第5讲

共用数据的保护、静态成员和友元

主讲人：赵文彬

本次课主要内容

- 共用数据的保护
- 静态成员
- 友元

共用数据的保护

- C++中存在数据共享
  - 实参与形参，变量与其引用，数据及其指针
- 出现误操作而改变数据

- 常量
  - 数据能在一定范围内共享
  - 保证不被任意修改

共用数据的保护

- 常对象成员
  - 常数据成员
    - 用const声明常数据成员

      const int hour;
  - 注意
    - 只能通过构造函数的参数初始化表对常数据成员进行初始化，任何其他函数都不能对常数据成员赋值。

      Time::Time(int h)

      { hour = h;}

Time::Time(int h): hour(h) { }

共用数据的保护

- 常对象成员
  - 常成员函数
    - 一般形式

      类型名 函数名 （参数表） const;

      void get_time() const;
  - 注意
    - const是函数类型的一部分，声明和定义中都要加关键字const，调用时不用。
    - 如果将成员函数声明为常成员函数，则只能引用本类的数据成员，而不能修改它们。
    - 常成员函数不能调用另一个非const成员函数。

共用数据的保护

- 常对象
  - 定义（一般形式）

    类名 const 对象名（实参表） 或

    const 类名 对象名（实参表）
  - 说明
    - 定义常对象时必须同时进行初始化，且之后不能再改变。
    - 除了系统自动隐式调用的构造函数和析构函数外，常对象只能调用类的常成员函数，而不能调用普通成员函数。

共用数据的保护

- 指向对象的常指针
  - 定义

    一般形式—— 类名* const 指针变量名

    Time * const ptr1;
  - 说明
    - 将指针变量声明为const型，则赋值后该指针指向不再改变。但所指向对象的内容可以改变。
    - 常用于函数的形参，不允许在函数执行过程中改变指针变量的值。

共用数据的保护

- 指向常对象的指针变量
  - 定义

    一般形式—— const 类型名 * 指针变量名；

    const Time * ptr2;
  - 说明
    - 指针的指向可以改变。
    - 指针可以指向const对象，也可以指向非const对象，但常对象必须由指向常对象的指针变量指向它。
    - 只能通过指针引用对象的成员，而不能改变其值。

（这里所说指针均为指向常对象的指针）

共用数据的保护

- 举例

```
Time t1(10,20,30);
const Time t2(11,21,31);
Time * const pt1;
const Time * pt2;
Time * pt3;
```

```
pt1 = t1;   合法
pt1 = t2;   不合法
```
不合法

```
pt2  = t2;  合法
pt2  = t1;  合法
```
合法

```
pt3  =  t1; 合法
pt3  = t2;  不合法
```
不合法

共用数据的保护

- 对象的常引用
  - 定义

```
Time t1(10,20,30);
cosnt Time &t = t1;
```

  - 说明
    - 定义了常引用后，不能通过引用变量改变被引用变量的值。
    - 常用于函数形参，保证在函数执行过程中实参对象的值不被修改。

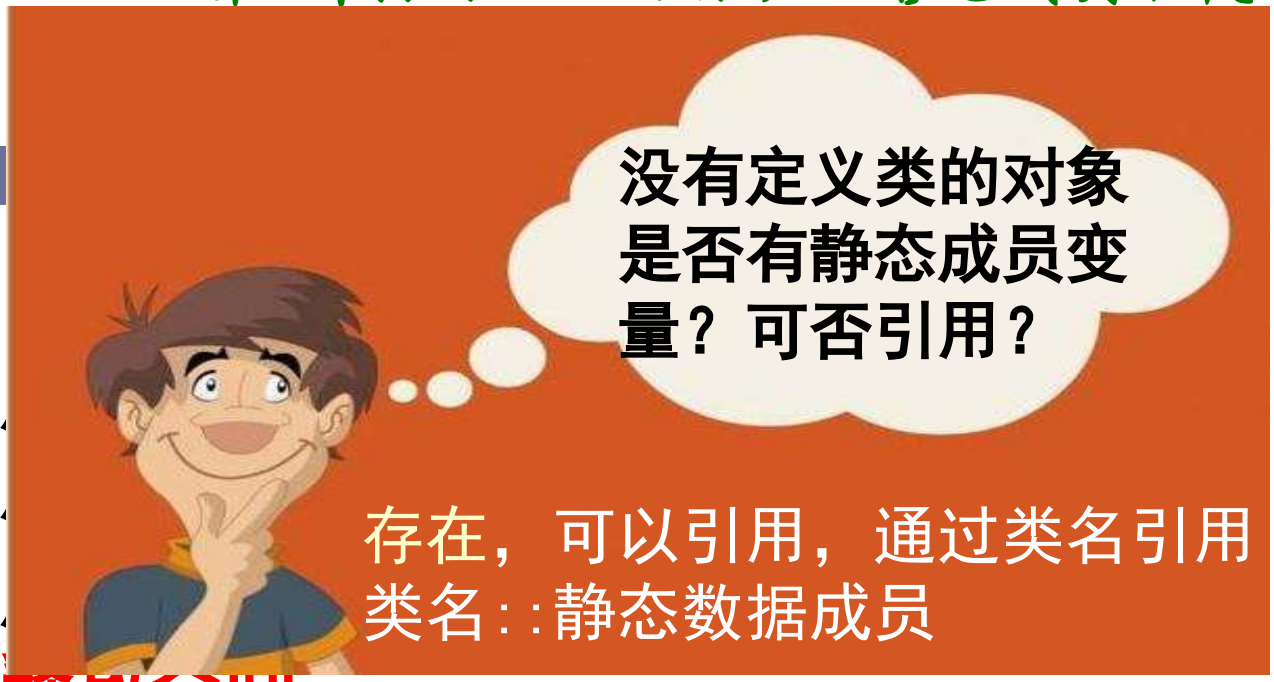
静态成员

- 回顾
  - 静态变量
  - 全局变量
- 静态数据成员
- 静态成员函数

静态成员

- 静态数据成员
  - 如果希望各对象中的数据成员的值是一样的，就可以把它定义为静态数据成员。

```
class Box
{
        int volume();
private:
        static int height;
        int width;
        int length;
};
```



- 说明
  - 静态数据
    不为静态
  - 静态数据
    结束时才释放空间。
  - 静态数据成员可以被初始化，但必须在类外。

```
int Box:: height = 10;
```

一般形式：

数据类型 类名::静态数据成员名 = 初值

静态成员

- 说明
  - 不能用参数初始化表对静态数据成员初始化。

    Box(int h, int w, int len): height (h)

    
  - 在类外静态数据成员可以通过对象名引用，也可以通过类名引用。

    Box a;

    Box::height ⟺ a.height

注意：静态数据成员的作用域仅限于所在类的作用域！

静态成员

- 静态成员函数
  - 在类中声明函数的前面加static就成了静态成员函数。

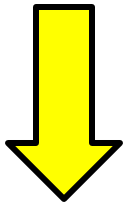
    static int volume();
  - 静态成员函数是类的一部分而不是对象的一部分。
  - 若要在类外调用静态成员函数，需要用类名和域运算符

    类名::静态成员函数名（实参）

    Box :: volume ();

静态成员

- 静态成员函数与非静态成员函数的区别
  - 非静态成员函数有this指针
  - 静态成员函数没有this指针



- 静态成员函数不能访问非静态数据成员。
- 用于访问静态数据成员。

静态成员

- 用途
  - 若需类的一些行为不需要实例化对象，就可以直接调用，则把这些行为定义为静态函数。
  - 例如
    - 定义了数学函数的类Math，包括一些常用成员函数用于求正弦、余弦等。
    - Math::sin(x);
    - Math math;

      math.sin(x);

友元

友元函数

将普通函数声明为友元函数

```
#include <iostream>
using namespace std;
class Time
{public:
        Time ();
        friend void display();
private:
        int hour,minu,sec;};
Time::Time()
{       hour=11;
        minu=11;
        sec=11;}
```

```
void display()
{Time t;
cout<<t.hour<<":"<<t.mi
nu<<":"<<t.sec<<endl;}
void main()
{
      display();
}
```

友元

友元成员函数

```
#include <iostream>
using namespace std;
class Date;
class Time
{public:
        Time(int,int,int);
        void display(Date &);
 private:int hour,minu,sec;};
class Date
{public:
        Date(int,int,int);
        friend void Time::display(Date &);
 private:int year,month,day;};
```

友元

```
Time::Time(int h, int m, int s)
{hour=h;      minu=m;      sec=s;}
Date::Date(int y,int m,int d)
{year=y;      month=m;     day=d;}
void Time::display(Date &d1)
{cout<<d1.year<<"."<<d1.month<<"."<<d1.day<<endl;
  cout<<hour<<":"<<minu<<":"<<sec<<endl;}
void main()
{       Time t1(23,31,24);
        Date d1(2014,2,3);
        t1.display(d1);
}
```

友元

- 友元类
  - friend 类名

```
#include <iostream>
using namespace std;
class Time;
class Date
{ public:
        Date(int,int,int);
        void display(Time &);
 private:
        int year,month,day;};
class Time
{public:
        Time(int,int,int);
        friend Date;
 private:
        int hour,minu,sec;};
```

友元

```
Time::Time(int h,int m,int s)
{hour=h;      minu=m;      sec=s;}
Date::Date(int y, int m, int d)
{year=y;      month=m; day=d;}
void Date::display(Time &t)
{cout<<year<<"/"<<month<<"/"<<day<<endl;
 cout<<t.hour<<":"<<t.minu<<":"<<t.sec;}
void main()
{       Date d1(2014,2,5);
        Time t1(11,1,1);
        d1.display(t1);

}
```

友元

- 说明
  - 友元是**单向**的而不是双向的；
  - 友元的关系**不能传递**；
  - 友元有助于**数据共享**，可**提高**程序的**效率**，但却**破坏**了面向对象程序设计的**封装原则**，在实际使用过程中要选择一个数据共享和信息隐蔽之间的恰当的**平衡点**。

小结

- 常量
- 静态　　数据共享
- 友元

- 作业
  - 今天没有

课堂练习

- 先定义一个点类，类名为point，用课堂教师演示的方式添加类，即类的定义在头文件中，另外有一个描述类成员函数实现的cpp文件，还有一个主函数的文件。
  - 将其三维坐标定义为私有成员，通过构造函数为其初始化，并在构造函数和析构函数中有输出语句，以便于从运行结果看出构造函数析构函数的运行。
  - 写三个构造函数用于重载，包含一个默认构造函数。
  - 定义一个对象指针，并通过该指针完成对点对象坐标的输入和输出。
  - 定义对象数组，观察构造函数和析构函数调用的顺序。