# 面向对象方法与C++程序设计

## 第2章　　类与对象

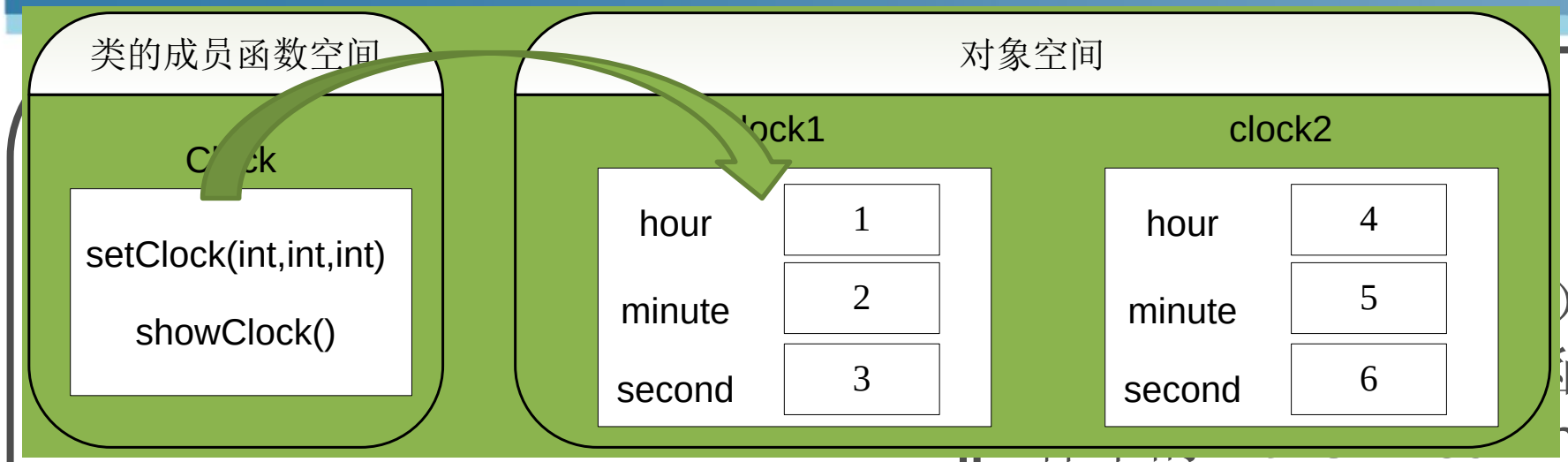
大连理工大学
主讲人-赵小薇

# this关键字



- 每个对象有自己独立的数据空间，但是类的成员函数只存储一份，为所有对象共享；
- 当通过对象调用非静态成员函数（静态情况下一节介绍）时，需要把调用对象的地址也传递给成员函数，以确定成员函数要处理的数据是哪一个对象的数据，成员函数通过this指针接收调用对象的地址，所以每个成员函数（非静态的）都有一隐含的指针变量this。

# this关键字




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

即给c1的hour赋值。

```
void Clock::setClock (Clock *this, int h, int m, int s) {
    this->hour=h;     this->minute=m;
    this->second=s;}
```

# this关键字



有了this指针，把形式参数与类数据成员命相同的名字：

```
void Clock::setClock (int hour, int minute, int second) {
    this->hour= hour;
   this->minute= minute;
this->second= second;
}
```

在类的非静态成员函数中返回类调用对象本身的时候，直接使用 “return *this”，在第4章运算符重载部分会有应用。

# static成员



在类内数据成员的声明前加上关键字static，该数据成员就是类内的静态数据成员，静态数据成员也被称作是类的成员。

- 无论这个类的对象被定义了多少个，静态数据成员在程序中也只有一份拷贝，由该类型的所有对象共享访问；
- 对该类的多个对象来说，静态数据成员只分配一次内存；
- 静态数据成员的值对每个对象都是一样的，而非静态数据成员，每个类对象都有自己的拷贝。

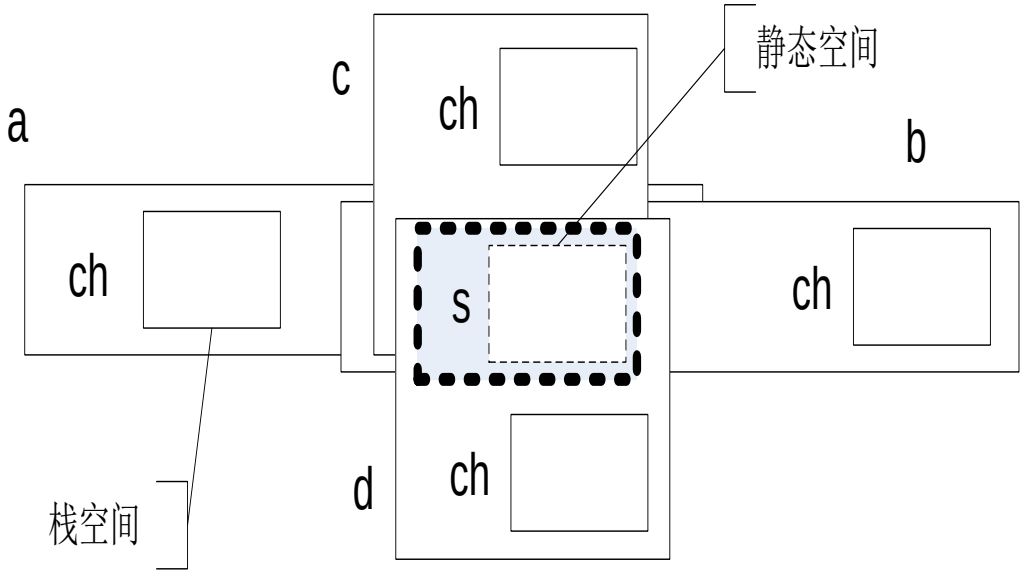
# Static空间

对于类X的定义：

```
class  X {
public:
char  ch ;
static  int s;
 };
 int X::s = 0 ;
```

对象a、b、c、d的定义：

```
void f( )
 { X  a , b , c , d ;
}
```




sizeof(X)？

静态数据成员初始化：

**＜数据类型＞＜类名＞::＜静态数据成员名＞=＜值＞**

如：int X::s = 0 ;

- 初始化在类体外进行，而前面不加static；
- 初始化时不加该成员的访问权限控制符public、protected、private等；
- 若未在类外初始化，程序连接时会出现错误
- sizeof运算符不包含静态数据成员的空间

# 静态数据成员访问



静态数据成员访问形式：

**＜类对象名＞.＜静态数据成员名＞**

**或 ＜类类型名＞::＜静态数据成员名＞**

静态数据成员s可以有5种访问办法：

X::s（没有定义对象a、b、c、d，X::s也是正确的）；

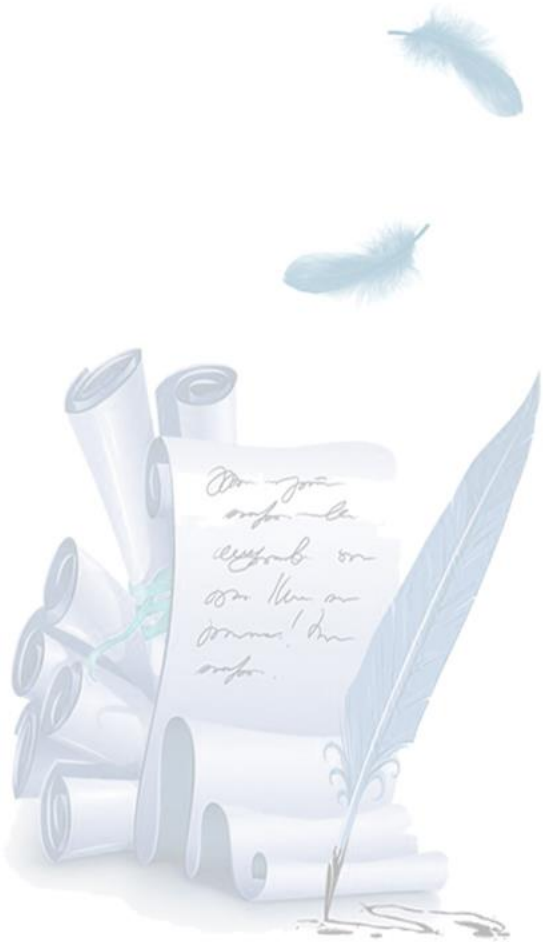
a.s、b.s、c.s、d.s，他们都代表访问同一个变量空间，通过一个改变值，其他方式获取的值全部改变。

静态数据成员和普通数据成员一样遵从public、protected、private访问规则，若上述X的静态数据成员是私有的，X::s、a.s、b.s、c.s、d.s访问s都是错误的，因为不可见。

# static成员与全局变量



同全局变量相比，使用静态数据成员有两个优点：
1.是静态数据成员没有进入程序的全局名字空间，因此不存在与程序中其它全局名字冲突的可能性；
2.是可以实现信息隐藏，静态数据成员可以是private成员，而全局变量不能。



# 静态成员函数



- 成员函数前面加static修饰成静态成员函数
- 静态成员函数声明时前面加static关键字，在类外写函数实现时不需要关键字static
- 静态成员函数是不依赖于对象成员的函数，通常用来定义一些工具函数



- 普通的成员函数一般都隐含了一个this指针，静态成员函数由于不是与任何的对象相联系，因此它不具有this指针；
- 静态成员函数可以访问静态数据成员和访问静态成员函数，但是静态成员函数不能访问非静态成员函数和非静态数据成员；
- 非静态成员函数可以任意地访问静态成员函数和静态数据成员。

# Const关键字



- 数据的安全性可以通过访问控制权限管理；
- const控制读写权限；

- 类中某些数据成员的值是不需要改变的，如数学类的π、物理类的g等都是常量，可以用关键字const来声明常数据成员。
- 因为常数据成员的值是不能改变的，所以，定义对象时必须初始化，需要通过构造函数的初始化列表对常数据成员进行初始化。

# Const数据成员



```
class  Clock
{
private :
    const int hour;                 //常量数据
    int  minute, second;              //非常量数据
public :
    Clock (int h, int m, int s):hour(h),minute(m),second(s){ }
};
```

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

Clock (int h, int m, int s) {hour=h; minute=m; second=s; } ×

Clock (int m, int s) { minute=m; second=s; } ×

成员函数可以引用本类中的非const数据成员，可以修改它们；成员函数可以引用本类中的const数据成员，不可以修改它们。

# Const函数



定义的类的成员函数不改变类的数据成员（比如打印函数），这些函数是“只读”函数，加上const关键字进行标识，标识为常成员函数。声明常成员函数格式如下：

**类型 成员函数名(参数表) const;**

```
class  Clock{
private :
    int hour, minute, second;
public :
    void setClock (int h, int m, int s){
          hour=h;  minute=m; second=s;   }
    void showClock ( ) const{    //常成员函数
        cout<<hour<<minute<<second;  }
};
```

```
void Clock::showClock ( ) const{
    minute++;
    cout<<hour<<minute<<second;}
```

改变数据成员，编译出错。；

- 常成员函数可以引用const数据成员，也可以引用非const的数据成员，只要在常成员函数不改变数据成员即可；
- const数据成员可以被const成员函数引用，也可以被非const的成员函数引用，只要const数据成员不被修改即可；
- 常成员函数不能调用另一个非常成员函数，因为非常成员函数是可以改变数据成员的，这样常成员函数就间接改变了数据，违背了常成员函数的定义规则。

# Const对象



定义常对象的一般形式为：

**类名 const 对象名[(实参表列)];**

或

**const 类名 对象名[(实参表列)];**

const Clock c1(12,34,46);

一个对象被声明为常对象，则不能调用该对象的非const型的成员函数（除了由系统自动调用的隐式的构造函数和析构函数）。例如，对于例3.11中已定义的Circle类，定义常对象：

const Circle c1(1.2,3.4,3,3.14); //定义常对象c1

c1. getRadius(); //调用常对象c1中的非const成员函数，非法

# 举例

计算一组学生的总成绩和平均成绩



```
class Student{
private:
   char name[20];
   int age;
   float score;
public:
    Student(char * n, int a, int s){
       strcpy(name,n);   age=a;
       score=s;sum+=score;  }
    static float sum;   //声明静态
};
float Student::sum=0;   //初始化
```

```
int main(){
  Student
stud[3]={ Student("zhang
san",20,60), Student("Li
si",19,70), Student("Wang
wu",18,78)  };
float average;
average=Student::sum/3;
   //等价于对象名stud[1].sum
cout<<"sum="<<Student::su
m<<"   average="<<average;
return 0;
}
```