

# C++语言程序设计：MOOC版

清华大学出版社（ISBN 978-7-302-42104-7）

## 第7章 面向对象程序设计之一



中國農業大學

阚道宏

# 第7章 面向对象程序设计之一

- 程序=数据+算法
  - 变量是程序中的数据元素，函数是程序中的算法元素
  - 面向对象程序设计方法将程序中的数据元素和算法元素根据其内在关联关系进行分类管理，这就形成了“类”的概念。分类可以更好地管理
  - 类相当于是一种自定义数据类型，用类所定义的变量称为“对象”
- 类与对象编程
  - 某些程序员定义类，编写类代码
  - 某些程序员使用类定义对象，然后通过对象重用别人的类代码



# 第7章 面向对象程序设计之一

- 本章内容
  - [7.1 面向对象程序设计方法](#)
  - [7.2 类的定义](#)
  - [7.3 对象的定义与访问](#)
  - [7.4 对象的构造与析构](#)
  - [7.5 对象的应用](#)
  - [7.6 类中的常成员与静态成员](#)
  - [7.7 类的友元](#)



# 7.1 面向对象程序设计方法

- 结构化程序设计到面向对象程序设计
- **程序实例：**编写一个计算圆形、长方形面积和周长的C++演示程序。程序由甲乙2位程序员分工协作，共同编写。使用结构化程序设计方法，将程序划分成5个函数。甲负责编写主函数，乙负责编写计算面积、周长的子函数



# 7.1 面向对象程序设计方法

例7-1 计算圆形、长方形面积和周长的C++演示程序（数据分散管理）

程序员甲：主函数（1.cpp）

```
1 #include <iostream>
2 using namespace std;
3
4 // 声明外部函数原型
5 double CArea(double r);
6 double CLen(double r);
7 double RArea(double a, double b);
8 double RLen(double a, double b);
9
10 int main( )
11 {
12     double r, a, b; // 定义局部变量
13     cin >> r >> a >> b; // 输入半径和长宽
14
15     cout << CArea( r ) << endl; // 圆面积
16     cout << CLen( r ) << endl; // 圆周长
17
18     cout << RArea(a, b) << endl; // 长方形面积
19     cout << RLen(a, b) << endl; // 长方形周长
20     return 0;
21 }
```

程序员乙：子函数（2.cpp）

```
// 2个计算圆形面积和周长的函数
double CArea(double r)
{ return (3.14*r*r); }
double CLen(double r)
{ return (3.14*2*r); }

// 2个计算长方形面积和周长的函数
double RArea(double a, double b)
{ return (a*b); }
double RLen(double a, double b)
{ return ( 2*(a+b) ); }
```

**数据分散管理策略**：就是将数据分散交由各个函数管理，函数各自定义变量申请自己所需的内存空间，其它函数不能直接访问其中的数据，需要时可通过数据传递来实现共享。采用分散管理策略时，程序员应当将定义变量语句放在函数的函数体中，这样所定义的变量称为**局部变量**。局部变量属本函数所有，其它函数不能直接访问。



# 7.1 面向对象程序设计方法

例7-2 计算圆形、长方形面积和周长的C++演示程序（数据集中管理）

程序员甲：主函数（1.cpp）

```
1  #include <iostream>
2  using namespace std;
3
4  extern double r, a, b; // 声明外部全局变量
5
6  // 声明外部函数原型
7  double CArea( );
8  double CLen( );
9  double RArea( );
10 double RLen( );
11
12 int main( )
13 {
14     cin >> r >> a >> b; // 输入半径和长宽
15
16     cout << CArea( ) << endl; // 圆面积
17     cout << CLen( ) << endl; // 圆周长
18
19     cout << RArea( ) << endl; // 长方形面积
20     cout << RLen( ) << endl; // 长方形周长
21     return 0;
22 }
```

程序员乙：子函数（2.cpp）

double r, a, b; // 定义全局变量

// 2个计算圆形面积和周长的函数

```
double CArea( )
{ return (3.14*r*r); }
```

```
double CLen( )
{ return (3.14*2*r); }
```

// 2个计算长方形面积和周长的函数

```
double RArea( )
{ return (a*b); }
double RLen( )
{ return ( 2*(a+b) ); }
```

**数据集中管理策略：**将数据集中管理，统一定义公共的变量来存放共享数据，所有函数都可以访问。采用集中管理策略时，程序员应当将定义变量语句放在函数外面（不在任何函数的函数体中），这样所定义的变量称为**全局变量**。全局变量不属于任何函数，是公共的，所有函数都可以访问。

# 7.1 面向对象程序设计方法

例7-3 计算圆形、长方形面积和周长的C++演示程序（分类管理）

程序员甲：使用类（1.cpp）

程序员乙：定义类（2.cpp）

```
1 #include <iostream>
2 using namespace std;
3
4 class Circle // 圆形类：声明
5 {
6 public:
7     double r; // 半径：数据成员
8     double CArea( ); // 求面积：函数成员
9     double CLen( ); // 求周长：函数成员
10 };
11 class Rectangle // 长方形类：声明
12 {
13 public:
14     double a, b; // 长宽：数据成员
15     double RArea( ); // 求面积：函数成员
16     double RLen( ); // 求周长：函数成员
17 };
18
19 int main( )
20 {
21     Circle obj1; // 定义圆形类的对象obj1
22     Rectangle obj2; // 定义长方形类的对象obj2
23
24     // 输入obj1的半径、obj2的长宽
25     cin >> obj1.r >> obj2.a >> obj2.b;
26
27     cout << obj1.CArea( ) << endl; // obj1的面积
28     cout << obj1.CLen( ) << endl; // obj1的周长
29
30     cout << obj2.RArea( ) << endl; // obj2的面积
31     cout << obj2.RLen( ) << endl; // obj2的周长
32     return 0;
33 }
```

```
class Circle // 圆形类：声明，即声明成员
{
public:
    double r; // 半径：数据成员
    double CArea( ); // 求面积：函数成员
    double CLen( ); // 求周长：函数成员
};
// 圆形类：实现，函数成员的具体定义
double Circle::CArea( ) // 求面积
{ return (3.14*r*r); }
double Circle::CLen( ) // 求周长
{ return (3.14*2*r); }
```

```
class Rectangle // 长方形类：声明成员
{
public:
    double a, b; // 长宽：数据成员
    double RArea( ); // 求面积：函数成员
    double RLen( ); // 求周长：函数成员
};
// 长方形类：实现，函数成员的具体定义
double Rectangle::RArea( )
{ return (a*b); }
double Rectangle::RLen( )
{ return (2*(a+b)); }
```

分  
类

# 7.1 面向对象程序设计方法

例7-4 计算圆形、长方形面积和周长的C++演示程序（引入类头文件）

程序员甲：使用类（1.cpp）

```
1#include <iostream>
2using namespace std;
3
4#include "2.h" // 声明类Circle、Rectangle
5
6int main( )
7{
8    Circle obj1; // 定义圆形类的对象obj1
9    Rectangle obj2; // 定义长方形类的对象obj2
10
11    // 输入obj1的半径、obj2的长宽
12    cin >> obj1.r >> obj2.a >> obj2.b;
13
14    cout << obj1.CArea( ) << endl; // obj1的面积
15    cout << obj1.CLen( ) << endl; // obj1的周长
16
17    cout << obj2.RArea( ) << endl; // obj2的面积
18    cout << obj2.RLen( ) << endl; // obj2的周长
19    return 0;
20}
21
22
23
24
25
26
27
28
29
```

程序员乙：定义类

头文件（2.h）

```
class Circle // 圆形类：声明，即声明成员
{
public:
    double r; // 半径：数据成员
    double CArea( ); // 求面积：函数成员
    double CLen( ); // 求周长：函数成员
};
```

class Rectangle // 长方形类：声明

```
{
public:
    double a, b; // 长宽：数据成员
    double RArea( ); // 求面积：函数成员
    double RLen( ); // 求周长：函数成员
};
```

源程序文件（2.cpp）

```
#include "2.h" // 声明类Circle、Rectangle
```

// 圆形类：实现，具体的函数代码

```
double Circle::CArea( ) // 求面积
{ return (3.14*r*r); }
double Circle::CLen( ) // 求周长
{ return (3.14*2*r); }
```

// 长方形类：实现，具体的函数代码

```
double Rectangle::RArea( )
{ return (a*b); }
double Rectangle::RLen( )
{ return (2*(a+b)); }
```





# 7.1 面向对象程序设计方法

- 封装

- 提供代码的程序员

将相对独立、能广泛使用的程序功能提炼出来，编写成函数或类等形式的可重用代码。可重用代码的特点是“一次开发，长期使用”。编写重用代码程序员的性格是“辛苦自己，方便别人”，因为只辛苦一次

- 使用别人代码的程序员

图省事，图方便。例如，头文件就是为了方便程序员声明别人编写的函数或类而引入的语法形式



# 7.1 面向对象程序设计方法

- 圆形类Circle

- 增加输入输出功能

```
void Circle::Input( ) { cin >> r; }  
void Circle::Show( ) // 显示圆  
{  
    cout << CArea( ) << endl;  
    cout << CLen( ) << endl;  
}
```

- 使用圆形类Circle

```
Circle obj1;
```

```
obj1.Input( ); // 调用函数成员Input输入半径
```

```
obj1.Show( ); // 调用函数成员Show显示圆的面积和周长
```

```
class Circle // 圆形类：声明，即声明成员  
{  
    double r; // 半径：数据成员  
    double CArea( ); // 求面积：函数成员  
    double CLen( ); // 求周长：函数成员  
  
    void Input( ); // 输入半径  
    void Show( ); // 显示面积和周长  
};
```



# 7.1 面向对象程序设计方法

- 类的封装：开放、隐藏
- 访问权限
  - 公有权限（**public**）。被赋予公有权限的类成员是开放的，称为公有成员
  - 私有权限（**private**）。被赋予私有权限的类成员将被隐藏，称为私有成员
  - 保护权限（**protected**）。被赋予保护权限的类成员是半开放的，称为保护成员
- 编写类的程序员与使用类的程序员
  - 编写类的程序员通过设定访问权限将类成员封装起来，将需要访问的成员开放出来，不需访问的成员隐藏起来，这样可以避免误访问
  - 访问权限是编写类的程序员为控制使用类程序员的访问操作，保证正确访问类成员而设定的



# 7.1 面向对象程序设计方法

例7-5 计算圆形、长方形面积和周长的C++演示程序（引入访问权限）

程序员甲：使用类（1.cpp）

```
1#include <iostream>
2using namespace std;
3
4#include "2.h"
5
6int main( )
7{
8    Circle obj1; // 定义圆形类的对象obj1
9    obj1.Input(); // 输入半径：公有成员
10   obj1.Show(); // 显示结果：公有成员
11
12   Rectangle obj2; // 定义长方形类的对象obj2
13   cin >> obj2.a >> obj2.b; // 长宽：公有
14   cout << obj2.RArea() << endl; // 面积：公有
15   cout << obj2.RLen() << endl; // 周长：公有
16   return 0;
17}
18
19
```

程序员乙：定义类

头文件（2.h）

```
class Circle // 圆形类：声明，即声明成员
{
private: // 以下成员为私有权限
    double r; // 半径：数据成员
    double CArea(); // 求面积：
    double CLen(); // 求周长：
public: // 以下成员为公有权限
    void Input(); // 输入半径：
    void Show(); // 显示结果：
};
```

接口1

接口2

Circle
-r : double
-CArea() : double
-CLen() : double
+Input() : void
+Show() : void

class Rectangle // 长方形类：声明成员

```
{
public: // 所有成员均为公有权限
    double a, b; // 长宽：数据成员
    double RArea(); // 求面积：
    double RLen(); // 求周长：
};
```

接口1

接口2

接口3

接口4

Rectangle
+a : double
+b : double
+RArea() : double
+RLen() : double

源程序文件（2.cpp） .....代码省略



中國農業大學

閻道宏

# 7.1 面向对象程序设计方法

- 类的接口
  - 公有成员是封装后类对外的接口
  - 一个类必须有公有成员，否则这个类无法使用
  - 程序员设计类时应根据功能要求合理设定成员权限。一方面要开放用户正常使用所必须的成员，另一方面要尽可能隐藏不需被直接访问的成员

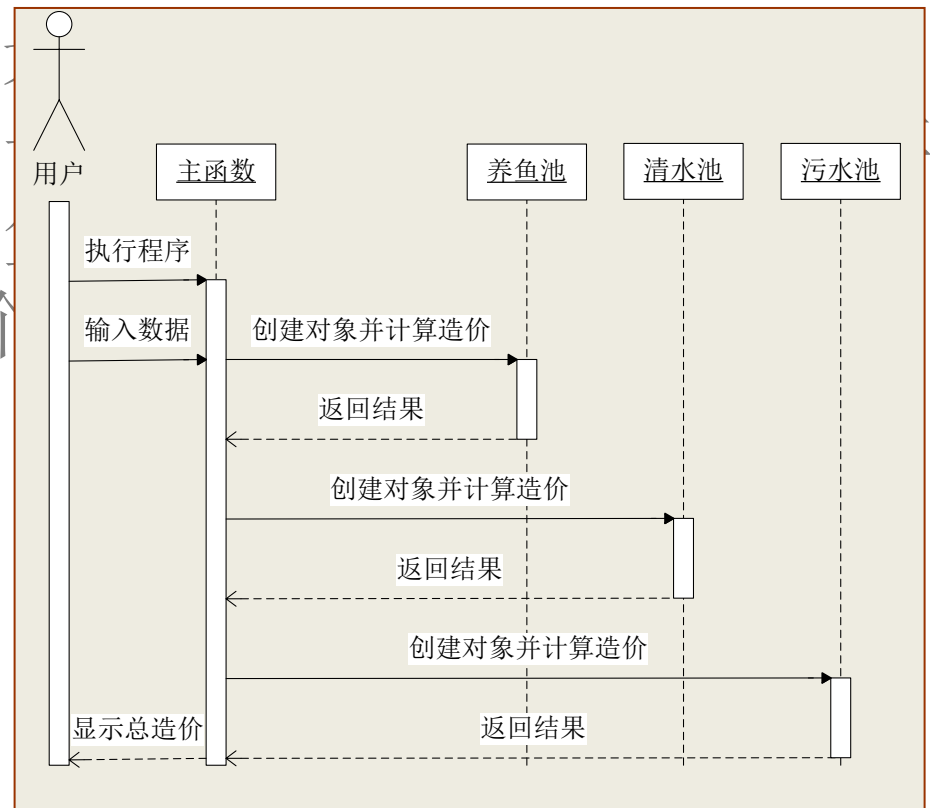


# 7.1 面向对象程序设计方法

- 面向对象程序的设计过程

- 设计任务：公园计划另外配套修建一大放清水和污水。养元/m<sup>2</sup>。请设计一个C++程序。

- 分析



# 7.1 面向对象程序设计方法

- 面向对象程序的设计过程
  - 抽象

CirclePool
+ r : double
+ CircleCost( ) : double

CirclePool
+ r : double
+ CircleCost( ) : double
+ FenceCost( ) : double

```
class CirclePool // 声明部分：声明类中有哪些成员以及各成员的权限
{
public:
    double r; // 声明数据成员：半径
    double CircleCost( ); // 声明函数成员的原型：计算圆形水池造价
};
// 实现部分：各函数成员的完整定义
double CirclePool::CircleCost( )
{ return (3.14 * r*r * 10); }
```



中國農業大學

阚道宏

# 7.1 面向对象程序设计方法

- 面向对象程序的设计过程

- 组装

```
int main( )
```

```
{
```

```
    RectanglePool obj1; // 定义长方形养鱼池对象obj1
```

```
    CirclePool obj2, obj3; // 定义圆形清水池对象obj2、圆形污水池对象obj3
```

```
    cin >> obj1.a >> obj1.b; // 输入养鱼池对象obj1的长宽
```

```
    cin >> obj2.r >> obj3.r; // 输入清水池对象obj2和污水池对象obj3的半径
```

```
    double totalCost = obj1.RectCost( ); // 计算养鱼池对象obj1的造价
```

```
    totalCost += obj2.CircleCost( ); // 累加清水池对象obj2的造价
```

```
    totalCost += obj3.CircleCost( ); // 累加污水池对象obj3的造价
```

```
    cout << totalCost << endl;
```

```
    return 0;
```

```
}
```



中國農業大學

閻道宏



# 7.2 类的定义

## C++语法：定义类

```
class 类名 // 类声明（Declaration）部分
{
    public :
        公有成员
    protected :
        保护成员
    private :
        私有成员
};
各函数成员的完整定义 // 类实现（Implementation）部分
```

### 语法说明：

- **class**是定义类时的关键字；
- 类名需符合标识符的命名规则；
- 类成员有2种，分别是数据成员和函数成员。某些特殊的类可能只包含1种成员，比如只包含数据成员，或只包含函数成员；
- 访问权限有3种，分别是**public**（公有权限）、**protected**（保护权限）和**private**（私有权限）；
- 类定义代码分为2个部分。**类声明部分**在大括号中声明数据成员、函数成员，并指定各成员的访问权限。声明数据成员的语法形式类似于定义变量语句，但声明时不能初始化。声明函数成员只是声明其原型，完整的函数定义代码放在大括号外面的**类实现部分**，定义时需在函数名前加“类名::”限定，指明该函数属于哪个类；
- 函数成员可以访问本类中任意位置的数据成员，或调用本类中任意位置的函数成员。类成员之间互相访问不需要“先定义，后访问”，或“先声明，后访问”，也不受权限约束。在类定义代码（包含声明和实现2部分）范围内，任何类成员都可以被本类的其它成员访问，类成员具有**类作用域**（Class scope）；
- 不同类作用域之间的标识符可以重名，即不同类的成员之间可以重名。



## 7.2 类的定义

- 类Abc

```
class Abc // 类声明部分
{
    int d0; // int型数据成员d0, 未指定访问权限时默认为private
public :
    float d1; // float型数据成员d1, 访问权限为public
    void fun1( ); // 函数成员fun1, 默认为其前一个成员的权限, 即public
protected :
    bool d2; // bool型数据成员d2, 访问权限为protected
    void fun2( char ch ); // 函数成员fun2, 默认为其前一个成员的权限, 即protected
private :
    char d3, str[10]; // char型数据成员d3和str (数组), 访问权限为private
    int fun3( ); // 函数成员fun3, 默认为其前一个成员的权限, 即private
};
```

// 类实现部分

```
void Abc::fun1( ) { ..... } // 函数成员fun1的完整定义代码
void Abc::fun2( char ch ) { ..... } // 函数成员fun2的完整定义代码
int Abc::fun3( ) { ..... } // 函数成员fun3的完整定义代码
```



中國農業大學

閻道宏

## 7.2 类的定义

- 数据成员的语法细则
  - 数据成员也称为属性，是类中的变量，用于保存数据。
  - 数据成员的类型可以是基本数据类型，也可以是自定义数据类型。
  - 不同数据成员之间的类型可以相同，也可以不同。
  - 数据成员不能与其它成员重名。
  - 声明数据成员的语法形式类似于定义变量，所不同的是声明数据成员不能初始化。



## 7.2 类的定义

- 函数成员的语法细则
  - 函数成员也称为方法，是类中的函数，其功能通常是对本类中的数据成员进行处理。
  - 函数成员可直接访问本类中的数据成员，数据成员相当于是类中的全局变量。函数成员可以直接调用本类中的其它函数成员。
  - 在类中声明函数成员时可以指定形参的默认值，即带默认形参值的函数。
  - 不同函数成员之间可以重名，即重载函数。两个函数的形参个数不同，或数据类型不同，那么这两个函数就可以重载。函数成员不能与数据成员重名。
  - 可以将函数成员定义成内联函数。在类实现部分定义函数成员时可以用“`inline`”关键字将其定义成内联函数，或者直接将该函数成员定义在类声明部分的大括号里。C++编译器默认将直接定义在类声明部分大括号里的函数成员当做内联函数处理。



## 7.2 类的定义

- 访问权限的语法细则
  - 每个类成员都有并且只有一种访问权限。
  - 定义类时，不同权限成员可以按任意次序编排。为便于阅读，通常将相同权限的成员编排在一起；或将数据成员编排在一起，将函数成员编排在一起。
  - 关键字**public**、**protected**和**private**指定了其后续成员的访问权限。在类声明中，同一关键字可出现多次，也可以不出现（如果没有该访问权限的成员）。通常，一个类需包含公有权限的成员，否则该类没有对外的接口，无法使用。



## 7.3 对象的定义与访问

- 类相当于是程序员自己定义的一种新的数据类型，可称为类类型
- 定义对象和定义变量的语法形式基本相同  
Circle obj1;
- 类就像是一张图纸，计算机执行定义对象语句就是按照图纸在内存中创建一个程序实例（即对象）



## 7.3 对象的定义与访问

- 定义好的对象可以访问。访问对象就是通过接口（即公有成员）操作内存中的对象，实现特定的程序功能，比如读写对象中的公有数据成员，或调用其中的公有函数成员
- 对象中的公有成员可以访问，非公有成员（私有成员、保护成员）不能访问，这就是对象中成员的访问权限控制
- 访问对象中公有成员需使用成员运算符“.”，以“对象名.数据成员名”的形式访问对象中的变量，以“对象名.函数成员名(实参列表)”的形式调用对象中的函数

```
obj1.Input( ); // 调用对象obj1的函数成员Input，输入obj1的半径  
obj1.Show( ); // 调用对象obj1的函数成员Show，显示其面积和周长
```

```
Rectangle obj2; // 定义1个Rectangle类的长方形对象obj2  
cin >> obj2.a >> obj2.b; // 输入长方形对象obj2的长宽  
cout << obj2.RArea( ) << endl; // 调用对象obj2的函数成员RArea计算其面积  
cout << obj2.RLen( ) << endl; // 调用对象obj2的函数成员RLen计算其面积
```



## 7.3 对象的定义与访问

- 对象指针

Rectangle obj2; // 定义1个类Rectangle的对象obj2  
obj2.a、obj2.b、obj2.RArea( )、obj2.RLen( )

Rectangle \*p; // 定义1个类Rectangle的对象指针p  
p = &obj2; // 将对象obj2的地址赋值给对象指针p

```
cin >> (*p).a >> (*p).b;  
cout << (*p).RArea( ) << endl;  
cout << (*p).RLen( ) << endl;
```

p->a、p->b、p->RArea( )、p->RLen( )





# 7.3 对象的定义与访问

- 类与对象的编译原理
  - 类代码的编译

函数成员CArea: 调整前		函数成员CArea: 调整后	
1	double Circle::CArea( ) // 求面积		double Circle::CArea( <b>Circle * const this</b> )
2	{		{
3	return ( 3.14 * r * r );		return ( 3.14 * <b>this-&gt;r</b> * <b>this-&gt;r</b> );
4	}		}
函数成员Show: 调整前		函数成员Show: 调整后	
1	double Circle::Show( ) // 显示面积和周长		double Circle::Show( <b>Circle * const this</b> )
2	{		{
3	cout << CArea( ) << endl;		cout << <b>this-&gt;CArea</b> ( ) << endl;
4	cout << CLen( ) << endl;		cout << <b>this-&gt;CLen</b> ( ) << endl;
5	}		}



# 7.3 对象的定义与访问

- 类与对象的编译原理

- 对象的创建

Circle obj1;

Rectangle obj2;

obj1.r

obj2.a, obj2.b

obj1.Input( );

obj1.Show( );

操作系统 及已运行的应用程序		代 码 区
int main( ) { ..... }		
Input(Circle * const this); Show(Circle * const this); <del>CArea(Circle * const this);</del> <del>CLen(Circle * const this);</del>	类 Circle	
RArea(Rectangle * const this); RLen(Rectangle * const this);	类 Rectangle	
栈剩余内存	自动 存储 区 (栈)	
系统空闲内存	堆	

(a) 加载后的程序副本

操作系统 及已运行的应用程序		代 码 区
int main( ) { ..... }		
Input(Circle * const this); Show(Circle * const this); <del>CArea(Circle * const this);</del> <del>CLen(Circle * const this);</del>	类 Circle	
RArea(Rectangle * const this); RLen(Rectangle * const this);	类 Rectangle	
obj1.r	obj1	
obj2.a	obj2	自动 存储 区 (栈)
obj2.b		
栈剩余内存		
系统空闲内存	堆	

(b) 主函数执行时的程序副本



中國農業大學

閻道宏

## 7.3 对象的定义与访问

- 类与对象的编译原理
  - 调用对象函数成员语句的编译

```
Circle obj1;  
obj1.Input( );
```

```
Input( &obj1 );
```

```
void Circle::Input( Circle * const this )  
{ cin >> this->r; }
```



## 7.3 对象的定义与访问

- 类与对象的编译原理

- 调用对象函数成员语句的编译

```
Rectangle rect1, rect2;
```

```
cin >> rect1.a >> rect1.b;
```

```
cin >> rect2.a >> rect2.b;
```

```
cout << rect1.RArea( ) + rect2.RArea( ) << endl;
```

定义长方形类对象: Rectangle rect1, rect 2;

	rect1.a
rect1	rect1.b
	rect2.a
rect2	rect2.b

- 每个对象所占用的内存空间都等于类中全部数据成员所需内存空间的总和

- 多个同类对象共用同一个函数，内存中只需要保存一份函数代码



中國農業大學

阚道宏

# 7.4 对象的构造与析构

- 变量在内存中的生存期
  - 全局变量、局部变量、静态变量
- 全局对象、局部对象、静态对象
- 程序执行过程中
  - 计算机创建对象，为对象分配内存空间，我们称对象在内存中诞生了
  - 当对象生存期结束时，计算机销毁对象，释放其内存空间，我们称对象死亡了
  - 创建对象的过程称为对象的构造，销毁对象的过程称为对象的析构



## 7.4 对象的构造与析构

- 程序员可参与对象的构造和析构过程
- 构造个性化对象
  - 初始化对象
  - 构造时申请额外的内存
- 销毁个性化对象
  - 释放额外申请的内存

定义长方形类对象: Rectangle rect1, rect 2;

rect1	rect1.a
	rect1.b
rect2	rect2.a
	rect2.b



# 7.4 对象的构造与析构

- 构造函数
  - 构造函数名必须与类名相同。
  - 构造函数由计算机自动调用，程序员不能直接调用。
  - 构造函数通过形参传递初始值（可指定默认形参值），实现对新建对象数据成员的初始化。
  - 构造函数可以重载，即定义多个同名的构造函数，这样可提供多种形式的对象构造方法。
  - 构造函数可以定义成内联函数。
  - 构造函数没有返回值，定义时不能写函数类型，写void也不行。
  - 构造函数通常是类外调用，其访问权限应设为public或protected，不能设为private。
  - 一个类如果没有定义构造函数，编译器在编译时将自动添加一个空的构造函数，称为默认构造函数，其形式为：  
类名() { }



## 7.4 对象的构造与析构

- 构造函数
  - 初始化对象

Circle :: Circle( double x ) // 带形参的构造函数

```
{ r = x; }
```

Circle :: Circle( ) // 不带形参的构造函数

```
{ r = 0; }
```

Circle obj( 5.0 );

Circle obj;

Circle :: Circle( double x = 0 )

```
{ r = x; }
```





## 7.4 对象的构造与析构

- 构造函数
    - 用一个已存在的对象初始化当前新对象
- ```
Circle :: Circle( Circle &obj )  
{  r = obj.r; }
```
- 称为拷贝构造函数

```
Circle obj1( 5.0 );
```

```
Circle obj2( obj1 );
```



## 7.4 对象的构造与析构

- 构造函数

- 显示对象的构造过程

Circle :: Circle( ) // 不带形参的构造函数

```
{ r = 0; cout << "Circle( ) called." << endl; }
```

Circle :: Circle( double x ) // 带形参的构造函数

```
{ r = x; cout << "Circle( double x ) called." << endl; }
```

Circle :: Circle( Circle &obj ) // 拷贝构造函数

```
{ r = obj.r; cout << "Circle( Circle &obj ) called." << endl; }
```

Circle obj; // 显示信息: Circle( ) called.

Circle obj1( 5.0 ); // 显示信息: Circle( double x ) called.

Circle obj2( obj1 ); // 显示信息: Circle( Circle &obj ) called.



中國農業大學

閻道宏

## 7.4 对象的构造与析构

- 构造对象时申请额外内存

```
class Student // 定义一个学生信息类Student
```

```
{
```

```
public:
```

```
    char Name[9], ID[11]; // 保存姓名和学号的字符数组Name、ID
```

```
    int Age; // 保存年龄的int型数据成员Age
```

```
    double Score; // 保存成绩的double型数据成员Score
```

```
    Student(char *pName, char *pID, int iniAge, double iniScore) // 内联构造函数
```

```
{
```

```
    strcpy(Name, pName); strcpy(ID, pID); // 初始化姓名、学号
```

```
    Age = iniAge; Score = iniScore; // 初始化年龄、成绩
```

```
}
```

```
    // .....其它函数成员省略
```

```
};
```

```
Student obj(“张三”, “1400500001”, 19, 95 );
```



中國農業大學

閻道宏

## 7.4 对象的构造与析构

- 增加备注成员: `char *Memo;`

```
Student(char *pName, char *pID, int iniAge, double iniScore, char *pMemo)
{
    strcpy(Name, pName);  strcpy(ID, pID); // 初始化姓名、学号
    Age = iniAge;  Score = iniScore; // 初始化年龄、成绩

    int len = strlen( pMemo ); // 计算实际传递来的备注信息长度
    if (len <= 0)  Memo = 0; // 没有备注信息。0表示空指针
    else
    {
        Memo = new char[len + 1]; // 按照实际长度分配内存
        strcpy(Memo, pMemo); // 初始化备注信息
    }
}
```

```
Student obj( "张三", "1400500001", 19, 95, "" );
Student obj( "张三", "1400500001", 19, 95, "市级三好学生" );
```



中國農業大學

閻道宏

## 7.4 对象的构造与析构

- 析构函数
  - 析构函数名必须为“~类名”。
  - 析构函数由计算机自动调用，程序员不能直接调用。
  - 析构函数没有形参。
  - 析构函数没有返回值，定义时不能写函数类型，写void也不行。
  - 一个类只能有一个析构函数。
  - 析构函数通常是类外调用，其访问权限应设为public或protected，不能设为private。
  - 一个类如果没有定义析构函数，编译器在编译时将自动添加一个空的析构函数，称为默认析构函数，其形式为：  
~类名() { }



## 7.4 对象的构造与析构

- 析构函数
  - 清理内存
  - 设置与当前对象相关的系统状态

```
~Student( )  
{  
    if (Memo != 0) delete [ ]Memo;  
}
```



# 7.4 对象的构造与析构

- 拷贝构造函数中的深拷贝与浅拷贝

```
Student(char *pName, Student &obj) {
    strcpy(Name, pName);
    Age = iniAge; Score = iniScore;
    Memo = obj.Memo; // 拷贝备注信息指针，注：并没有再分配内存
}

int len = strlen( pMemo ); // 计算实际传递来的备注信息长度
if (len <= 0) Memo = 0; // 没有备注信息。0表示空指针
else
{
    Memo = new char[len + 1]; // 按照实际长度分配内存
    strcpy(Memo, pMemo); // 初始化备注信息
}
}
```

```
Student obj1( "张三", "1400500001", 19, 95, "市级三好学生" );
```

```
Student obj2( obj1 ); // 将自动调用默认拷贝构造函数
```



中國農業大學

阚道宏

Student( Student &obj )

```
{
    strcpy(Name, obj.Name);  strcpy(ID, obj.ID); // 拷贝姓名、学号
    Age = obj.Age;  Score = obj.Score; // 拷贝年龄、成绩
    // Memo = obj.Memo; // 拷贝备注信息指针，注：并没有再分配内存
    int len = strlen( obj.Memo ); // 计算obj所引用对象的备注信息长度
    if (len <= 0)  Memo = 0; // 没有备注信息。0表示空指针
    else
    {
        Memo = new char[len+1]; // 按照实际长度分配内存（需加1个结束符'\0'）
        strcpy(Memo, obj.Memo); // 拷贝obj所引用对象的备注信息
    }
}
```

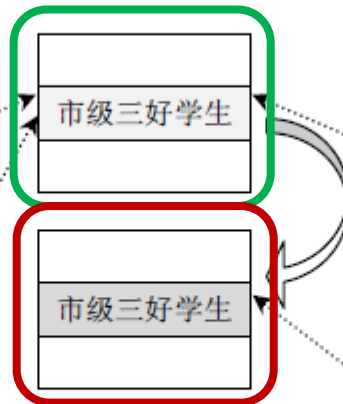
构

贝

生” );

|       |       |            |
|-------|-------|------------|
|       | Name  | 张三         |
|       | ID    | 1400500001 |
| obj1. | Age   | 19         |
|       | Score | 95         |
|       | Memo  |            |
|       | Name  | 张三         |
|       | ID    | 1400500001 |
| obj2. | Age   | 19         |
|       | Score | 95         |
|       | Memo  |            |

(a) 浅拷贝



|            |       |       |
|------------|-------|-------|
| 张三         | Name  |       |
| 1400500001 | ID    |       |
| 19         | Age   | obj1. |
| 95         | Score |       |
|            | Memo  |       |
| 张三         | Name  |       |
| 1400500001 | ID    |       |
| 19         | Age   | obj2. |
| 95         | Score |       |
|            | Memo  |       |

(b) 深拷贝





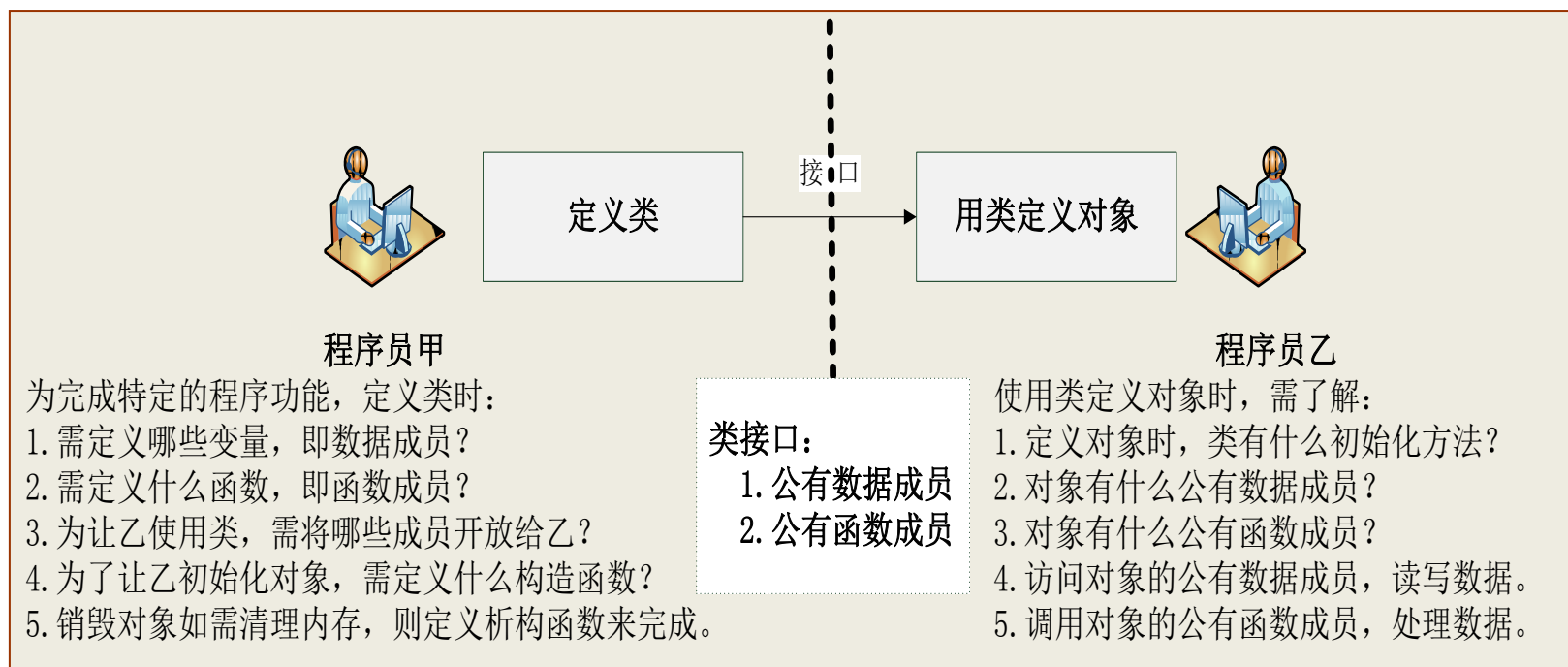
# 7.4 对象的构造与析构

- 类与对象编程的主要内容
  - 定义类。程序员在定义一个类的时候要考虑到5大要素，即数据成员、函数成员、各成员的访问权限、构造函数和析构函数。
  - 定义对象。将类当作一种自定义数据类型来定义变量，所定义的变量就称为对象。计算机执行定义对象语句就是严格按照类所描述的5大要素在内存中创建该类的对象，所创建的对象具有类所规定的的数据成员、函数成员及访问权限。
  - 访问对象。访问对象就是通过接口（即公有成员）操作内存中的对象，实现特定的程序功能，比如读写对象中的公有数据成员，或调用其中的公有函数成员。



# 类与对象编程举例

- 任务：编写一个模拟银行存款账户管理的C++程序
- 分析



# 类与对象编程举例

- 定义类（程序员甲）
  - 类模型应包含5个要素：
    - 描述该类事物需要哪些数据（属性）？
    - 对该类事物需做哪些处理（方法）？
    - 为保证类模型被正确使用，该如何封装上述属性和方法？
    - 使用类模型在内存中创建一个具体事物，称为该类的一个对象。创建对象时如何进行初始化（构造方法）？
    - 对象使用完后应释放内存，销毁对象。销毁对象时需做哪些善后处理（析构方法）？
  - C++语言使用类的语法形式来描述类模型，将属性定义成类的数据成员，将方法定义成类的函数成员。通过为类成员设定访问权限来实现类的封装。为类添加构造函数可以为该类对象提供初始化方法，添加析构函数可以在销毁对象时做某些特定的善后处理



# 类与对象编程举例

- 定义类  
程序员甲

```
// 头文件：account.h
class Account // 类声明部分
{
public:
    int no; // 账号
    char name[10]; // 账户名
    float money; // 存款金额
    void Deposit( ); // 存款
    void Withdraw( ); // 取款
    void Show( ); // 显示余额
};
```

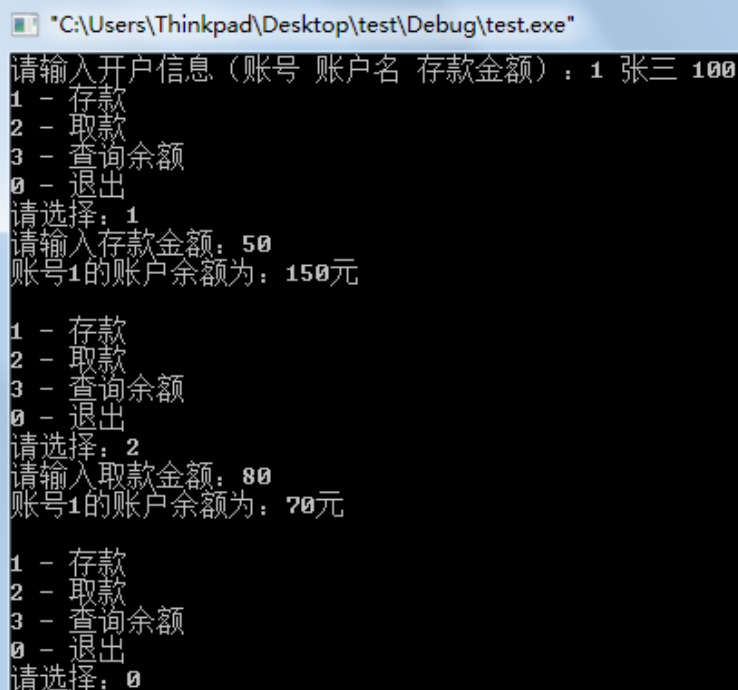
```
// 源程序文件：Account.cpp
#include <iostream>
using namespace std;
#include "account.h" // 插入头文件account.h，声明银行账户类Account
// 类实现部分
void Account :: Deposit( ) // 存款
{
    cout << "请输入存款金额： ";
    float x; cin >> x;
    money += x;
    Show( ); // 显示存款后的账户余额
}
void Account :: Withdraw( ) // 取款
{
    cout << "请输入取款金额： ";
    float x; cin >> x;
    if (money < x) cout << "账户余额不足。";
    else money -= x;
    Show( ); // 显示取款后的账户余额
}
void Account :: Show( ) // 显示余额
{
    cout << "账号" << no << "的账户余额为： " << money << "元\n\n";
}
```



# 类与对象

- 使用类  
程序员乙

```
#include <iostream>
using namespace std;
#include <string.h> // 插入头文件string.h, 声明系统函数strcpy()
#include "account.h" // 插入头文件account.h, 声明银行账户类Account
int main() // 主函数定义代码
{
    // 键盘输入账号、账户名和存款金额等开户信息, 先定义3个临时变量
    cout << "请输入开户信息 (账号 账户名 存款金额): ";
    int x; char str[10]; float y;
    cin >> x >> str >> y;
```



```
"C:\Users\Thinkpad\Desktop\test\Debug\test.exe"
请输入开户信息 (账号 账户名 存款金额): 1 张三 100
1 - 存款
2 - 取款
3 - 查询余额
0 - 退出
请选择: 1
请输入存款金额: 50
账号1的账户余额为: 150元

1 - 存款
2 - 取款
3 - 查询余额
0 - 退出
请选择: 2
请输入取款金额: 80
账号1的账户余额为: 70元

1 - 存款
2 - 取款
3 - 查询余额
0 - 退出
请选择: 0
```

# 类与对象编程举例

- 类代码的完善与优化

- 构造函数

- 程序员甲

```
Account :: Account(int iniNo, char *iniName, float iniMoney)
{
    no = iniNo; strcpy(name, iniName); money = iniMoney;
}
```

- 程序员乙

```
Account obj( x, str, y );
```



# 类与对象编程举例

- 类代码的完善与优化

- 访问权限

- 程序员甲

```
class Account // 类声明部分
```

```
{
```

```
private: // 隐藏3个数据成员
```

```
    int no; // 账号
```

```
    char name[10]; // 账户名
```

```
    float money; // 存款金额
```

```
public: // 开放3个函数成员
```

```
    void Deposit( ); // 存款
```

```
    void Withdraw( ); // 取款
```

```
    void Show( ); // 显示余额
```

```
    Account(int iniNo, char *iniName, float iniMoney); // 构造函数，公有权限
```

```
};
```

- 程序员乙



中國農業大學

閻道宏

# 7.5 对象的应用

- 对象数组

例7-7 一个正方形类Square对象数组的C++演示程序

```
1  #include <iostream>
2  using namespace std;
3
4  class Square // 定义一个正方形类Square
5  {
6  public:
7      double a; // 保存边长的double型数据成员a
8      double Area( ) // 求正方形面积的函数成员Area，内联函数
9      { return ( a*a ); }
10     Square( double x = 0 ) // 带默认形参值的构造函数，内联函数
11     { a = x; }
12 };
13
14 int main( )
15 {
16     // 定义正方形类Square的对象数组obj，定义时初始化
17     // 数组obj有3个元素，每个元素都是一个正方形对象，边长分别是2、3、4
18     Square obj[3] = { Square(2), Square(3), Square(4) };
19
20     // 显示数组中各正方形对象的边长和面积
21     for (int n = 0; n < 3; n++)
22     {
23         cout << obj[n].a << endl;
24         cout << obj[n].Area( ) << endl;
25     }
26     return 0;
27 }
```





# 7.5 对象的应用

- 对象数组

- 定义

```
Square obj[3];
```

```
Square obj[3] = { Square(2), Square(3), Square(4) };
```

```
Square obj[3] = { Square(2), Square(3) };
```

构造：有多少个数组元素，就会调用多少次构造函数函数

- 访问

对象数组名[下标]. 公有成员名

```
cout << obj[n].a << endl;
```

```
cout << obj[n].Area( ) << endl;
```

```
Square *p = &obj[n];
```

```
cout << p->a << endl;
```

```
cout << p->Area( ) << endl;
```

- 析构：有多少个数组元素，就会调用多少次析构函数



中國農業大學

閻道宏

# 7.5 对象的应用

- 对象的动态分配

```
Square *p;
```

```
p = new Square;
```

```
.....
```

```
delete p;
```

```
p = new Square( 2 );
```

```
Square *p;
```

```
p = new Square[3];
```

```
p[0].a = 2; cout << p[0].Area( ) << endl;
```

```
(p+1)->a = 3; cout << (p+1)->Area( ) << endl;
```

```
.....
```

```
delete [ ]p;
```



# 7.5 对象的应用

例7-8 一个求正方形对象内切圆面积的C++程序

- 对象

```
1 #include <iostream>
2 using namespace std;
3
4 class Square // 定义一个正方形类Square
5 {
6 public:
7     double a; // 保存边长的double型数据成员a
8     double Area() // 求正方形面积的函数成员Area, 内联函数
9     { return ( a*a ); }
10    Square( double x = 0 ) // 带默认形参值的构造函数, 内联函数
11    { a = x; }
12 };
13
14 double InnerCircleArea( Square s ) // 求正方形对象s的内切圆面积
15 {
16     double r = s.a / 2; // 内切圆直径等于正方形边长, 因此半径r = s.a / 2
17     return (3.14*r*r); // 返回内切圆的面积
18 }
19
20 int main( )
21 {
22     Square obj(10); // 定义1个正方形对象obj, 边长初始化为10
23     cout << InnerCircleArea( obj ) << endl; // 调用函数求obj的内切圆面积
24     return 0;
25 }
```

Square s( obj );



中国农业大学

阚道宏

## 7.5 对象的应用

- 对象作为函数的形参

- 值传递与常对象

```
double InnerCircleArea(Square s); //求正方形对象s的内切圆面积  
cout << InnerCircleArea( obj ) << endl;
```

```
const Square obj(2);  
cout << obj.a << endl;  
obj.a = 5; // 错误
```

```
double InnerCircleArea( const Square s ) // 常对象形参  
{  
    .....  
}
```



## 7.5 对象的应用

- 对象作为函数的形参

- 引用传递与常引用

```
double InnerCircleArea( Square &s ) // 引用传递
{
    double r = s.a / 2;
    return (3.14*r*r);
}
```

```
cout << InnerCircleArea( obj ) << endl;
```

```
double InnerCircleArea( const Square &s ) { ..... }
```



中國農業大學

閻道宏

## 7.5 对象的应用

- 对象作为函数的形参

- 指针传递与指向常对象的指针

```
double InnerCircleArea( Square *s ) // 指针传递
{
    double r = s->a / 2;
    return (3.14*r*r);
}
```

```
cout << InnerCircleArea( &obj ) << endl;
```

```
double InnerCircleArea( const Square *s ) { ..... }
```



中國農業大學

閻道宏

## 7.6 类中的常成员与静态成员

- `const`数据保护机制
- `static`静态机制
- 类中的常成员与静态成员
  - 常数据成员、常函数成员
  - 静态数据成员、静态函数成员



### 例7-9 圆形水池类CirclePool的类定义

```
1 #include <iostream>
2 using namespace std;
3 class CirclePool // 定义1个圆形水池类CirclePool
4 {
5     private:
6         double price; // 圆形水池建造单价 (10元/m2)
7         double r; // 半径 (m)
8     public:
9         CirclePool( double p1 = 0, double p2 = 0) { price = p1; r = p2; } // 构造函数
10        void SetPrice( double x ) // 设置建造单价
11        {
12            if (x <= 0) { price = 0; cout << "" << endl; } // 数据合法性检查
13            else price = x;
14        }
15        double GetPrice( ) { return price; } // 读取建造单价
16        void SetRadius( double x ) // 设置半径
17        {
18            if (x <= 0) { r = 0; cout << "" << endl; } // 数据合法性检查
19            else r = x;
20        }
21        double GetRadius( ) { return r; } // 读取半径
22        double GetCost( ) { return ( 3.14*r*r * price ); } // 计算圆形水池的造价
23    };
24
25    int main( ) // 主函数：测算养鱼池工程总造价
26    {
27        double totalCost = 0;
28        // ..... 计算长方形养鱼池的造价，代码省略
29
30        double r1, r2; cin >> r1 >> r2; // 先输入圆形清水池和污水池的半径r1、r2
31        CirclePool pool1, pool2; // 定义圆形清水池对象pool1、圆形污水池对象pool2
32
33        pool1.SetPrice( 10 ); // 设置对象pool1的建造单价
34        pool1.SetRadius( r1 ); // 设置对象pool1的半径
35        totalCost += pool1.GetCost( ); // 累加pool1的造价
36        pool2.SetPrice( 10 ); // 设置对象pool2的建造单价
37        pool2.SetRadius( r2 ); // 设置对象pool2的半径
38        totalCost += pool2.GetCost( ); // 累加pool2的造价
39
40        cout << totalCost << endl;
41        return 0;
42    }
```



# 7.6 类中的常成员与静态成员

- 常数据成员

例7-10 圆形水池类CirclePool的类定义（常数据成员price）

```
1  #include <iostream>
2  using namespace std;
3  class CirclePool // 定义1个圆形水池类CirclePool
4  {
5  private:
6      const double price; // 修改1: 声明时使用“const”关键字
7      double r; // 半径 (m)
8  public:
9      // 修改2: 在构造函数中不能直接对price赋值, 改用初始化列表的形式
10     CirclePool( double p1 = 0, double p2 = 0) : price(p1) { price = p1; r = p2; } // 构造函数
11     // 修改3: 初始化后, 不能再设置或修改price, 删除函数SetPrice
12     void SetPrice( double x ) { ..... } // 设置建造单价
13     double GetPrice( ) { return price; } // 读取建造单价
14     // ..... 其它代码省略
15 };
16
17 int main( ) // 主函数: 测算养鱼池工程总造价
18 {
19     // ..... 省略此前代码
20     double r1, r2; cin >> r1 >> r2; // 先输入圆形清水池和污水池的半径r1、r2
21     CirclePool pool1(10, r1), pool2(10, r2); // 修改4: 定义对象时必须初始化常数据成员
22
23     pool1.SetPrice(10); // 修改5: 不能再设置或修改对象pool1的建造单价
24     pool1.SetRadius( r1 ); // 可以再设置或修改对象pool1的半径
25     // ..... 省略此后代码
26 }
```



# 7.6 类中的常成员与静态成员

- 常数据成员的语法细则
  - **关键字const**。在类定义中声明常数据成员需使用关键字const进行限定，声明时不能初始化
  - **初始化列表**。类中的任何函数都不能对常数据成员赋值，包括构造函数。为构造函数添加初始化列表是对常数据成员进行初始化的唯一途径。在构造函数的函数头后面添加初始化列表：

```
构造函数名(形参列表): 常数据成员名1(形参1), 常数据成员名2(形参2), .....  
{  
    ..... // 在函数体中初始化其它数据成员  
}
```

形参1、形参2等是从形参列表中提取出来的，并在初始化列表中进行二次接力传递

- **定义对象时初始化**。定义含常数据成员类的对象时需要初始化，给出常数据成员的初始值



## 7.6 类中的常成员与静态成员

- 常函数成员

- 内联函数

```
double GetCost( ) const { return ( 3.14*r*r * price ); }
```

- 非内联函数

```
double GetCost( ) const; // 声明
```

```
double CirclePool :: GetCost( ) const // 实现  
{ return ( 3.14*r*r * price ); }
```



# 7.6 类中的常成员与静态成员

- 常函数成员的语法细则
  - 声明、定义常函数成员需在函数头后面加关键字`const`进行限定
  - 常函数成员只能读类中的数据成员，不能赋值修改
  - 常函数成员只能调用其它常函数成员。换句话说，常函数成员不能调用其它无`const`限定的函数成员，以防这些函数间接修改了数据成员
  - 通过常对象只能调用其常函数成员。换句话说，通过常对象不能调用无`const`限定的函数成员，以防这些函数间接修改了常对象的数据成员
  - 除形参个数、类型之外，还可以用关键字`const`分类中的重载函数



## 7.6 类中的常成员与静态成员

- 静态数据成员

CirclePool pool1( 10, 3 );

CirclePool pool2( 20, 5 );

|        |       |    |
|--------|-------|----|
| 静态存储区  |       |    |
| pool1. | price | 10 |
|        | r     | 3  |
| pool2. | price | 20 |
|        | r     | 5  |

(a) 对象都有各自的单价 price

|        |       |    |
|--------|-------|----|
| 静态存储区  |       |    |
|        | price | 10 |
| pool1. | r     | 3  |
| pool2. | r     | 5  |

(b) 对象共用 1 个单价 price



# 7.6 类中的常成员与静态成员

- 静态数据成员

例7-11 圆形水池类CirclePool的类定义（全局变量price）

```
1 #include <iostream>
2 using namespace std;
3 double price = 10; // 修改1: 将price定义成全局变量
4 class CirclePool // 定义1个圆形水池类CirclePool
5 {
6 private:
7     double price; // 修改2: 删除数据成员price
8     double r; // 半径 (m)
9 public:
10    CirclePool( double p1 = 0 ) { r = p1; } // 修改3: 构造函数
11    // ..... 其它代码省略
12 };
13
14 int main( ) // 主函数: 测算养鱼池工程总造价
15 {
16    // ..... 代码省略
17 }
```



# 7.6 类中的常成员与静态成员

- 静态数据成员

例7-12 圆形水池类CirclePool的类定义（静态数据成员price）

```
1 #include <iostream>
2 using namespace std;
3 class CirclePool // 定义1个圆形水池类CirclePool
4 {
5 private:
6     static double price; // 修改1: 声明时使用“static”关键字
7     double r; // 半径 (m)
8 public:
9     CirclePool( double p1 = 0 ) { r = p1; } // 修改2: 构造函数
10    // ..... 其它代码省略
11 };
12 double CirclePool :: price = 10; // 修改3: 在类外定义、初始化, 定义时加“类名::”
13
14 int main() // 主函数: 测算养鱼池工程总造价
15 {
16     // ..... 代码省略
17 }
```



# 7.6 类中的常成员与静态成员

- 静态数据成员的语法细则
  - **关键字static**。在类定义中声明静态数据成员需使用关键字static进行限定，声明时不能初始化。例如，例7-12中的代码第6行
  - **定义及初始化**。必须在类声明的大括号外面（通常是和函数成员定义一起放在类实现部分）对静态成员进行定义，定义时不能再加关键字static。定义时可以初始化。例如，例7-12中的代码第12行
  - **在同类函数成员中访问**。在同类的函数成员中访问静态数据成员直接使用其成员名访问，访问时不受权限约束
  - **在类外其它函数中访问**。在类外其它函数（例如主函数）中访问静态数据成员需以“类名::静态数据成员名”的形式访问，或通过任何一个该类对象以“对象名.静态数据成员名”的形式访问。类外访问受权限约束，只能访问公有的静态数据成员
  - **生存期及作用域**。和全局变量一样，静态数据成员是静态分配的，程序加载后立即分配内存，直到程序执行结束退出时才被释放。访问权限决定静态数据成员的作用域。私有静态数据成员具有类作用域，只能在类内访问。公有静态数据成员具有文件作用域，可以被本文件中的任何函数访问，并且可通过类声明将其作用域扩展到任何程序文件





## 7.6 类中的常成员与静态成员

- 静态函数成员

**static** double GetPrice( ); // 声明

double CirclePool :: GetPrice( ) // 实现  
{ return price; }



# 7.6 类中的常成员与静态成员

- 静态函数成员的语法规则
  - 声明时使用关键字`static`进行限定，定义时不能再使用关键字`static`
  - 静态函数成员只能访问类中的静态数据成员，因为静态函数成员可以不定义对象直接调用，而非静态数据成员只有在定义对象后才分配内存空间，才能访问。同样道理，静态函数成员不能调用其它非静态函数成员
  - 类中的其他函数成员调用静态函数成员直接使用其函数名调用，调用时不受权限约束
  - 在类外调用静态函数成员需以“类名::静态函数成员名()”的形式调用，或通过任何一个该类对象以“对象名.静态函数成员名()”的形式调用。类外调用受权限约束，只能调用公有的静态函数成员
  - 静态函数成员不能是内联函数，因为编译器在编译时会调整内联函数，此时所访问的静态数据成员可能还未初始化，因此其数据是不可靠的，此时访问会导致程序的运行结果出错



## 7.6 类中的常成员与静态成员

```
class Math
{
public:
    static double pi;
    static double sin( double x );
    static double cos( double x );
    .....
};

double Math :: pi = 3.1415926;
double Math :: sin( double x ) { ..... };
double Math :: cos( double x ) { ..... };
.....

cout << Math :: pi << endl;
cout << Math :: sin(2) << endl;
```



## 7.7 类的友元

```
class A
{
public: int x; // 公有成员x
protected: int y; // 保护成员y
private: int z; // 私有成员z
    A(int p1=0, int p2=0, int p3=0) { x = p1; y = p2; z = p3; } // 构造函数
};

void fun1( )
{
    A obj1( 2, 4, 6 ); // 定义A类对象obj1，将其3个成员分别初始化为2、4、6
    cout << obj1.x << endl; // 正确：可以访问对象的公有成员x
    cout << obj1.y << endl; // 错误：不能访问对象的保护成员y
    cout << obj1.z << endl; // 错误：不能访问对象的私有成员z
}

void fun2( )
{
    A obj2( 3, 5, 7 ); // 定义A类对象obj2，将其3个成员分别初始化为3、5、7
    cout << obj2.x << endl; // 正确：可以访问对象的公有成员x
    cout << obj2.y << endl; // 错误：不能访问对象的保护成员y
    cout << obj2.z << endl; // 错误：不能访问对象的私有成员z
}
```



# 7.7 类的友元

- 友元函数

C++语法：在类中声明友元函数

```
class 类名 // 类声明部分
{
    .....
    friend 友元函数的原型声明;
};
```

语法说明：

- 在类声明部分声明友元函数的原型，声明时使用“**friend**”关键字。声明语句可以放在大括号内的任意位置，该位置的访问权限与友元函数无关；
- 友元函数是类外的其它函数，不是类的成员；
- 友元函数可以在其函数体内访问该类对象的所有成员，不受权限约束。



中國農業大學

阚道宏

## 7.7 类的友元

```
class A
{
public: int x; // 公有成员x
protected: int y; // 保护成员y
private: int z; // 私有成员z
    A(int p1=0, int p2=0, int p3=0) { x = p1; y = p2; z = p3; } // 构造函数
    friend void fun1( ); // 声明函数fun1为类A的友元函数
};

void fun1( )
{
    A obj1( 2, 4, 6 ); // 定义A类对象obj1, 将其3个成员分别初始化为2、4、6
    cout << obj1.x << endl; // 正确: 可以访问对象的公有成员x
    cout << obj1.y << endl; // 正确: 友元函数可以访问对象的保护成员y
    cout << obj1.z << endl; // 正确: 友元函数可以访问对象的私有成员z
}
```



# 7.7 类的友元

- 友元类

```
class B
{
    .....
    void fun1( ) { ..... }
    void fun2( ) { ..... }
};
```

```
class A
{
    .....
    friend void B :: fun1( ); // 声明类B的函数成员fun1为类A的友元函数
};
```

```
class A
{
    .....
    friend class B; // 声明类B为类A的友元类
};
```



## 7.7 类的友元

- 友元类
  - 友元关系是单向的。若类A声明类B是自己的友元，并不意味着自己同时成为对方的友元，除非对方声明自己是它的友元
  - 友元关系不能传递。假设类B是类A的友元，类C又是类B的友元，这并不意味着类A和C之间存在任何友元关系，除非它们自己单独声明





# 本章要点

- 需从代码分类管理、数据类型、归纳抽象等多个维度才能准确理解类与对象的概念
- 需认真学习类与对象编程的具体语法规则
- 要深入领会面向对象程序设计通过设置访问权限来实现类封装的基本原理
- 要深入了解对象的构造与析构过程。程序员通过编写构造与析构函数来参与对象的构造与析构过程
- 要从2个不同的角度，即定义类的程序员和使用类定义对象的程序员，这样才能更容易地理解类与对象相关的各种语法知识

