第 5 章

类 与 对 象

经典的编程方法称为面向过程的程序设计方法。在面向过程的程序设计中，使用函数实现各个功能模块，这些函数往往具有大量的形式参数，并且其中还调用了大量其他的函数。这需要程序设计人员花费大量的时间管理和分类这些函数。一种对这些函数分类的好办法为把这些函数按其处理的客观对象进行分类，例如，计算圆的周长的函数和计算圆的面积的函数，因为它们处理的客观对象都是圆，所以，将它们归为一类，同时，把它们的形式参数，即这些函数要处理的数据，也归到这一类中。通过这种方法，实现了对客观对象的属性（即数据）和方法（即函数）的封装。对于圆这个类而言，其属性就是它的半径，方法就是计算它的周长和计算它的面积，而且方法（即函数）不再需要形式参数，只需要对圆的属性进行操作。这样，就从面向过程的程序设计方法过渡到了面向对象的程序设计方法。

面向对象的程序设计方法具有三大特点：封装、继承和多态。本章将介绍类与对象的概念与程序设计方法，体现了其封装特点；第 6 章将介绍继承和多态。

自本章开始，类内部定义的变量将被称为属性或数据成员；类内部定义的函数将被称为方法或方法成员。

本章的学习目标：

- 了解类在程序设计中的重大意义
- 掌握类的设计方法
- 熟练应用类与对象进行程序设计
- 掌握借助于指针对对象成员进行操作的方法

5.1 结构体与类

C++语言设计者最初拓展 C 语言的结构体实现了封装特性，由于结构体实现了属性（即数据）的“封装”，只需要将方法（即函数）添加到结构体中，就可实现“封装”特性。但是，结构体中默认定义的属性和方法均是“公有”的，这里的“公有”的含义表示在结构体的外部，通过结构体定义的变量可以直接访问结构体中的全部属性和方法。

“封装”的真正含义在于客观对象的属性（或部分属性）是“私有”的，这里的“私有”的含义表示这些属性只能在结构体内部被其方法（或函数）访问，在结构体的外部无法访问。为了实现结构体的“封装”特性，引入了访问控制符 private 和 public，分别表示私有访问属性和公有访问属性。

程序段 6-1 展示了结构体的封装特性。


视频讲解

程序段 5-1　结构体的用法实例

(1) 头文件 main.h

```
#pragma once
struct Rectangle
{
private:
  double w, h;
public:
  Rectangle()
  {
      w = 0;
      h = 0;
  }
  void setW(double a)
  {
      w = a;
  }
  void setH(double b)
  {
      h = b;
  }
  double area()
  {
      return w * h;
  }
  double perimeter()
  {
      return 2.0 * (w + h);
  }
};
```

(2)主程序文件 main.cpp

```
#include <iostream>
#include "main.h"
using namespace std;

int main()
{
  cout << "Input two numbers:";
  double a, b;
  cin >> a >> b;
  Rectangle rect;
  rect.setW(a);
  rect.setH(b);
  cout << "Area = " << rect.area() << endl;
  cout << "Perimeter = " << rect.perimeter() << endl;
}
```

在程序段 5-1 的头文件 main.h 中，第 2～28 行定义一个结构体类型 Rectangle，用于抽象表示客观事物——矩形。其中，第 4 行的代码“private:”表示其后的语句（直到遇到第 6 行的 public）均为私有访问属性，即第 5 行的语句“double w,h;”定义的双精度浮点型变量 w 和 h 均为私有属性，分别表示矩阵的宽和高。

第 6 行的代码“public:”表示其后的语句（直到遇到新的访问属性控制符或结构体的末尾）均具有公有访问属性。这里表示第 7～27 行的方法均为公有访问属性，即在结构体外部（通过结构体定义的变量）可以直接访问这些方法。

第 7～11 行的方法为 Rectangle()，该方法与结构体同名，称为“构造”方法，一般用于初始化结构体的属性（即数据）成员。这里的第 9 行和第 10 行将数据成员 w 和 h 赋值为 0。

第 12～15 行的方法属于 set 方法，这类方法具有参数，用于给结构体的数据成员赋值。第 14 行的语句“w=a;”将 a 赋给数据成员 w。

第 16～19 行的方法也属于 set 方法，具有一个双精度浮点型参数 b，第 18 行的语句“h=b;”将 b 赋给数据成员 h。

第 20～23 行的方法 area()方法用于计算长方形面积，第 22 行的语句“return w * h;”使用数据成员 w 和 h 计算长方形的面积，并返回这个面积。

第 24～27 行的方法 perimeter()用于计算长方形的周长，第 26 行的语句“return 2.0 * (w+h);”使用数据成员 w 和 h 计算长方形的周长，并返回这个周长值。

在主程序文件 main.cpp 中，第 30 行的预编译指令“#include "main.h""将头文件 main.h 包括在程序中。

在 main()函数中，第 35 行的语句“cout << "Input two numbers: ";”输出提示信息“Input two numbers:”。

第 36 行的语句“double a,b;”定义两个双精度浮点型变量 a 和 b。

第 37 行的语句“cin >> a >> b;”输入变量 a 和 b 的值。

第 38 行的语句“Rectangle rect;”定义结构体 Rectangle 类型变量 rect。

第 39 行的语句“rect.setW(a);”调用公有方法 setW()给私有数据成员（宽）赋值。

第 40 行的语句“rect.setH(b);”调用公有方法 setH()给私有数据成员（高）赋值。

第 41 行的语句“cout << "Area=" << rect.area() << endl;”输出字符串"Area="和调用公有方法 area()得到的矩形的面积。

第 42 行的语句“cout << "Perimeter=" << rect.perimeter() << endl;”输出字符串"Perimeter="和调用公有方法 perimeter()得到的矩形的周长。

程序段 5-1 的执行结果如图 5-1 所示。

图 5-1　程序段 5-1 的执行结果

上述程序段 5-1 中的结构体封装了数据成员和方法成员，这种做法比传统基于过程的模块化编程有很多优势：

(1) 数据成员为私有成员,只能在结构体内部使用其方法访问这些成员,有效地保护了数据成员,避免了其被非法访问和修改;

(2) 方法成员仅限于操作结构体的数据成员,方法的执行不受外部数据的干扰,更容易维护且更加健壮;

(3) "封装"形式的程序更易于实现团队的协作和程序代码的拼装与移植,可大大加速软件的设计进程,大幅度缩短软件的开发周期。

这些优势也是面向对象编程优于面向过程编程的地方。

5.1.1 类

为了与C语言的结构体兼容,除了拓展C语言结构体的定义外,C++语言定义了一种全新的数据类型——类,用class关键字定义。类是一种封装了数据成员和方法成员的数据类型。与结构体不同的是,类中默认定义的成员均为私有成员。在本书中,出于对严谨性的考虑,所有程序实例均使用了访问控制符(private、public和protected)显式地表示访问控制属性,其中,protected访问控制符在第6章介绍。

类类型的一般定义形式如下:

```
class 类名
{
private:
    私有数据成员;
    私有方法成员;
protected:
    保护型数据成员;
    保护型方法成员;
public:
    公有数据成员;
    公有方法成员;
};
```

在本书中,类名使用大写首字母的"大骆驼表示法",即首字母大写且类名中完整的英文单词的首字母也大写,如"MyRectangle";类定义的变量(即对象或实例)使用全小写字母或首字母小写的"小骆驼表示法",即首字母小写但变量名中完整的英文单词的首字母大写,如"myRectangle"。

类是一种数据类型,类似于整型int,在定义类类型时,并不为类分配存储空间。因此,类中的数据成员和方法成员的顺序可随意排列,即方法成员可以放在前面,而数据成员(尽管被方法成员使用)可以放在方法成员的后面。此外,类中的各种访问控制属性的顺序也可随意排列,或者分开放置,例如:

```
class 类名
{
private:
    私有成员1;
public:
    公有成员1;
private:
```

```
    私有成员 2;
public:
    公有成员 2;
};
```

类定义的变量称为对象或实例，“类名　对象名;”和“int　a;”类似，这时系统将为“对象名”表示的对象分配存储空间。类定义对象只能在类的外部实现，因此，对象仅能访问类中的公有方法！

现在使用类类型替换结构体类型重新设计程序段 5-1，如程序段 5-2 所示。

程序段 5-2　类的定义与用法实例


视频讲解

(1) 头文件 main.h

```
1    #pragma once
2    class Rectangle
3    {
4    private:
5      double w, h;
6    public:
7      Rectangle()
8      {
9          w = 0;
10         h = 0;
11     }
12     Rectangle(double a,double b = 1.0):w(a),h(b)
13     {
14     }
15     void setW(double a)
16     {
17         w = a;
18     }
19     void setH(double b)
20     {
21         h = b;
22     }
23     double area();
24     double perimeter();
25   };
26   double Rectangle::area()
27   {
28     return w * h;
29   }
30   double Rectangle::perimeter()
31   {
32     return 2.0 * (w + h);
33   }
```

(2)主程序文件 main.cpp

```
34   #include <iostream>
```

```
35   #include "main.h"
36   using namespace std;
37
38   int main()
39   {
40     cout << "Input two numbers:";
41     double a, b;
42     cin >> a >> b;
43     Rectangle rect(a,b);
44     cout << "Area = " << rect.area() << endl;
45     cout << "Perimeter = " << rect.perimeter() << endl;
46   }
```

在程序段 5-2 的头文件 main.h 中，第 2～25 行定义了类 Rectangle，这是一种数据类型。其中，第 4 行的“private:”表示第 5 行的语句“double w,h;”定义的数据成员 w 和 h 为私有属性，实际上从 private 开始到遇到其他的访问控制符前定义的成员均为私有成员。从第 6 行的“public:”开始至第 25 行类结束符，其中的方法只受到了 public 访问控制符的影响，表示第 7～24 行的方法均为公有方法。

第 7～11 行的 Rectangle()方法为构造方法，第 12～14 行的 Rectangle()方法也是构造方法，这两个构造方法是重载的关系。当类创建对象时，构造方法用于给对象的数据成员赋值。构造方法将在 5.1.2 节详细介绍。这里，第 7～11 行的构造方法将数据成员均赋值为 0，第 12～14 行的构造方法将参数 a 和 b 分别赋给数据成员 w 和 h。

第 15～18 行的 setW()方法属于 set()方法，将参数 a 赋给数据成员 w。第 19～22 行的 setH()方法也属于 set()方法，将参数 b 赋给数据成员 h。set()方法的特点为其名称约定由 set 和大写首字母的数据成员构成。

第 23 行的语句“double area();”声明了公有方法 area()。

第 24 行的语句“double perimeter();”声明了公有方法 perimeter()。

第 23～24 行的两个方法的实现部分放在了类 Rectangle 的外部，如第 26～33 行所示。类中的方法的实现可以放在类的内部，如第 7～14 行的构造方法和第 15～22 行的 set()方法；也可放在类的外部，如第 23 行和第 24 行的两个方法。

第 26～29 行为类 Rectangle 的 area()方法的实现部分，这里的“Rectangle::area”表示类 Rectangle 中声明的方法 area()，其中“::”表示归属关系的域运算符。

第 30～33 行为类 Rectangle 的 perimeter()方法的实现部分。

在主程序文件 main.cpp 的 main()函数中，第 43 行的语句“Rectangle rect(a,b);”定义对象 rect，并调用类 Rectangle 的构造方法(见第 12 行)使用 a 和 b 分别初始化对象 rect 的数据成员 w 和 h。

第 44 行的语句“cout << "Area=" << rect.area() << endl;”输出字符串"Area="，并调用对象 rect 的公有方法 area()得到长方形的面积。

第 45 行的语句“cout << "Perimeter=" << rect.perimeter() << endl;”输出字符串"Perimeter="，并调用对象 rect 的公有方法 perimeter()得到长方形的周长。

程序段 5-2 的执行结果如图 5-2 所示。

图 5-2　程序段 5-2 的执行结果

5.1.2　构造方法

在类的公有方法中，有一种和类名同名的方法，称为构造方法。在创建类时，如果没有显式的构造方法，类将自动创建一个无参数的构造方法。构造方法的作用在于给类中的数据成员赋初始值。

这里针对程序段 5-2 中的 Rectangle 类，介绍几种定义构造方法的重载形式。

1. 无参数的构造方法

无参数的构造方法由类名和一对圆括号“( )”构成，如程序段 5-3 中的第 6 行所示。

程序段 5-3　无参数的构造方法实例


视频讲解

```
1    class Rectangle
2    {
3    private:
4      double w, h;
5    public:
6      Rectangle()
7      {
8          w = 0;
9          h = 0;
10     }
11     //其他语句
12   }
```

下述语句将调用程序段 5-3 中第 6 行的无参数构造方法：

```
Rectangle rect;
```

请注意，上述语句中没有圆括号“( )”。

建议每个类都应该设计一个无参数的构造方法。

2. 带参数的构造方法

带参数的构造方法如程序段 5-4 中的第 11～15 行的方法。

程序段 5-4　带参数的构造方法实例


视频讲解

```
1    class Rectangle
2    {
3    private:
4      double w, h;
5    public:
6      Rectangle()
7      {
8          w = 0;
```

```
9           h = 0;
10      }
11      Rectangle(double a, double b)
12      {
13          w = a;
14          h = b;
15      }
16      //其他语句
17  };
```

第11～15行的方法与类名Rectangle同名，具有两个双精度浮点型参数，在方法中，第13～14行的语句分别将a和b赋给数据成员w和h。这样表述是为了叙述简单，切记，这种表述方式是不准确的！类只是一种数据结构，定义类时并不会为类分配存储空间。对于第11～15行的构造方法准确的表述应该是当应用类创建对象时，将赋给形式参数a和b的实际参数值分别赋给所创建的对象的数据成员w和h。在不引起歧义的情况下，本节在介绍构造函数时，使用给数据成员赋值这种简略说法。

在上述情况下，使用以下语句调用程序段5-4中第11～15行的有参数构造方法：

```
Rectangle rect(3.0,4.0);
```

3. 带默认参数值的构造方法

构造方法中的参数可以使用默认值，如程序段5-5所示。


视频讲解

程序段5-5　带默认参数值的构造方法

```
1   class Rectangle
2   {
3   private:
4     double w, h;
5   public:
6     Rectangle(double a = 1.0, double b = 1.0)
7     {
8         w = a;
9         h = b;
10    }
11    //其他语句
12  };
```

在程序段5-5中，第6～10行为带默认参数值的构造方法，这里两个参数均有默认值，相当于默认的无参数构造方法，所以，这里的第6～10行的构造方法不能与程序段5-3中的构造方法共存。

调用程序段5-5中第6～10行带默认参数的构造方法的语句形如：

```
Rectangle rect;  //或  Rectangle rect(3.0);  或  Rectangle rect(3.0,5.0);
```

其中，前两者相当于“Rectangle rect(1.0,1.0);”或“Rectangle rect(3.0,1.0);”。

如果构造方法只有部分参数带有默认值，则带有默认值的参数必须位于参数列表中没有默认值的参数的右边，如程序段5-6所示。

程序段 5-6　部分参数带默认值的构造方法


视频讲解

```
1    class Rectangle
2    {
3    private:
4      double w, h;
5    public:
6      Rectangle()
7      {
8          w = 0;
9          h = 0;
10     }
11     Rectangle(double a, double b = 1.0)
12     {
13         w = a;
14         h = b;
15     }
16     //其他语句
17   };
```

这里的第 11～15 行为只有参数 b 带有默认值的构造方法，与第 6～10 行的构造方法是重载关系。

调用程序段 5-6 中第 11～15 行的构造方法的语句形如：

```
Rectangle rect(3.0);  //或  Rectangle rect(3.0,5.0);
```

其中，前者相当于“Rectangle rect(3.0,1.0);”。

4. 使用 this 指针的构造方法

带参数的构造方法中，参数名是构造方法的局部变量，参数名可以与数据成员名相同。当参数名与数据成员名相同时，构造方法中使用 this 指针区分参数和数据成员，如程序段 5-7 所示。

程序段 5-7　使用 this 指针的构造方法


视频讲解

```
1    class Rectangle
2    {
3    private:
4      double w, h;
5    public:
6      Rectangle()
7      {
8          w = 0;
9          h = 0;
10     }
11     Rectangle(double w, double h)
12     {
13         this->w = w;
14         this->h = h;
15     }
16     //其他语句
17   };
```

在程序段 5-7 中，第 11～15 行的构造方法的两个参数名与数据成员名相同。当使用类定义一个对象时，this 指针将作为对象本身的指针，第 13 行的“this-> w”指的是数据成员 w，第 14 行的“this-> h”指的是数据成员 h，从而第 13 行的语句“this-> w＝w;”表示将参数 w 赋给数据成员 w，第 14 行的语句“this-> h＝h;”表示将参数 h 赋给数据成员 h。

5. 使用列表初始化的构造方法

使用列表初始化的构造方法是最常用的构造方法，如程序段 5-8 所示。


视频讲解

程序段 5-8　使用列表初始化的构造方法

```
1    class Rectangle
2    {
3    private:
4      double w, h;
5    public:
6      Rectangle()
7      {
8          w = 0;
9          h = 0;
10     }
11     Rectangle(double a,double b) :w(a), h(b)
12     {
13         //其他语句
14     }
15     //其他语句
16   };
```

在程序段 5-8 中，第 11～14 行的构造方法为使用列表初始化的构造方法，这里使用“:”连接初始化列表“w(a)，h(b)”，这个列表表示将参数 a 赋值给 w，将参数 b 赋值给 h。

使用列表初始化的构造方法中，可以使参数名与数据成员名相同，即程序段 5-8 中的第 11～14 行可以写作如下形式：

```
11      Rectangle(double w, double h) :w(w), h(h)
12      {
13        //其他语句
14      }
```

在上述代码中，尽管可以在构造方法中放入“其他语句”，但实际上，一般来说，构造方法主要用于给类定义的对象的数据成员赋值，即在类定义对象时给其数据成员赋初始值。

5.1.3　set()方法与 get()方法

类中的构造方法在使用类创建对象时被调用以初始化对象中的数据成员，当类创建对象后，构造方法不再被调用，此时只能通过 set()方法修改对象的私有数据成员，所以，一般地，类都需要 set()方法。与 set()方法相对的方法称为 get()方法，通过 get()方法可以得到类中的私有数据成员的值。注意，这里的 set()方法和 get()方法的称谓，是程序员的约定，set()方法一般由 set()加上首字母大写的数据成员名命名，get()方法一般由 get()加上首字母大写的数据成员名命名，如程序段 5-9 所示。

程序段 5-9　set()方法和 get()方法实例


视频讲解

(1) 头文件 main.h

```
#pragma once
class Rectangle
{
private:
  double w, h;
public:
  Rectangle(double w = 0, double h = 0) :w(w), h(h){}
  void setW(double w)
  {
      if (w > 0)
          this->w = w;
      else
          this->w = 0;
  }
  void setH(double h)
  {
      if (h > 0)
          this->h = h;
      else
          this->h = 0;
  }
  double getW() const
  {
      return w;
  }
  double getH() const
  {
      return h;
  }
  double area();
  double perimeter();
};
double Rectangle::area()
{
  return w * h;
}
double Rectangle::perimeter()
{
  return 2.0 * (w + h);
}
```

(2)主程序文件 main.cpp

```
#include <iostream>
#include "main.h"
using namespace std;

int main()
```

```
46  {
47      cout << "Input two numbers:";
48      double a, b;
49      cin >> a >> b;
50      Rectangle rect;
51      rect.setW(a);
52      rect.setH(b);
53      cout << "Width: " << rect.getW() << endl;
54      cout << "Height: " << rect.getH() << endl;
55      cout << "Area = " << rect.area() << endl;
56      cout << "Perimeter = " << rect.perimeter() << endl;
57  }
```

在程序段 5-9 的头文件 main.h 中，第 7 行的语句“Rectangle(double w=0,double h=0) : w(w),h(h){}”为构造方法，带有默认参数值(可作为默认的构造方法)。

第 8～14 行的语句为 setW()方法，用形式参数 w 给数据成员 w 赋值。第 10～13 行为一个 if 结构，如果参数 w 大于 0，则将 w 的值赋给数据成员 w；否则，将数据成员 w 的值设为 0。

第 15～21 行的语句为 setH()方法，用形式参数 h 给数据成员 h 赋值。第 17～20 行为一个 if 结构，如果参数 h 大于 0，则将 h 的值赋给数据成员 h；否则，将数据成员 h 的值设为 0。

第 22～25 行的语句为 getW()方法，第 22 行的代码“double getW() const”中，const 用于修饰 getW()，表示该方法不能修改数据成员的值。这里的 getW()方法返回数据成员 w 的值(见第 24 行)。

第 26～29 行的语句为 getH()方法，第 26 行的代码“double getH() const”中，const 用于修饰 getH()，表示该方法不能修改数据成员的值。这里的 getH()方法返回数据成员 h 的值(见第 28 行)。

在主程序文件 main.cpp 中的 main()函数中，第 50 行的语句“Rectangle rect;”定义类 Rectangle 类型的对象 rect，将调用默认的构造方法，将数据成员 w 和 h 均赋值为 0。

第 51 行的语句“rect.setW(a);”调用对象 rect 的 setW()方法将 a 赋给数据成员 w。

第 52 行的语句“rect.setH(b);”调用对象 rect 的 setH()方法将 b 赋给数据成员 h。

第 53 行的语句“cout << "Width: " << rect.getW() << endl;”输出字符串"Width: "和调用 getW()方法得到的长方形的宽。

第 54 行的语句“cout << "Height: " << rect.getH() << endl;”输出字符串"Height: "和调用 getH()方法得到的长方形的高。

程序段 5-9 的执行结果如图 5-3 所示。

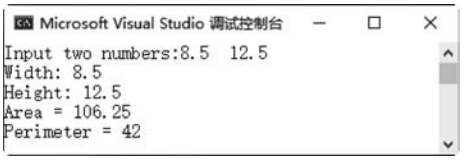

图 5-3 程序段 5-9 的执行结果

5.1.4　析构方法

与构造方法对立的方法称为析构方法。构造方法在类创建对象时对对象的数据成员进行实始化，而析构方法用于清除对象时清理对象占据的存储空间。大多数情况下，无须程序员设计析构方法，类本身有一个隐式的析构方法完成清理工作。但是，如果程序员使用了“char *”类型的数据成员，则需要编写显式的析构方法。

析构方法名是类名前添加符号“～”，而且，应尽量设为虚析构方法，即在析构方法前添加 virtual 关键字，具体原因在学习完第 6 章多态方面的内容后才会明白。

程序段 5-10 展示了析构方法的用法。


视频讲解

程序段 5-10　析构方法的用法实例

(1) 头文件 main.h

```
#pragma once
#include <cstring>
using namespace std;

class Student
{
private:
  char * name;
  char gender;
public:
  Student(const char * str,char g)
  {
      if (str != NULL)
      {
          name = new char[strlen(str) + 1];
          strcpy_s(name,strlen(str) + 1, str);
      }
      else
          name = NULL;
      gender = g;
  }
  char * getName() const
  {
      return name;
  }
  char getGender() const
  {
      return gender;
  }
  virtual ～Student()
  {
      if (name != NULL)
          delete[ ] name;
      cout << "Destructor is called." << endl;
  }
};
```

（2）主程序文件 main.cpp

```
37  #include <iostream>
38  #include "main.h"
39  using namespace std;
40
41  int main()
42  {
43    Student st("Zhang Fei", 'M');
44    cout << "Name: " << st.getName() << endl;
45    cout << "Gender: " << st.getGender() << endl;
46  }
```

在程序段 5-10 的头文件 main.h 中，定义了类 Student，其中，有两个私有数据成员，如第 8 行和第 9 行所示，为“char * name;”和“char gender;”，即指向字符的指针类型的变量 name 和字符类型的变量 gender。

第 11～21 行为构造方法 Student()，具有两个形式参数 str 和 g。这里的“const char *”表示指向常量字符串的指针，const 修饰上满足“就近”原则，例如，“char * const”类型表示指向字符串的常量指针。因此，“const char * p1”和“char * const p2”是不同的。前者表示定义一个指向常量字符串的指针 p1，p1 可以改变，但其指向的内容不可变；后者表示定义一个指向字符串的常量指针 p2，p2 不可变，但其指向的内容可变。在这两个定义下，“*p1”不可赋值，“p1”可赋值；“*p2”可赋值，“p2”不可赋值。另外，“const char * const p3”表示定义一个指向常量字符串的常量指针 p3，此时，“p3”和“*p3”均不可赋值。

在第 11～21 行的 Student()构造方法内部，第 13～19 行为一个 if 结构，如果 str 不为空，则第 15 行的语句“name=new char[strlen(str)+1];”使用 new 开辟一个长度为 strlen(str)+1 个字节的空间给 name，即 name 指向该存储空间。第 16 行的语句“strcpy_s(name,strlen(str)+1,str);”将字符串 str 复制到 name 指向的空间中，复制的长度为“strlen(str)+1”，这里“+1”不可少，因为 strlen(str)只返回字符串的真实长度，忽略了字符串结束符'\0'。第 20 行的语句“gender=g;”将参数 g 赋给数据成员 gender。

第 22～25 行为 getName()方法，用于返回数据成员 name。

第 26～29 行为 getGender()方法，用于返回数据成员 gender。

第 30～35 行为虚析构方法，其中第 32 行和第 33 行为一个 if 结构，如果数据成员 name 指针不为空，则调用第 33 行的语句“delete[] name;”释放它指向的存储空间。第 34 行的语句“cout << "Destructor is called." << endl;”输出提示信息“Destructor is called.”，这行语句仅用于测试。

在主程序文件 main.cpp 的 main()函数中，第 43 行的语句“Student st("Zhang Fei",'M');”创建类 Student 类型的对象 st，并用字符串"Zhang Fei"和字符'M'初始化数据成员。

第 44 行的语句“cout << "Name: " << st.getName() << endl;”输出字符串"Name: "和调用对象的 getName()方法得到的数据成员 name 的值。

第 45 行的语句“cout << "Gender: " << st.getGender() << endl;”输出字符串"Gender:"和调用对象的 getGender()方法得到的数据成员 gender 的值。

程序段 5-10 的执行结果如图 5-4 所示。

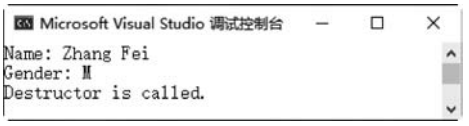

图 5-4　程序段 5-10 的执行结果

此外，建议在类中尽可能避免使用“char *”类型的数据成员，对于字符串类型的数据成员，可以使用 string 类型。这样，可以避免在类中编写析构方法。

5.2　对象与指针

类是一种数据类型，类定义的变量称为对象或实例。对象可以访问其中的公有方法，使用“.”作用符，形如“对象名.公有方法名(实际参数)”，如果没有“参数”，那么括号“( )”不可少(只有一个例外，就是构造方法在没有参数时，不能添加括号)。

可以定义类类型的指针，形式“类名　*　指针名”，这种指针可以指向该类定义的对象。程序段 5-11 介绍了指向对象的指针及其用法。

程序段 5-11　指向对象的指针实例


视频讲解

(1) 头文件 main.h

```
#pragma once
class Rectangle
{
private:
  double w, h;
public:
  Rectangle(double w = 0,double h = 0):w(w),h(h){}
  void setWH(double w, double h)
  {
      this->w = w;
      this->h = h;
  }
  double getW() const
  {
      return w;
  }
  double getH() const
  {
      return h;
  }
};
```

(2) 主程序文件 main.cpp

```
#include <iostream>
#include "main.h"
using namespace std;

```

```
26  int main()
27  {
28    Rectangle rect[4];
29    Rectangle * r;
30    for (int i = 0; i < 4; i++)
31    {
32        rect[i].setWH(2 + i * i, 3 + 2 * i);
33    }
34    r = rect;
35    for (int i = 0; i < 4; i++)
36    {
37        cout << "Width: " << r->getW() << ", Height: " << r->getH() << endl;
38        r++;
39    }
40  }
```

在程序段 5-11 的头文件 main.h 中，定义了类 Rectangle，具有两个私有数据成员 w 和 h(第 4 行和第 5 行)，具有一个构造方法 Rectangle()(第 7 行)、一个 setWH()方法(第 8～12 行)、一个 getW()方法(第 13～16 行)和一个 getH()方法(第 17～20 行)。

在主程序文件 main.cpp 的 main()函数中，第 28 行的语句“Rectangle rect[4];”定义了一个类 Rectangle 类型的对象数组 rect，具有 4 个元素。

第 29 行的语句“Rectangle * r;”定义一个指向 Rectangle 类类型的指针 r。

第 30～33 行为一个 for 结构，循环变量 i 从 0 按步长 1 递增到 3，对于每个 i，执行一次第 32 行的语句“rect[i].setWH(2+i * i,3+2 * i);”调用 rect[i]对象的 setWH 方法给对象的私有数据成员 w 和 h 赋值。

第 34 行的语句“r=rect;”使指针 r 指向 rect 数组的首地址，也可写为“r=&rect[0];”。

第 35～39 行为一个 for 结构，循环变量 i 从 0 按步长 1 递增到 3，对于每个 i，执行一次第 37 行和第 38 行的语句，其中，第 37 行的语句“cout << "Width: " << r->getW() << ", Height: " << r->getH() << endl;”输出字符串"Width: "、指针 r 指向的对象的 getW()方法返回的值、字符串", Height: "和指针 r 指向的对象的 getH()方法返回的值；第 38 行的语句“r++;”使指针 r 指向数组 rect 的下一个对象。这里的“r->getW()”为指向对象的指针调用对象中的公有方法的形式，使用“->”指向符号。

这里的第 34～39 行的代码可以替换为如下形式：

```
1   r = rect;
2   for (int i = 0; i < 4; i++)
3   {
4     cout << "Width: " << r[i].getW() << ", Height: " << r[i].getH() << endl;
5   }
```

上述代码中，使用“r[i].getW()”的形式访问对象的公有方法。这里的 rect 为一组数组，r 为指向一维数组的指针，因此，指针 r 和数组名 rect 等价(除了数组名 rect 为常量不可赋值而指针 r 为变量可赋值外)。

程序段 5-11 的执行结果如图 5-5 所示。

图 5-5　程序段 5-11 的执行结果

5.3　静态函数与友元函数

类的最大优点在于具有“封装”特性，使用类创建对象后，对象通过私有访问控制将数据成员保护起来，通过公有访问控制向外部提供实现特定功能“接口”的公有方法。但是，有两个“破坏”类的封装特性的途径，称为静态函数和友元函数（本书不介绍友元类）。建议尽可能少使用或者不使用静态函数和友元函数。

在类中的静态函数，在编译时就为其分配了存储空间，而类创建的对象是在运行时才分配存储空间，因此，静态函数属于类，而不属于类创建的对象。有一个说法，“类中的静态函数被类创建的对象共享”，这种说法也不妥。应将静态函数视为类所有的，直接使用“类名::静态函数名(实际参数)”这种形式调用静态函数。静态函数的作用主要体现在：如果想设计一些特定功能的函数，例如数学函数，这些函数没有特定的物理对象相对应，那么这时可以考虑设计一个数学类，然后，将其中的所有函数均设计为静态函数（一般会带有大量参数），通过类名可以调用这些静态函数。

类的友元函数是一种“可怕”的函数，友元函数可以直接访问类的私有成员。相比于静态函数，友元函数没有任何优点，可能的作用在于通过“对象名.私有数据成员名”的方式访问对象中的私有数据成员（这种访问私有成员的方式本应是不合法的）。正是因为这个原因，本书没有介绍友元类。注意，类的友元函数不属于类，所以友元函数既可以放在“public:”公有部分，也可以放在“private:”私有部分。

程序段 5-12 介绍了静态函数和友元函数的用法。注意，静态函数不属于类创建的对象，它无法访问对象中的方法，也就是说，静态函数无法访问类中的私有成员和公有成员，只能访问类中的静态成员。应将静态函数与对象划清界限。

程序段 5-12　静态函数和友元函数实例


视频讲解

(1) 头文件 main.h

```
#pragma once
class Circle
{
private:
  double r;
public:
  Circle(double r = 0):r(r){}
  void setR(double r)
  {
      this->r = r;
  }
```

```
12      double getR() const
13      {
14          return r;
15      }
16      double area()
17      {
18          return 3.14 * r * r;
19      }
20
21      static double sarea(double a)
22      {
23          return 3.14 * a * a;
24      }
25      friend double farea(Circle& c,double b)
26      {
27          c.r = b;
28          return c.area();
29      }
30  };
```

(2) 主程序文件 main.cpp

```
31  #include <iostream>
32  #include "main.h"
33  using namespace std;
34
35  int main()
36  {
37    Circle c1;
38    c1.setR(5.0);
39    cout << "Radius: " << c1.getR() << endl;
40    cout << "Area: " << c1.area() << endl;
41
42    cout << "Area: " << Circle::sarea(6.0) << endl;
43    cout << "Radius: " << c1.getR() << endl;
44    cout << "Area: " << farea(c1, 7.0) << endl;
45    cout << "Radius: " << c1.getR() << endl;
46  }
```

在程序段 5-12 的头文件 main.h 中,第 2～30 行定义了一个类 Circle,具有一个私有数据成员 r,如第 4 行和第 5 行所示,r 为双精度浮点型变量。

第 6～19 行为类 Circle 的公有方法,其中,第 7 行的代码“Circle(double r=0): r(r){}”为具有默认值的构造方法。

第 8～11 行为 setR()方法,将参数 r 赋给数据成员 r。

第 12～15 行为 getR()方法,返回数据成员 r 的值。

第 16～19 行为 area()方法,使用数据成员 r 返回圆的面积。

第 21～24 行为类的静态函数 sarea(),具有一个双精度浮点型参数 a,计算半径为 a 的圆的面积(第 23 行)。

第 25～29 行为类的友元函数 farea(),具有两个参数,即 Circle 类类型的引用参数 c 和

双精度浮点型参数 b。第 27 行的语句“c.r＝b;”将参数 b 赋给对象 c 的私有数据成员 r。第 28 行的语句“return c.area();”调用对象 c 的 area()方法得到圆的面积。

在主程序文件 main.cpp 的 main()函数中,第 37 行的语句“Circle c1;”定义类 Circle 类型的对象 c1。

第 38 行的语句“c1.setR(5.0);”调用对象 c1 的公有方法 setR()将 5.0 赋给其数据成员 r。

第 39 行的语句“cout << "Radius: " << c1.getR() << endl;”输出字符串"Radius: "和调用对象 c1 的公有方法 getR()得到的圆的半径,即数据成员 r 的值,结合第 38 行可知,这里 r 的值为 5.0。

第 40 行的语句“cout << "Area: " << c1.area() << endl;” 输出字符串"Area: "和调用对象 c1 的公有方法 area()得到的圆的面积,即对应于 r＝5.0 的圆的面积,为 78.5。

第 42 行的语句“cout << "Area: " << Circle::sarea(6.0) << endl;”中,使用“类名::静态函数名(实际参数列表)”的方式调用了静态函数“Circle::sarea(6.0)”,返回半径为 6.0 的圆的面积,为 113.04。

第 43 行的语句“cout << "Radius: " << c1.getR() << endl;”输出对象 c1 的数据成员 r 的值。这一条语句的执行结果表明第 42 行语句对对象 c1 的数据成员没有影响。

第 44 行的语句“cout << "Area: " << farea(c1,7.0) << endl;”调用了友元函数 farea(),将对象 c1 和数值 7.0 作为其参数,得到半径为 7.0 的圆的面积,为 153.86。

第 45 行的语句“cout << "Radius: " << c1.getR() << endl;”输出对象 c1 的私有数据成员 r 的值,将发现 r 的值为 7(而非原来的 5),是因为第 44 行的友元函数 farea()的执行改变了对象 c1 的私有数据成员 r。

程序段 5-12 的执行结果如图 5-6 所示。

图 5-6　程序段 5-12 的执行结果

5.4　对象复制

在使用类定义一个新对象时可以使用它已定义的对象初始化,即将已定义的对象赋给同类型的新对象。

在 C++语言中,存在着赋值和赋址两种方式。赋值是指将一个变量的值赋给另一个变量,这两个变量占有不同的存储空间,赋值完成后修改任一变量的值,另一个变量不受影响。赋址是指将一个变量的地址赋给另一个变量,这类变量一般为指针,这样两个变量指向相同的地址,修改它们中任一变量的“值”,都是修改它们共同指向的存储空间中的“值”,因此,修改其中一个变量的“值”将影响到另一个变量的“值”。

下面程序段 5-13 展示了赋值和赋址的情况。


视频讲解

程序段 5-13　赋值与赋址的不同情况实例

```
1    #include <iostream>
2    #include <cstring>
3    #include <string>
4    using namespace std;
5
6    int main()
7    {
8      int a, b;
9      a = 30;
10     b = a;
11     cout << "a = " << a << endl;
12     cout << "b = " << b << endl;
13     a = 15;
14     cout << "a = " << a << endl;
15     cout << "b = " << b << endl;
16
17     string s1, s2;
18     s1 = "Hello";
19     s2 = s1;
20     cout << "s1 = " << s1 << endl;
21     cout << "s2 = " << s2 << endl;
22     s1 = "World!";
23     cout << "s1 = " << s1 << endl;
24     cout << "s2 = " << s2 << endl;
25
26     int * p = &a;
27     int * q;
28     q = p;
29     cout << " * p = " << * p << endl;
30     cout << " * q = " << * q << endl;
31      * p = 20;
32     cout << " * p = " << * p << endl;
33     cout << " * q = " << * q << endl;
34
35     char * c1 = new char[20]; strcpy_s(c1, 6, "Hello");
36     char * c2;
37     c2 = c1;
38     cout << "c1 = " << c1 << endl;
39     cout << "c2 = " << c2 << endl;
40     strcpy_s(c1, 6, "World");
41     cout << "c1 = " << c1 << endl;
42     cout << "c2 = " << c2 << endl;
43   }
```

在程序段 5-13 的 main()函数中，第 8～15 行为变量间赋值的情况，第 8 行定义整型变量 a 与 b；第 9 行将 30 赋值给变量 a；第 10 行将变量 a 赋值给变量 b；第 11 行和第 12 行输出变量 a 和 b 的值，此时两个变量均为 30。然后，第 13 行将 15 赋给变量 a；第 14 行和第

15 行输出变量 a 和 b 的值，此时，变量 a 为 15，变量 b 仍为 30。

第 17～24 行为 string 类型的变量间赋值的情况。其中，第 17 行定义字符串变量 s1 和 s2；第 18 行将字符串“Hello”赋值给 s1；第 19 行将 s1 赋值给变量 s2；第 20 行和第 21 行输出变量 s1 和 s2 的值，此时，两者均为“Hello”；第 22 行将“World!”赋给变量 s1；第 23 行和第 24 行输出 s1 和 s2 的值，此时，s1 为“World!”，而 s2 为“Hello”。

第 26～33 行为变量间赋址的情况。其中，第 26 行定义指向整型变量的指针 p，p 指向整型变量 a；第 27 行定义指向整型变量的指针 q；第 28 行将指针 p 赋给指针 q，此时两个指针都指向相同的地址(即 &a)；第 29 行和第 30 行输出“ * p”和“ * q”的值，均为 a 的值，即 15。第 31 行将“ * p”赋为 20；第 32 行和第 33 行输出“ * p”和“ * q”的值，由于指针 p 和 q 均指向相同的地址，“ * p”和“ * q”的值相同，均为 20。

第 35～42 行为变量间赋址的情况。其中，第 35 行定义指向字符的指针 c1，并使用 new 操作符开辟长度为 20 字节的存储空间，然后，调用 strcpy_s 函数将字符串 Hello 赋值给指针 c1 指向的空间；第 36 行定义指向字符的指针 c2；第 37 行将指针 c1 赋给 c2；第 38 行和第 39 行输出指针 c1 和 c2 的值，即输出这两个指针指向的字符串的值，由于两个指针指向同一地址，故均输出“Hello”；第 40 行将字符串“World”赋值给指针 c1 指向的空间；第 41 行和第 42 行输出指针 c1 和 c2 的值，即输出这两个指针指向的字符串的值，由于两个指针指向同一地址，此时均输出“World”。

程序段 5-13 的执行结果如图 5-7 所示。

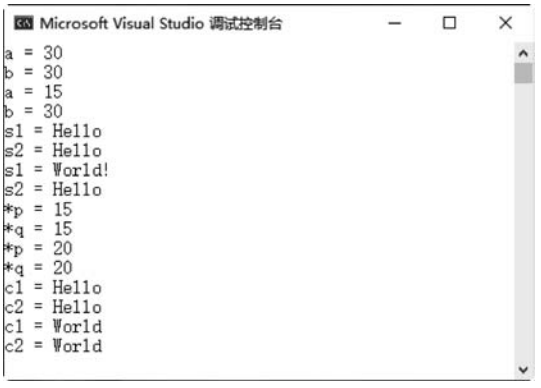

图 5-7　程序段 5-13 的执行结果

由于类是一种数据类型，类定义的变量(即对象)也可以复制。如果类的私有成员全是赋值类型的变量，类定义的对象间可以互相赋值；但若类的私有成员中有赋址类型的变量，则类定义的对象间不能直接赋值，这是因为当一个对象赋值给另一个对象时，两个对象中的赋址类型的数据成员指向同一个地址空间，这时如果修改了一个对象的这种赋址类型的数据成员，那么另一个对象的指向这个地址的赋址类型的数据成员的“值”也跟着变化，从而破坏了类的“封装”特性。

下面的程序段 5-14 就反映了这种情况，这种情况称为对象间的“浅复制”，这个程序在 Visual Studio 2022 下可编译通过，也可执行(但会有警告)。


视频讲解

程序段 5-14　对象间的浅复制实例

(1) 头文件 main.h

```
#pragma once
#include <iostream>
#include <cstring>
using namespace std;

class Student
{
private:
  char * name;
  char gender;
  double score[5];
public:
  Student()
  {
      name = NULL;
      gender = 'F';
      for (int i = 0; i < 5; i++)
          score[i] = 0;
  }
  Student(const char * str, char g)
  {
      name = NULL;
      if (str != NULL)
      {
          name = new char[200];
          strcpy_s(name, strlen(str) + 1, str);
      }
      gender = g;
      for (int i = 0; i < 5; i++)
          score[i] = 0;
  }
  void setScore(double *d)
  {
      for (int i = 0; i < 5; i++)
          score[i] = *(d + i);
  }

  void setName(const char* str)
  {
      if (strlen(str) < 200)
      {
          strcpy_s(name, strlen(str) + 1, str);
      }
      else
          cout << "Error" << endl;
  }
  void getMember() const
```

```
48      {
49          if (name != NULL)
50              cout << "Name: "<< name << endl;
51          cout << "Gender: " << gender << endl;
52          cout << "Score: ";
53          for (int i = 0; i < 5; i++)
54          {
55                cout << score[i] << " ";
56          }
57          cout << endl;
58      }
59      virtual ~Student()
60      {
61          if (name != NULL)
62              delete[ ] name;
63      }
64  };
```

(2) 主程序文件 main.cpp

```
65  #include < iostream >
66  #include "main.h"
67  using namespace std;
68
69  int main()
70  {
71    Student s1("Zhang Fei", 'M'), s2;
72    double sc[5] = { 91.5,93.2,97.5,94.9,90.1 };
73    s1.setScore(sc);
74    s1.getMember();
75    s2 = s1;
76    s2.getMember();
77    cout << endl;
78
79    s1.setName("Guan Yu");
80    s1.getMember();
81    s2.getMember();
82  }
```

在程序段 5-14 的头文件 main.h 中定义了类 Student，在类 Student 中的私有成员中，使用了指向字符的指针 name(第 9 行)。

由于类 Student 使用了指针类型的私有成员，在第 20～31 行的构造方法中，为指针 name 开辟了长度为 200 字节的空间(第 25 行)，然后，将构造方法的参数 str 赋给 name 指向的空间。

由于类 Student 在构造方法中使用 new 操作符开辟了一块存储空间，在第 59～63 行添加了虚析构方法，将开辟的存储空间释放掉(第 61 行和第 62 行)。

类 Student 中其余的部分如下：在私有数据成员中，定义了字符型的 gender 成员，表示学生的性别(第 10 行)；定义了双精度浮点型的一维数组 score，保存学生的成绩(第 11 行)。在公有方法中，定义了一个默认构造方法(第 13～19 行)；定义了一个 setScore()方

法，用于向 score 成员赋值(第 32～36 行)；定义了一个 setName()方法，用于向指针 name 指向的空间赋值(第 38～46 行)；定义了一个 getMember()方法，用于显示数据成员的值(第 47～58 行)。

在主程序文件 main.cpp 的 main()函数中，第 71 行的语句“Student s1("Zhang Fei", 'M'),s2;”定义了类 Student 类型的对象 s1 和 s2，同时，对 s1 进行了初始化。

第 72 行的语句“double sc[5]={ 91.5,93.2,97.5,94.9,90.1 };”定义了双精度浮点型一维数组 sc，具有 5 个元素，并做了初始化。

第 73 行的语句“s1.setScore(sc);”调用对象 s1 的 setScore()方法使用一维数组 sc 对对象 s1 的成员 score 进行赋值。

第 74 行的语句“s1.getMember();”调用对象 s1 的 getMember()方法输出对象 s1 的各个数据成员的值。

第 75 行的语句“s2=s1;”将对象 s1 赋值给 s2。

第 76 行的语句“s2.getMember();”调用对象 s2 的 getMember()方法输出对象 s2 的各个数据成员的值。此时，对象 s2 和 s1 的数据成员的值相同。

第 77 行的语句“cout << endl;”输出回车换行符。

第 79 行的语句“s1.setName("Guan Yu");”调用对象 s1 的方法 setName 将对象 s1 中成员 name 指向的空间赋为“Guan Yu”(原来是“Zhang Fei”)。由于对象 s2 的数据成员 name 和对象 s1 的数据成员 name 指向相同的地址空间(共享同一存储空间)，所以，针对对象 s1 的 name 的赋值，将影响到对象 s2 的 name 成员。

第 80 行语句“s1.getMember();”调用对象 s1 的 getMember()方法输出对象 s1 的各个数据成员的值，其中 name 的输出为“Guan Yu”。

第 81 行的语句“s2.getMember();”调用对象 s2 的 getMember()方法输出对象 s2 的各个数据成员的值。此时，对象 s2 中的成员 name 受到第 79 行语句的影响而输出“Guan Yu”。

程序段 5-14 的执行结果如图 5-8 所示。

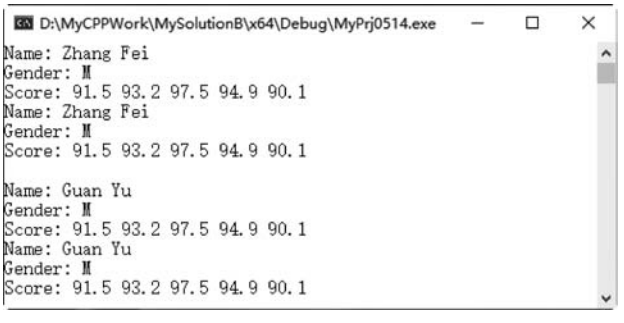

图 5-8　程序段 5-14 的执行结果

为了修正程序段 5-14 的缺陷，即当类中有指针类型的数据成员时，必须添加一个“复制构造方法”，复制构造方法与其他的构造方法是重载的关系。针对程序段 5-14 的修改有以下两点：

(1) 将程序段 5-15 所示的“复制构造方法”插入到程序段 5-14 的第 31 行和第 32 行

之间。

程序段 5-15　复制构造方法


视频讲解

```
1       Student(const Student& stu)
2       {
3           cout << "Copy." << endl;
4           name = NULL;
5           if (stu.name != NULL)
6           {
7               name = new char[200];
8               strcpy_s(name, strlen(stu.name) + 1, stu.name);
9           }
10          gender = stu.gender;
11          for (int i = 0; i < 5; i++)
12              score[i] = stu.score[i];
13      }
```

在程序段 5-15 中，复制构造方法的参数为“const Student& stu”即类 Student 的引用类型的参数，这里使用了 const 表示形式参数 stu 为常量，在复制构造方法内部不能被修改。调用复制构造方法的形式为“Student 目标对象(被复制的对象)”，结合第 1 行代码，这里的 stu 为“被复制的对象”。

第 3 行的语句“cout << "Copy." << endl;”输出一个提示信息“Copy.”，用于表示该复制构造方法被调用。

第 7 行和第 8 行的语句对“目标对象”中的 name 开辟一块新的存储空间，然后，将“被复制的对象”stu 的 name 指向的内容复制到“目标对象”的 name 中。

这种使用了“复制构造方法”的情况，称为对象间的“深度”复制。

(2) 将程序段 5-16 中的主程序文件 main.cpp 替换程序段 5-14 中的 main.cpp。

程序段 5-16　主程序文件 main.cpp


视频讲解

```
1     #include <iostream>
2     #include "main.h"
3     using namespace std;
4
5     int main()
6     {
7       Student s1("Zhang Fei", 'M');
8       double sc[5] = { 91.5,93.2,97.5,94.9,90.1 };
9       s1.setScore(sc);
10      s1.getMember();
11      Student s2(s1);
12      s2.getMember();
13      cout << endl;
14
15      s1.setName("Guan Yu");
16      s1.getMember();
17      s2.getMember();
18    }
```

在程序段 5-16 中，使用了“Student s2(s1);”将对象 s1 赋值给对象 s2，而不是使用“s2＝s1”，这是因为赋值运算符“＝”对于私有成员 char * 没有重载，运算符重载将在第 7 章介绍。

程序段 5-14 修改后的执行结果如图 5-9 所示。

图 5-9　程序段 5-14 修改后的执行结果

尽管采用深度“复制”可以完美解决类的私有数据成员为指针时的对象复制问题，但是建议初学者在创建类时尽量不使用指针，全部使用具有赋值特性的数据成员。这样，无须考虑深度复制的情况，即无须编写复制构造方法和析构方法。更重要的是，在绝大多数情况下，类中只需要使用具有赋值特性的数据成员。

下面程序段 5-17 实现了与程序段 5-14 完全相同的功能，但是仅使用了赋值特性的数据成员，且实现了深度复制(不用编写“复制构造方法”)。


视频讲解

程序段 5-17　类中仅使用赋值特性的数据成员(实现与程序段 5-14 相同的功能)实例

(1) 头文件 main.h

```
1    #pragma once
2    #include <iostream>
3    using namespace std;
4
5    class Student
6    {
7    private:
8      string name;
9      char gender;
10     double score[5];
11   public:
12     Student(string str = "", char g = 0)
13     {
14         name = str;
15         gender = g;
16         for (int i = 0; i < 5; i++)
17               score[i] = 0;
18     }
19     void setScore(double * d)
20     {
21         for (int i = 0; i < 5; i++)
```

```
22              score[i] =  *(d + i);
23      }
24      void setName(string str)
25      {
26          name = str;
27      }
28      void getMember() const
29      {
30          cout << "Name: " << name << endl;
31          cout << "Gender: " << gender << endl;
32          cout << "Score: ";
33          for (int i = 0; i < 5; i++)
34              cout << score[i] << " ";
35          cout << endl;
36      }
37  };
```

(2) 主程序文件 main.cpp

```
38  #include <iostream>
39  #include "main.h"
40  using namespace std;
41
42  int main()
43  {
44    Student s1("Zhang Fei", 'M'), s2;
45    double sc[5] = { 91.5,93.2,97.5,94.9,90.1 };
46    s1.setScore(sc);
47    s1.getMember();
48    cout << endl;
49    s2 = s1;
50    s2.getMember();
51    cout << endl;
52
53    s1.setName("Guan Yu");
54    s1.getMember();
55    cout << endl;
56    s2.getMember();
57  }
```

在程序段 5-17 的头文件 main.h 中，定义了类 Student(第 5～37 行)，包含私有数据成员字符串 string 类型的 name(第 8 行)、字符类型的 gender(第 9 行)和双精度浮点型的一组数组 score(第 10 行)。还包含了公有方法：第 12～18 行的构造方法 Student()、第 19～23 行的 setScore()方法、第 24～27 行的 setName()方法和第 28～36 行的 getMember()方法。这些数据成员和方法成员的含义与程序段 5-14 相同，不再赘述。

在程序段 5-17 的类 Student 中，私有数据成员没有使用指针等赋址类型的变量，因此，无须创建“复制构造方法”和析构方法，就可以实现深度复制，即一个对象赋值给另一个对象后，两个对象间不会有共享的存储空间，可认为两个对象互不相关。

主程序文件 main.cpp 的 main()函数与程序段 5-14 的 main()函数相同(只是这里的

main()函数中多了第 48 行和第 55 行的语句“cout << endl;”),第 49 行的语句“s2=s1;”将对象 s1 赋给对象 s2。然后,第 53 行的语句“s1. setName("Guan Yu");”将对象 s1 中的私有数据成员 name 赋值为“Guan Yu”。第 54 行的语句“s1. getMember();”调用对象 s1 的方法 getMember()输出对象 s1 的数据成员。第 56 行的语句“s2. getMember();”调用对象 s2 的方法 getMember()输出对象 s2 的数据成员。程序段 5-17 的执行结果如图 5-10 所示。由图 5-10 可知,第 53 行修改对象 s1 的数据成员 name 没有影响到对象 s2 中的数据成员 name。

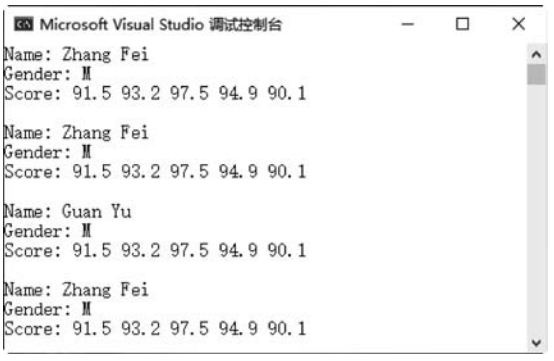

图 5-10 程序段 5-17 的执行结果

5.5 本章小结

将所要实现的功能划分为小的功能模块,基于这些功能模块编写大量的函数,进而通过函数调用实现所需功能,这种软件设计方法称为面向过程的程序设计方法。在面向过程的程序设计方法中,函数是程序设计的中心。面向对象的程序设计方法是面向过程的程序设计方法的进化,在面向对象的程序设计方法中,类是程序设计的中心,将所要实现的功能的客体分成小的客体,基于这个小的客体抽象成数据结构——类,类定义的变量称为对象或实例,对应着小的客体,类中定义了所抽象的小的客体的属性(即数据成员)和方法(即函数),通过类定义的对象实现所需要的功能。面向对象的程序设计方法管理一个个的“类”,比起面向过程的程序设计方法管理一个个的函数而言,更加直观方便。

面向对象程序设计具有抽象、封装、继承和多态的优点。如何将一个客体抽象为类需要程序员不断地实践积累经验,这是面向对象程序设计思想的难点,特别是对于那些从面向过程的程序设计过渡到面向对象的程序设计的程序员而言。本章重点讨论了类的封装特性,介绍了类的定义、设计方法和封装技术(即访问控制技术),介绍了对象的使用方法,简述了借助对象访问类中公有方法的技巧,并介绍了静态函数和友元函数,讨论了对象复制的注意事项。第 6 章将讨论类的继承和多态特性。

对于 C++语言入门者而言,建议在类中不使用赋址类型的数据成员,字符串使用 string 而不使用“char *”定义,不使用需要借助 new 操作符为数据成员开辟空间的数据成员和公有方法,这样可避免设计复制构造方法和析构方法。

习题

1. 将圆抽象为一个类，圆的半径和圆心坐标为类的私有数据成员，求圆的面积为类的公有方法成员，并编写构造方法、set 方法和 get 方法。

2. 将三角形抽象为一个类，三角形的三个顶点坐标为类的私有数据成员，求三角形的周长为类的公有方法成员，并编写构造方法、set 方法和 get 方法。

3. 编写一个学生类，其中学生的姓名、性别、学号和 9 门课的成绩（使用共用体类型）作为私有数据成员，输入和输出学生的信息作为公有方法成员。

4. 编写一个扇形类，扇形的圆心坐标、半径和圆心角作为类的私有数据成员，计算扇形的面积作为类的公有方法，并编写带默认参数的构造方法、set 方法和 get 方法。然后，编写主程序计算一个扇形的面积。

5. 编写一个整数类，类的私有数据成员为两个整数，类的公有方法成员为实现两个整数间的四则运算，同时编写带默认参数的构造方法、set 方法和 get 方法。

6. 编写一个复数类，类的私有数据成员为两个复数（用结构体类型表示），实现两个复数间的加、减、乘和除法运算作为类的 4 个公有方法成员，同时编写构造方法、set 方法和 get 方法。编写主程序文件实现两个复数间的各种运算。