员。

此题未设置答案，请点击右侧设置按钮

```
class CSample{
private:
    int i; //私有成员
public:
    int j; //公有成员
    void Set(int i1,int j1) //公有成员
    {   i=i1;           //类的成员函数可以访问类的私有成员i
        j=j1;   }      //类的成员函数可以访问类的公有成员j
};
int main()
{   CSample a; //定义类Sample的对象a
    a.Set(3,5); (1)
    cout<<a.i<<endl; (2)
    cout<<a.j<<endl; (3)
    return 0;
}
```

哪行代码存在错误?

A

(1)

B

(2)

C

(3)

D

全对

提交

### 3 对象的定义与使用

class定义的是类型，具有类类型的**变量**称为**对象**。  
声明了类之后，在类的使用时再定义对象，一旦定义了class之后，就可以像用int、float那样去定义对象。

数据类型	实 例
int	a,b (变量)
学生类	张三,李四( 对象)
水果类	苹果,桔子 (对象)



```
int main()
{
    CScore sc1,sc2;           //定义对象sc1和sc2
    sc1.SetScore(80,88);
    //调用对象sc1的成员函数SetScore(), 给sc1的数据成员赋值
    sc2.SetScore(90,92);
    //调用对象sc2的成员函数SetScore(), 给sc2的数据成员赋值
    sc1.ShowScore ();
    //调用对象sc1的成员函数ShowScore()
    sc2.ShowScore ();
    //调用对象sc2的成员函数ShowScore()
    return 0;
}
```

对象的使用

## 3.1 定义对象的格式

类名 对象名1, 对象名2.....; 如:

```
CScore sc1,sc2;
```

sc1,sc2 是CScore类类型的对象。

- 同基本数据类型一样，C++中的类也是一种数据结构，只不过这种数据结构是自定义的。
- 声明了一个类就是声明了一种类型，这时没有给它分配存储空间，只有定义了对象后，系统才为对象分配存储空间。

## 3.2 对象的使用

对象的使用是指访问**对象**的成员，不论是数据成员还是成员函数，只要是**公有**的，就可以被外部函数直接使用。

使用格式：

**对象名.数据成员**

**对象名.成员函数（实参表）**

称为“成员选择符”，  
简称点运算符

```
#include<iostream>           //运行程序并分析运行结果
using namespace std;
class CPoint {
private:
    int m_nX,m_nY;    //私有数据成员
public:
    void SetPoint (int a,int b) { m_nX=a; m_nY=b; }
    int GetX (){ return m_nX; }
    int GetY (){ return m_nY; }
};
void main( )
{ CPoint p; //定义了类CPoint的对象p
  int i, j;
  p.SetPoint (1, 2);
  i=p.m_nX;
  j=p.m_nY;
  cout<< " p.x= " <<i<< " p.y= " <<j<<endl;
}
```

错误

```
#include<iostream>           //运行程序并分析运行结果
using namespace std;
class CPoint {
private:
    int m_nX,m_nY;    //私有数据成员
public:
    void SetPoint (int a,int b) { m_nX=a; m_nY=b; }
    int GetX (){ return m_nX; }
    int GetY (){ return m_nY; }
};
void main( )
{ CPoint p; //定义了类CPoint的对象p
  int i, j;
  p.SetPoint (1, 2);
  i=p.GetX();
  j=p.GetY();
  cout<< " p.x= " <<i<< " p.y= " <<j<<endl;
}
```

## 3.3 对象中成员的访问方式

不论是数据成员，还是成员函数，只要是公有的，在类的外部可以通过类的对象进行访问。

访问对象中的成员通常有三种方法。

## 3.3 对象中成员的访问方式

(1) 通过对象名和对象选择符访问对象中的成员

对象名.数据成员名  
或  
对象名.成员函数名(实参表)

## (2) 指针变量访问类中的成员

用指向对象的**指针变量**引用类中的成员格式:

指针变量名->数据成员名

指针变量名->成员函数名 (参数)

```
int main()
{
    CPoint point1, *p;
    p = &point1;
    p->SetPoint(100, 200);
    cout<<"point1.x="<<p->GetX()<<"    "
        <<"point1.y="<<p->GetY()<<endl;
    return 0;
}
```



### (3) 引用访问对象的成员

```
int main()
{
    CPoint point1;    //定义对象
    CPoint &r = point1;    //定义对象引用
    //通过引用调用对象的公有成员函数
    r.SetPoint(100, 200);
    //通过引用调用对象的公有成员函数
    cout<<"point1.x="<<r.GetX()<<" "
        <<"point1.y="<<r.GetY()<<endl;
    return 0;
}
```

## 3.4 对象数组

### (1) 对象数组

定义一维对象数组的格式如下:

类名 数组名[下标表达式];

例如:

CScore as[5];



说明: 共建立了5个对象,即每一个数组元素是一个对象(即as[0]、as[1]、as[2]、as[3]、as[4]),共调用了5次构造函数。

# 对象数组的初始化形式

(1) 类中含有带有1个参数的构造函数

定义对象数组时，通过类名调用构造函数，也可以直接使用实参，而忽略类名。

```
D ad[3]={D(80), D(90), D(70) };  
D ad[3]={80, 90, 70 };
```

```
#include<iostream>
using namespace std;
class D {
    int x;
public:
    D(int n){ x=n; }
    int GetX( ){ return x; }
};
int main( )
{
    D ob[4]={ D(11), D(22), D(33), D(44)};
    for (int i=0;i<4;i++)
        cout<<ob[i].GetX( )<<' ';
    return 0;
}
```

通过初始值表给对象数组赋值的另一种方法

运行结果如下;  
11 22 33 44

```
#include<iostream>
using namespace std;
class D {
    int x;
public:
    D(int n){ x=n; }
    int GetX( ){ return x; }
};
int main( )
{
    D ob[4]={ 11, 22, 33, 44};
    for (int i=0;i<4;i++)
        cout<<ob[i].GetX( )<<' ';
    return 0;
}
```

帶有一个参数的构造函数

初始值表给对象数组赋值

运行结果如下;  
11 22 33 44

```
#include<iostream>
using namespace std;
class D {
    int x;
public:
    D( ){ x=123;}
    D(int n) { x=n;}
    int GetX() { return x; }};
int main()
{
    D ob[4]={55,66};
    for (int i=0;i<4;i++)
        cout<<ob[i].GetX()<<' ';
    return 0;
}
```

运行结果如下;  
55 66 123 123

# 对象数组的初始化形式

(2) 类中含有带有多个参数的构造函数  
则只能调用对应的构造函数

```
CScore as[3]={CScore(80,88), CScore(90,92), CScore(70,80) };
```

## 3.5 对象指针

对象指针就是用于存放对象地址的指针变量。  
声明对象指针的一般语法形式为：

**类名\* 对象指针名;**

`CScore *ps; //定义类CScore的对象指针变量ps`



# (1) 访问单个对象成员

```
class CScore {
```

```
    ...
```

```
};
```

```
int main()
```

```
{
```

```
    CScore *ps; //定义类CScore的对象指针变量ps
```

```
    CScore s; //定义类CScore的对象s
```

```
    ps=&s;
```

```
    ...
```

```
}
```

用对象指针访问对象成员时，不能用“.”操作符，而应使用“->”操作符

```
#include<iostream>
```

```
.....
```

```
int main()
```

```
{
```

```
    CScore score ;    //定义类Score 的对象score
```

```
    CScore *ps;        //定义ps为指向类CScore的对象指针变量
```

```
    ps=&score;        //将对象score的起始地址赋给pd
```

```
    ps->SetScore(80,88);
```

```
        //调用ps所指向的对象score中的函数 SetScore()
```

```
    ps->ShowScore();
```

```
        //调用ps所指向的对象score中的函数ShowScore()
```

```
    return 0;
```

```
}
```

## (2) 访问对象数组

```
CScore *ps;    //定义对象指针变量ps
CScore as[2];  //定义对象数组as
ps=as;         //把对象数组的第一个元素
               的地址赋给对象指针变量ps
```

```
int main()
{
    CScore score[2]; //定义类CScore 的对象score
    CScore *ps; //定义ps为指向类CScore的对象指针变量
    score[0].SetScore(80,88);
        //调用元素score[0]的成员函数SetScore()
    score[1].SetScore(90,92);
        //调用元素score[1]的成员函数SetScore()
    ps=score; //将对象score的起始地址赋给ps
    ps->ShowScore();
    ps++;
    ps->ShowScore();
    return 0;
}
```

对象指针变量ps加1，即指向下一个元素score[1]的地址。调用ps所指向的对象score中的函数ShowScore()。

此题未设置答案，请点击右侧设置按钮

假定A为一个类，则执行“`A a[3], b(2), *p;`”语句时，调用该类构造函数的次数为

- ☐ A 3
- ☐ B 4
- ☐ C 5
- ☐ D 6

提交


## 3.6 对象作为函数参数

- 使用对象作为函数参数
- 使用对象指针作为函数参数
- 使用对象引用作为函数参数

传值调用,函数中对形参对象成员的任何修改均不影响调用该函数的实参对象本身

```
#include <iostream>
using namespace std;
class E{
    int x,y;
public:
    E(int i,int j) {x=i;y=j;}
    int getx() {return x;}
    int gety() {return y;}
    void swap(E ob)
    { int temp; temp=ob.x; ob.x=ob.y; ob.y=temp; }
};

int main()
{
    E e(5,10);
    cout<<"x="<<e.getx()<<" y="<<e.gety()<<endl;
    e.swap(e);
    cout<<"x="<<e.getx()<<" y="<<e.gety()<<endl;
    return 0;
}
```




运行结果:

x=5 y=10

x=5 y=10

传址调用,函数中对形参对象成员的任何修改均影响调用该函数的实参对象本身。

```
#include <iostream>
using namespace std;
class E{
    int x,y;
public:
    E(int i,int j) {x=i;y=j;}
    int getx() {return x;}
    int gety() {return y;}
    void swap(E *ob)
    { int temp; temp=ob->x; ob->x=ob->y; ob->y=temp; }
};
int main()
{   E e(5,10);
    cout<<"x="<<e.getx()<<" y="<<e.gety()<<endl;
    e.swap(&e);
    cout<<"x="<<e.getx()<<" y="<<e.gety()<<endl;
    return 0;
}
```




运行结果:  
x=5 y=10  
x=10 y=5



引用调用,函数中对形参对象成员的任何修改均影响调用该函数的实参对象本身。

```
#include <iostream>
using namespace std;
class E{
    int x,y;
public:
    E(int l,int j) {x=l;y=j;}
    int getx() {return x;}
    int gety() {return y;}
    void swap(E &ob)
    { int temp; temp=ob.x; ob.x=ob.y; ob.y=temp; }
};

int main()
{
    E e(5,10);
    cout<<"x="<<e.getx()<<" y="<<e.gety()<<endl;
    e.swap(e);
    cout<<"x="<<e.getx()<<" y="<<e.gety()<<endl;
    return 0;
}
```



运行结果:  
x=5 y=10  
x=10 y=5