第九讲

类与对象基础

（一）

目录页

Contents

1 面向对象的基本特点

- 高级语言发展
- 为什么面向对象
- 面向对象的特征

高级语言发展

- 第一代：对数学表达式有很强的运算处理能力
  代表有 Fortran，Algol 60
- 第二代：重点是如何有效地表达算法
  代表有 Pascal，C
- 第三代：引入抽象数据类型的程序设计语言
  代表有 Ada
- 第四代：面向对象程序设计语言
  代表有 Simula67，Smalltalk80、C++、Java

为什么面向对象

出发点

模仿人类认识和理解现实世界的自然思维，更直观地描述客观世界中存在的事物（对象）以及它们之间的关系。

目的

通过提高代码的可重用性，降低软件开发成本和维护成本，从而大大提高程序员的生产力。

面向对象程序设计语言

基本特点

- 高级语言
- 将客观事物看作具有属性（数据）和行为（函数）的对象
- 通过抽象找出同一类对象的共同属性和行为，形成类
- 通过类的继承与多态实现代码重用

主要特征

抽象、封装、继承、多态

抽象

什么是抽象

对具体问题/事物（对象）进行概括，抽出这一类对象的共有性质并加以描述的过程。

- 首先关注的是问题的本质及描述，其次是实现过程或细节
- 抽象包括：数据抽象和行为抽象
  - 数据抽象：
    描述某类对象的属性或状态（对象相互区别的物理量）
  - 行为抽象（或功能抽象）：
    描述某类对象的共同行为或功能特征
- 抽象的实现：类与对象

抽象示例

例：时钟的描述

- 数据抽象：

  `int hour, int minute, int second`

- 行为抽象：

  `ShowTime(), SetTime()`

封装

什么是封装

将抽象得到的数据和行为（或功能）相结合，形成一个有机的整体，即将数据与操作数据的函数进行有机结合，形成“类”，其中数据和函数都是类的成员。

为什么封装

- 封装可以增强数据的安全性，并简化编程。用户不必了解具体的实现细节，而只需要通过外部接口，以给定的访问权限，来访问类的成员。
- 提供接口供外部调用，隐藏内部细节。
- 简化被封装对象间的接口，降低系统的耦合度。

示例（时钟类）

例：时钟的描述

- 数据抽象：int hour, int minute, int second
- 行为抽象： ShowTime(), SetTime()

时钟类的声明

```
class Clock
{
  public:
   void SetTime(int h, int m, int s);
   void ShowTime();
  private:
   int hour, minute, second;
};
```

此处的分号不能省！

† public 和 private 用于指定成员的不同访问权限

继承与多态

继承

C++ 提供继承机制，允许程序员在保持原有类特性的基础上，进行更具体、更详细的说明。

多态

同一段程序能处理多种类型对象。

在 C++ 中，多态有强制多态（如类型转换）、重载多态（如函数重载、运算符重载）、类型参数化和虚函数、模板等。

2 类和对象的基本操作

- 类的声明
- 类的成员：数据与函数
- 对象的创建
- 对象成员的访问
- 成员函数的定义，内联成员函数

类和对象

类 是 C++ 面向对象程序设计的核心！

类与函数的区别

- 函数是结构化（过程式）程序设计的基本模块，用于完成特定的功能。
- 类是面向对象程序设计的基本模块，类将逻辑上相关的数据与函数封装，是对问题的抽象描述。

类的集成程度更高，更适合大型复杂程序的开发

类的声明

类必须先声明后使用

- 类的声明（定义）

```
class 类的名称
{
    public:    // 公有访问属性
        公有成员（外部接口）
    private:   // 私有访问属性
        私有成员
    protected: // 保护访问属性
        保护成员
};
```

类的声明示例

```
class Clock
{
  public:
    void SetTime(int h, int m, int s);
    void ShowTime();
  private:
    int hour, minute, second;
};
```

Example

```
Clock x;  // 声明对象

Clock y, z;
```

类的成员与访问属性

类的成员

- 数据成员（描述事物的属性）
- 函数成员（描述事物的行为/操作/功能）

成员的访问属性（访问权限控制，仅针对外部函数）

- public（公有类型、外部接口）：
  任何外部函数都可以访问公有类型的数据和函数
- private（私有类型）：
  任何外部函数都无法访问
- protected（保护类型）：
  与私有类似，区别在于继承过程中的影响不同

注意：外部函数是相对于类的成员函数而言的。

几点说明

- 如果没有指定访问属性，则缺省为私有类型
- 一般情况下，数据成员建议声明为私有类型
- 一个类如果没有任何外部接口，则无法使用
- 在声明类时，不同访问属性的成员可以按任意顺序出现，修饰访问权限的关键字也可以多次出现，但一个成员只能有一种访问属性！

Example

```
class Clock
{
  public:
    void SetTime(int h, int m, int s);
  private:
    int hour, minute, second;
  public:
    void ShowTime();
};
```

通常将公有类型的成员放在最前面，便于阅读

† 类的声明只是对类的描述，即告诉编译器Clock是什么（包含哪些数据，有什么功能），但不会给Clock分配内存。

对象的创建

声明一个类后，便可将其作为新数据类型来创建变量，即对象

对象的声明（定义）

类的名称 对象名

Example

```
Clock x;  // 声明对象
Clock y, z;
```

NOTE

- 对象所占的内存空间只用于存放数据成员，函数成员在内存中只占一份空间，不会在每个对象中存储副本。
- 声明对象，也称为类的**实例化**，即称对象是类的**实例**。

对象成员的访问

- 访问对象的成员的一般方式

对象名.数据成员名

对象名.函数成员名(参数列表)

Example

```
Clock myclock;
myclock.ShowTime();
myclock.SetTime(16,10,28);
```

NOTE

- 类的成员函数可以访问所有数据成员和函数成员。
- 外部函数只能访问公有成员。

成员函数

成员函数的定义

- 可以直接在类内部定义（适合简短函数）
- 也可以在类内部声明，然后在类外部定义（适用复杂函数）

在外部定义的一般形式

```
数据类型 类的名称::函数名(形参列表)
{
    函数体;
}
```

- 注意：在外部定义时需加上类的名称和两个连续冒号（作用域分辨符）

Example

```
void Clock::SetTime(int h, int m, int s)
{
   hour=h; minute=m; second=s;
}
```

目的对象（目标对象）

调用成员函数时，需用“.”指定调用所针对的对象，此时，该对象就称为本次调用的目的对象/目标对象/当前对象。

成员函数可以直接引用目的对象的所有数据成员，而无需使用“.”操作符。

在成员函数中，引用其它对象的数据成员和函数成员时，必须使用“.”操作符。

成员函数既可以访问目的对象的私有成员，也可以访问所在类的其它对象的私有成员。

形参带缺省值

- 成员函数的形参可以带缺省值

Example

```
class Clock
{
   public:
      void SetTime(int h=0, int m=0, int s=0);

   ... ...
};
```

NOTE

† 形参的缺省值只能在类内部设置，不能在类外部的函数定义中设置。

内联成员函数

- 内联成员函数的定义：隐式方式和显式方式
  - 隐式方式：将函数体直接放在类的声明中

Example

```
class Clock
{
   public:
      void SetTime(int h=0, int m=0, int s=0);
      void ShowTime()
        { cout<<hour<<“:”<<minute<<“:”<<second<<endl; }

   ... ...
};
```

内联成员函数

- 内联成员函数的定义：隐式方式和显式方式
  - 显式方式：在外部定义时加上关键词 inline

Example

```
class Clock
{
   public:
      void SetTime(int h=0, int m=0, int s=0);
      void ShowTime();
   ... ...
};

inline void Clock::ShowTime()
{ cout << hour << ":" << minute << ":" << second << endl; }
```

† 使用内联函数可以减少调用开销，提高效率，但只适合相当简单的函数。

例：类与对象

Example

```
#include <iostream>
using namespace std;

class Clock      // 时钟类的声明
{
  public:   // 外部接口，公有成员函数
    void SetTime(int h=0, int m=0, int s=0);
    void ShowTime();
  private:  // 私有数据成员
    int hour, minute, second;
};

// 成员函数的定义（时钟类成员函数的具体实现）
void Clock::SetTime(int h, int m, int s)
{ hour=h; minute=m; second=s; }

inline void Clock::ShowTime() // 内联成员函数
{ cout<<hour<<":"<<minute<<":"<<second<<endl; }
```

例：类与对象

ex09_class_Clock01.cpp

```
// 主函数
int main()
{
  Clock myClock; // 定义对象myClock

  cout<<"First time: "<<endl;
  myClock.SetTime();     // 设置时间为默认值
  myClock.ShowTime();    // 显示时间

  cout<<"Second time: "<<endl;
  myClock.SetTime(16,10,28);     // 设置时间为 16:10:28
  myClock.ShowTime();    // 显示时间

  return 0;
}
```

3 构造函数和析构函数

- 构造函数：数据初始化
- 构造函数的重载
- 复制构造函数
- 对象作为函数参数
- 匿名对象
- 析构函数

构造函数与析构函数

类与对象的关系类似于基本数据类型与普通变量之间的关系。

不同对象之间的主要区别：对象名与数据。

对象的初始化：定义对象时，设置数据成员的值。

构造函数：负责对象初始化。

析构函数：在对象被释放时自动调用的函数。某些特定对象使用结束后，需要进行一些清理工作，这部分工作就由析构函数负责。

构造函数与析构函数是两类特殊的成员函数，每个类都有。

构造函数

- 对象创建的过程
  - 申请内存空间，用于存放（非静态）数据成员
  - 初始化：调用构造函数对数据成员进行赋值

- 构造函数：负责数据成员的初始化
  - 构造函数的函数名与类的名称相同；
  - 构造函数没有返回类型，不带任何数据类型声明；
  - 构造函数在对象创建时会被系统自动调用；
  - 若没有定义构造函数，系统会自动生成一个缺省的构造函数；
    （形参和函数体都为空，如：`Clock() { }` ）
  - 可以是内联函数，形参可以带缺省值，也可以函数重载。

例：构造函数

```
class Clock
{
   public:
      Clock(int x, int y, int z); // 构造函数
      void SetTime(int h=0, int m=0, int s=0);
      void ShowTime();
   private:
      int hour, minute, second;
};

// 构造函数的定义
Clock::Clock(int x, int y, int z)
{ hour=x; minute=y; second=z; }

 ... ...
```

注意：构造函数前不能加数据类型或 void！

例：构造函数

ex09_class_Clock02.cpp

```
int main()
{
   Clock c1(0,0,0);  // 声明对象并初始化

   c1.ShowTime();
   c1.SetTime(16,10,28);

   Clock c2; // ERROR
   return 0;
}
```

† 如果自定义了构造函数，则系统不再提供缺省的构造函数。

形参带缺省值

ex09_class_Clock02II.cpp

```
class Clock
{
   public:
      Clock(int x=10, int y=10, int z=10); // 形参带缺省值
      ... ...
};

... ...

int main()
{
   Clock c2;  // OK, 全部使用缺省值时不要加小括号!
   Clock c2(); // ERROR
   ... ...
}
```

构造函数重载

- 构造函数重载：提供多个构造函数

ex09_class_Clock03.cpp

```
class Clock
{
   public:
      Clock(int x, int y, int z); // 构造函数
      Clock()
      { hour=0; minute=0; second=0; } // 构造函数重载
      ... ...
};

... ...

int main()
{
   Clock c1(8,8,8);  // OK
   Clock c2;  // OK, 使用不带参数的构造函数时不要加小括号!
   Clock c2(); // ERROR
   ... ...
}
```

复制构造函数

复制构造函数/拷贝构造函数：

一类特殊的构造函数，将已有对象的指复制给其它对象

```
class 类名
{  public:
     类名(类名 & 对象名); // 复制构造函数，形参必须是引用！
   ... ...
};

类名::类名(类名 & 对象名) // 复制构造函数的定义
{  函数体;  }
```

- **缺省复制构造函数：**

  系统自动生成，将已有对象的数据成员全部简单复制到指定的对象中 → 浅拷贝

注意：无论是否存在自定义的复制构造函数，缺省复制构造函数总是存在！

例：复制构造函数

Example

```
class Point      //Point 类的声明
{
   public:       //外部接口
      Point(int a=0, int b=0) { x=a; y=b; }   // 构造函数
      Point(Point &);  // 自定义复制构造函数，形参变量名可以省略
      int Getx() {return x;}
      int Gety() {return y;}
   private:      // 私有数据
      int x, y;
};

Point::Point(Point & p) // 复制构造函数的定义
{
   x=p.x; y=p.y;
   cout << "自定义复制构造函数被调用！" << endl;
}
... ...
```

复制构造函数

复制构造函数在以下几种情况下会被调用：

- 自定义复制构造函数
  - 用一个对象去初始化另一个同类的对象时
  - 若函数的形参是类的对象，调用函数时形参与实参的结合

> † 只有在进行值传递时，复制构造函数才会被调用；若传递的是引用，则不会调用复制构造函数。因此传递比较大的对象时，传递引用效率要高。

- 缺省复制构造函数：赋值语句

> † 自定义复制构造函数不影响赋值号（初始化除外）的行为！
> † 若需要修改赋值号的行为，可通过运算符重载实现。

复制构造函数

ex09_class_Point.cpp

```
int main()
{
   Point A(1,2);
   Point B(A); // 用 A 初始化 B，自定义复制构造函数被调用
   Point C=A;  // 与上面语句完全一样
   Point D;

   D=A; // 赋值，默认复制构造函数被调用！
   cout << B.getx() << endl;

   return 0;
}
```

```
viod f(Point p)  // 实参与形参结合时自定义复制构造函数被调用
{ cout << p.getx() << endl;  }
```

对象作为函数参数

- 对象可以作为成员函数和非成员函数的参数
- 实参与形参的传递方式：值传递，引用传递，地址传递

ex09_class_Clock03III.cpp

```
void printTime(Clock & c)
{
    cout << "The time is: ";
    c.ShowTime();
}
```

匿名对象

在大多数情况下，创建对象时都需要指定一个对象名，但在某些情况下，可能需要创建一个临时对象，只使用一次，这时可以使用匿名对象。

- 匿名对象的创建

```
类的名称();   // 使用不带参数的构造函数
类的名称(参数列表); // 带参数
```

NOTE

- 创建匿名对象时，使用不带参数的构造函数或形参都使用缺省值时，一定要带小括号！

例：匿名对象

ex09_class_Clock03II.cpp

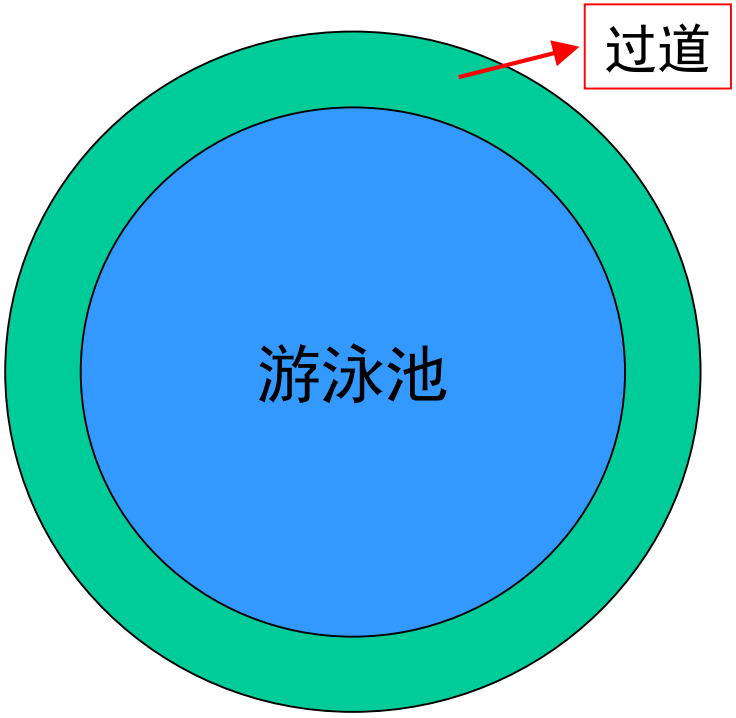
```
class Clock
{
   public:
      Clock(int x, int y, int z); // 构造函数
      Clock()
      { hour=0; minute=0; second=0; } // 构造函数重载
      ... ...
};
... ...

int main()
{
   Clock c1, c2;

   c1=Clock();  // 使用不带参数的构造函数
   c2=Clock(16,16,16); // 带参数
   ... ...
}
```

例：游泳池

例：一圆形游泳池如图所示，现在需在其周围建一圆形过道，并在其四周围上栅栏。栅栏价格为35元/米，过道造价为20元/平方米。过道宽度为3米，游泳池半径由键盘输入。要求编程计算并输出过道和栅栏的造价。



分析：可以定义一个类：圆。数据成员：圆的半径；

函数成员：计算周长与面积；

例：游泳池

Example

```
class Circle     // 声明类 Circle
{
  public:        // 外部接口
    Circle(float x)      { r=x; } // 构造函数
    float Circum();      // 计算圆周长
    float Area();        //计算圆面积
  private:       //私有数据成员
    float r;
};

// 成员函数
float Circle::Circum()   // 计算圆的周长
{ return 2*pi*r; }

float Circle::Area()     // 计算圆的面积
{ return pi*r*r; }
```

例：游泳池

ex09_class_Circle.cpp

```
int main()
{
  float x, y, z;

  cout << "输入游泳池半径：";       // 提示用户输入半径
  cin >> x;
  Circle Pool(x);          // 声明 Circle 对象
  Circle PoolRim(x+3);

  // 计算栅栏造价并输出
  y=PoolRim.Circum()*price1;
  cout << "栅栏造价为：" << y << endl;

  // 计算过道造价并输出
  z=(PoolRim.Area()-Pool.Area())*price2;
  cout << "过道的造价为：" << z << endl;

  return 0;
}
```

析构函数

析构函数：

负责对象被释放/删除时的一些清理工作。

- 析构函数的函数名由类名前加“～”组成
- 析构函数没有返回值
- 析构函数在对象生存期即将结束时被自动调用
- 析构函数不接收任何参数
- 若没有自定义析构函数，系统会自动生成一个缺省析构函数（函数体为空，如：`~Point() { }` ）

上机作业

1、设计一个名为 **Rectangle** 的类：表示矩形，这个类包括：

- 两个double型数据成员：width 和 height，分别表示宽和高
- 一个不带形参的构造函数，用于创建缺省矩形：宽和高都为 1

  `Rectangle()`
- 一个带形参的构造函数，用于创建指定宽度和高度的矩形

  `Rectangle(double x, double y)`
- 成员函数 `void setwh(double, double)`，用于更改矩形的宽度和高度
- 成员函数 `double getw()`，用于获取矩形的宽度
- 成员函数 `double geth()`，用于获取矩形的高度
- 成员函数 `double getArea()`，返回矩形的面积
- 成员函数 `double getPerimeter()`，返回矩形的周长

实现这个类，并在主函数中测试这个类：创建两个 Rectangle 对象：R1 和 R2，R1的宽和高分别为 4 和 40，R2的宽和高分别为 3.5 和 35.9，并在屏幕上输出 R1 和 R2 的面积和周长。

（程序取名 **hw09_01.cpp**）

上机作业

2、设计一个名为 **Complex** 的类，表示一个复数，这个类包括：

- 两个 double 型数据成员：x 和 y，分别表示实部和虚部
- 一个带形参的构造函数，用于创建指定实部和虚部的复数

  `Complex(double newx, double newy)`
- 成员函数 `double getx()`，获取实部
- 成员函数 `double gety()`，获取虚部
- 成员函数 `void Display()`，在屏幕上输出一个复数，如 2+3i，4-5i
- 成员函数 `double Abs()`，返回一个复数的模
- 成员函数 `Complex Minus(Complex &)`，返回当前复数与指定复数之差
- 成员函数 `Complex Multiply(Complex &)`，返回当前复数与指定复数的乘积

实现这个类，并在主函数中测试这个类：创建两个复数 z1=1.4-2.3i 和 z2=-3.5+2.7i，并在屏幕上输出 z1 的模，z1-z2 和 z1×z2 的值。

（程序取名为 **hw09_02.cpp**）

上机作业

3、设计一个名为 Integer 的类，表示整数，这个类包括：

- 一个 int 型数据成员：value，分别整数的值
- 一个带形参的构造函数，用给定的整数初始化 value
- 成员函数 void Display()，输出 value 的值
- 成员函数 int Getvalue()，返回 value 的值
- 成员函数 bool Isprime()，判断是否为素数
- 成员函数 bool Isequal(int)，判断与给定的整数是否相等
- 成员函数 bool Isequal(Integer &)，判断是否与给定的 Integer 对象相等
- 成员函数 Integer Add(Integer &)，实现两个 Integer 对象的加法
- 非成员函数 Integer Add(Integer &, int)，实现 Integer 对象与整数的加法

实现这个类，并在主函数中测试这个类：(1) 创建 Integer 对象 x，其 value 的值为 2019，判断其是否为素数；(2) 要求用户输入一个整数，然后判断 x 是否与这个整数相等；(3) 创建 Integer 对象 y，其 value 为 2109，判断 x, y 是否相等；(4) 分别计算 x 与 y 的和， x 与整数 119 的和，并输出结果。

（程序取名为 **hw09_03.cpp**）