第十讲 类与对象基础

（二）

潘建瑜@MATH.ECNU

目录页

Contents

1 类的组合 —— 组合类

- 什么是组合类
- 组合类的初始化
- 常量和引用的初始化
- 前向引用声明

类的组合

类的组合（类的聚合）

将已有的类的对象作为新的类的成员。

```
class Point    // 声明Point类
{  public:
    ... ...
   private:
    double x, y;
};
```

```
class Line // 声明Line类
{  public:
    ... ...
   private:
    Point p1, p2;
};
```

组合类的初始化

内嵌对象成员

- 在创建类的对象时，如果这个类包含其它类的对象（称为内嵌对象成员），则各个内嵌对象将首先被自动创建。

组合类的初始化

- 内嵌对象成员初始化 + 普通数据成员初始化

组合类的初始化：构造函数

类名::类名(总的参数列表)：内嵌对象1(参数), 内嵌对象2(参数), . . .

{ 类的数据成员的初始化; }

初始化列表：对内嵌对象进行初始化

- 这里的“参数”前面不用加数据类型
- 初始化列表最前面有个冒号

- 除了自身的构造函数外，
  内嵌对象的构造函数也被调用
- 构造函数调用顺序：
  - 按内嵌对象在组合类定义中出现的顺序依次调用内嵌对象的构造函数
  - 最后调用本类的构造函数
- 析构函数的调用顺序与构造函数相反

内嵌对象初始化：初始化列表

ex10_class_Line_01.cpp

```
class Point // 声明Point类
{
   public:
     Point(float x1, float y1) { x = x1; y = y1; } // 构造函数
     ... ...
   private:
     float x, y;
};

class Line  // 声明Line类
{
   public:
     Line(float x1, float y1, float x2, float y2)
         : A(x1, y1), B(x2,y2) { }  // 组合类构造函数，初始化列表
    ... ...
   private:
      Point A, B;
};
```

总参数列表中的参数需要带数据类型（形参），初始化列表则不需要

内嵌对象初始化

- 如果采用不带参数的构造函数 (或者全部使用缺省值) 初始化内嵌对象, 则不用写在初始化列表中.

ex10_class_Line_02.cpp

```
class Point
{
  public:
    Point(float x1 = 0, float y1 = 0) { x = x1; y = y1;}  // 形参都带缺省值
    ... ...
};

class Line
{
   public:
     Line(float x2, float y2): B(x2,y2) {  } // A 采用缺省值, 即 A(0,0)
   ... ...
};
```

内嵌对象初始化

- 内嵌对象也可以在构造函数体内赋值，但此时要求内嵌对象所属的类存在不需要提供数据的构造函数（如形参都带缺省值，或不带形参的构造函数）

ex10_class_Line_03.cpp

```
class Point
{
  public:
    Point(float x1=0, float y1=0) { x=x1; y=y1;} // 形参都带缺省值
    ... ...
};

class Line
{
   public:
     Line(float x1, float y1, float x2, float y2)
         { A=Point(x1, y1); B=Point(x2,y2); } // 函数体内初始化
   ... ...
};
```

常量和引用的初始化

- 数据成员中的常量和引用也需要通过初始化列表进行初始化

ex10_class_const_ref.cpp

```
class Myclass
{
  public:
     Myclass(int x, int y, int z);  // 构造函数
     ... ...
   private:
     const int a;
     int & b;
     int c;
};

Myclass::Myclass(int x, int y, int z) : a(x),b(y)
{ c=z; }
```

- 不能在类的声明中初始化任何数据成员!
- 普通数据成员可以在函数体内赋值（赋值号），也可以在初始化列表中赋值（小括号）

常量和引用的初始化

```
Myclass::Myclass(int x, int y, int z) : a(x),b(y),c(z) { } // OK
```

```
Myclass::Myclass(int x, int y, int z) : a(x),b(y),c=z { }  // ERROR
```

```
Myclass::Myclass(int x, int y, int z) : a(x),b(y) { c(z); } // ERROR
```

前向引用声明

- 类必须先定义后使用
- 若两个类互相引用，则需要使用前向引用声明

```
class ClassB;    // 前向引用声明
class ClassA       // 声明类 A
{
   public:
     void f(ClassB b);
};

class ClassB     // 声明类 B
{
   public:
     void g(ClassA a);
};
```

使用前向引用声明时，只能使用被声明的符号，而不能涉及类的任何细节。

2 结构体与联合体

- 结构体 (struct) 与 联合体 (union)

  两种特殊形态的类，为了保持与 C 语言程序的兼容性，从 C 语言继承而来 。

结构体的声明

```
struct 结构体名称
{
    public:
       公有成员
    private:
       私有成员
    protected:
       保护型成员
};
```

- 结构体与类的唯一区别：对于未指定访问控制属性的成员，在类中默认为私有成员；而在结构体中则默认为公有成员。
- C++ 中的结构体可以含数据和函数，而 C 语言中的结构体只能含数据。

联合体的声明

```
union 联合体名称
{
    数据成员1
    数据成员2
    ... ...
};
```

```
union Mark
{
    char grade;   // 等级制
    bool pass;    // 是否通过
    int score;    // 百分制
};
```

- 联合体的所有成员共享一个存储单元
- 同一时间段，联合体的所有成员中至多一个有意义
- 联合体中成员的默认访问属性是公有类型
- 联合体一般只存数据，不含函数成员

3 对象生存期

- 局部变量与“全局”变量
- 对象的生存期
- 对象的静态成员（数据、函数）

数据成员与成员函数中的局部变量

- 数据成员可以被类中的所有成员函数访问（可以看作是这个类中的全局变量）
- 成员函数中声明的变量是局部变量，只在该函数中有效
- 如果成员函数中声明了与数据成员同名的变量，则在该函中数据成员被屏蔽（类似于同名的全局变量和局部变量）

示例

ex10_class_01.cpp

```
class Point
{
   public:
      Point(double a, double b) { x=a; y=b;}
      void mycout()
      {   cout << x << endl; // 数据成员 x
          int x = 10;
          cout << x << endl;  // 局部变量 x
      }
   private:
     int x, y;
};

int main()
{
   Point A(4,5);
   A.mycout();
   return 0;
}
```

对象的生存期

与普通变量一样，对象有静态和动态生存期

- 动态生存期：当对象所在的程序块执行完后即消失
- 静态生存期：生存期与程序的运行期相同，即一直有效

- 全局对象：静态生存期
- 动态对象：动态生存期

静态成员

为什么静态成员

一般情况下，同一个类的不同对象分别有自己的数据成员，名字一样，但值不同，互不相干。但有时希望某些数据成员为所有对象所共有，这样可以实现数据共享。

静态成员

- 全局变量可以达到共享数据的目的，但安全性得不到保证：任何函数都可以自由修改全局变量的值，很有可能偶然失误，全局变量的值被错误修改，导致程序失败。
- C++ 提供静态数据成员，用于在同一个类的多个对象之间实现数据共享。
- 静态数据成员由特定的成员函数（静态成员函数）来维护。

静态数据成员

- 关键字 static
- 该类的 所有对象共同使用和维护 该成员
- 静态变量可以 初始化，但必须在 类外部 初始化

```
class Point
{
   public:
     ... ...
   private:
     int x, y;
     static int count; // 引用性声明
};

int Point::count = 0; // 静态数据成员的定义和初始化
... ...
```

静态数据成员（续）

关于静态数据成员的几点说明

- 静态数据成员为类的所有对象共有，不属于任何特定对象
- 静态数据成员在内存中只占一份空间
- 只要在类中定义了静态数据成员，即使不定义对象，也为静态数据成员分配空间，它可以被引用
- 如果静态数据成员没有初始化，则系统会自动赋予初值 0
- 静态数据成员既可以通过对象名引用，也可以通过类名来引用，即：
  对象名.静态成员名 或 类名::静态成员名

ex10_static_01.cpp

静态成员函数

- 用关键字 static 修饰，为整个类所有
- 调用方式：类名::静态函数成员名
- 没有目标对象，所以不能对非静态数据成员进行默认访问
- 静态成员函数一般用于访问静态数据成员，维护对象之间共享的数据

```
class A
{
   public:
     ... ...
     static void fun(A a);
   ... ...
};
```

静态成员函数声明时需加 static，
但定义时不能加 static

实际上也允许通过对象名调用静态成员函数，但此时使用的是类的性质，而不是对象本身。

静态成员函数的注记

- 静态成员函数可以直接访问静态成员，但不能直接访问非静态成员

```
// 静态成员函数中有以下语句：
cout << height; // 若 height 是 static，则合法
cout << width;  // 若 width 是非静态数据成员，则不合法
```

- 静态成员函数访问非静态数据成员时，需指明对象

```
cout << p.width;  // 访问对象 p 的非静态数据成员 width
```

ex10_static_02.cpp

建议：静态成员函数只用来处理静态数据成员，逻辑清楚，不易出错。

4

友元

- 什么是友元关系
- 为什么要友元
- 友元函数、友元类
- 关键字 friend

友元关系

友元关系

- 提供一种类与外部函数之间进行数据共享的机制。
- 通俗说法：一个类主动声明哪些类或外部函数是它的朋友，从而给它们提供对本类成员的访问特许，即可以访问私有成员和保护成员。

- 好处：友元提供了一种数据共享的方式，提高了程序效率和可读性。
- 坏处：友元在一定程度上 破坏了数据封装和数据隐藏机制。

- 友元包括：友元函数 与 友元类
- 友元类的所有函数成员都是友元函数

友元函数

- 用关键字 friend 修饰
- 可以是普通函数或其它类的成员函数
- 友元函数可以 通过对象名直接访问 私有成员和保护成员

```
class Point
{
   public:
     ... ...
     friend float dist(Point & p1, Point & p2);
   private:
     int x, y;
};
float dist(Point & p1, Point & p2)
{  double x=p1.x-p2.x, y=p1.y-p2.y;
   return sqrt(x*x+y*y);
}
```

ex10_friend_Point.cpp

友元类

- 用关键字 friend 修饰
- 友元类的所有成员函数都是友元函数

```
class A
{
   public:
     ... ...
     friend class B; // 声明 B 是 A 的友元类
   ... ...
};
```

除非确有必要，一般不建议把整个类声明为友元类，而只需将确实有需要的成员函数声明为友元函数，这样更安全。

友元关系的特点

友元关系的特点

- 友元关系不能传递
- 友元关系是单向的
- 友元关系不能被继承

面向对象程序设计的一个基本原则是封装性和信息隐蔽，而友元是对封装原则的一个小的破坏。但是它能有助于数据共享，提高程序的效率。在使用友元时，要注意它的副作用，不要过多地使用友元，只有在使用它能使程序精炼，并能大大提高程序的效率时才用友元，否则可能得不偿失。

第十讲上机作业

1、 (a) 设计一个名为 **Score** 的类，表示成绩，这个类包括：

- 两个 int 型数据成员：math 和 eng，分别表示数学成绩和英语成绩
- 一个带两个形参的构造函数，用给定的分数初始化 math 和 eng
- 成员函数 show()，输出数学成绩和英语成绩

(b) 设计一个名为 **Student** 的类，表示学生，这个类包括：

- 两个数据成员：stuid（int 型，表示学号）和 mark（Score对象）
- 一个带三个形参的构造函数，对数据成员进行初始化
- 成员函数 stushow()，输出学号和相应的成绩

实现这两个类，并在主函数中测试这个类：创建一个学生：学号为 2017007，数学成绩为 98，英语成绩为 85，在屏幕上输出该生的学号和成绩。（程序取名为 **hw10_01.cpp**）

注：若没有特别说明，所有数据成员都是 private，所有函数成员都是 public

第十讲上机作业

2、设计一个名为 **MyDate** 的类，表示日期，这个类包括：

- 三个 int 型数据成员：year，month，day，分别表示年、月、日
- 一个带一个形参的构造函数，用给出的自1970年1月1日0时流逝的秒数创建一个 MyDate 对象，如果没有给定时间，则缺省为当前时间：

```
MyDate(unsigned long second=time(0))
```

- 一个带三个形参的构造函数，用给定的年月日创建一个 MyDate对象：

```
MyDate(int y, int m, int d)
```

- 成员函数 showDay()，在屏幕上输出对象中的年月日

实现这个类，并在主函数中测试这个类：创建表示当前时间的 MyDate 对象 d1 和 d2(3456201512)，然后输出它们所对应的日期。（程序取名为 **hw10_02.cpp**）