



桂林电子科技大学
GUILIN UNIVERSITY OF ELECTRONIC TECHNOLOGY



人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE

第五讲 类和对象

赵彬

zhaobin@guet.edu.cn

5.1 动态内存分配

在C++程序中，所有内存需求都是在程序执行之前通过定义所需的变量来确定的。但是可能存在程序的内存需求只能在运行时确定的情况。例如，当需要的内存取决于用户输入。在这些情况下，程序需要动态分配内存，C++语言将运算符new和delete合成在一起。动态内存分配具有以下特点：

- 不需要预先分配内存空间
- 分配的控件可根据程序需要扩大/缩小
- 动态内存分配的生存期由编程者决定
- 在堆中开辟，由程序员开辟和释放

栈区 (stack) — 由编译器自动分配释放，存放函数的参数值，局部变量的值等

堆区 (heap) — 一般由程序员分配释放，若程序员不释放，程序结束时可能由操作系统回收

5.1 动态内存分配

5.1.1 new的用法

(1) new运算的过程

当我们使用关键字new在堆上动态创建一个对象时，它实际上做了三件事：获得一块内存空间、调用构造函数、返回正确的指针。如果我们创建的是简单类型的变量，例如int或float类型，那么第二步会被省略。

(2) 开辟单变量地址空间

使用new运算符时必须已知数据类型，new运算符会向系统堆申请足够的存储空间。如果申请成功，就返回该内存块的首地址，如果申请不成功，则返回零值。其基本语法形式如下：

```
指针变量名=new 类型标识符;
```

或

```
指针变量名=new 类型标识符（初始值）;
```

或

```
指针变量名=new 类型标识符 [内存单元个数];
```

5.1 动态内存分配

5.1.1 new的用法

指针变量名=new 类型标识符;

或

指针变量名=new 类型标识符 (初始值) ;

或

指针变量名=new 类型标识符 [内存单元个数];

格式1和格式2都是申请分配某一数据类型所占字节数的内存空间，其中前者只对变量进行申请，后者则同时将初值存放到该内存单元中；格式3可以同时分配若干个内存单元，相当于形成一个动态数组。

例：

```
int *a=new int;           //动态分配一个int
int *pi=new int(1);       //动态分配一个int，初始化为1
int *pa=new int[1];       //动态分配一个数组，数组大小为1
```

5.1 动态内存分配

5.1.1 new 的用法

(3) 开辟数组空间

对于数组进行动态分配的格式为：

指针变量名=new 类型名[下标表达式];

例：

```
int *prt;  
prt=new int[3];
```

5.1 动态内存分配

5.1.2 delete的用法

当程序不再需要由new分配的内存空间时，可以使用delete释放这些内存空间。delete []的方括号中不需要填数组元素数，系统自知。即使写了，编译器也忽略。其语法形式如下：

(1) 删除单变量地址空间

delete指针变量名;

例：

```
int *a=new int;  
delete a;
```

(2) 删除数组空间

delete [] 指向该数组的指针变量名;

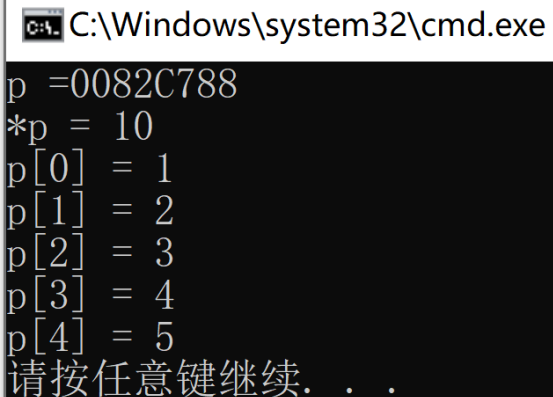
例：

```
int *a=new int [15];  
delete [] a;
```

5.1 动态内存分配

【例 1】动态内存分配

```
#include<iostream>
#include<stdio.h>
using namespace std;
int main()
{
    int* p = new int;
    *p = 5;
    *p = *p + 5;
    cout<<"p ="<< p <<endl;
    cout<<"*p = "<<*p<<endl;
    delete p;
    p = new int[5];
    for(int i=0; i<5; i++)
    {
        p[i] = i + 1;
        printf("p[%d] = %d\n", i, p[i]);
    }
    delete[] p; //释放5个指针
    return 0;
}
```



```
C:\Windows\system32\cmd.exe
p =0082C788
*p = 10
p[0] = 1
p[1] = 2
p[2] = 3
p[3] = 4
p[4] = 5
请按任意键继续. . .
```


5.2 面向对象的特征

5.2.1 类和对象

- 对象可以是一个有形的具体存在的事物，也可以是一个无形的，抽象的事物
- 对象一般具有两个因素：**属性**（attribute）和**行为**（behavior）。属性描述的是事物的静态特征，比如一个人的姓名，年龄，身份证号等信息；行为（或方法）描述事物具有的动态特征，比如一个人走路、说话，一辆汽车向前行驶等
- “类”是一组具有相同属性和行为的对象的抽象，**类**是一种自定义数据类型，对象是类的**实例**

5.2 面向对象的特征

5.2.1 类和对象

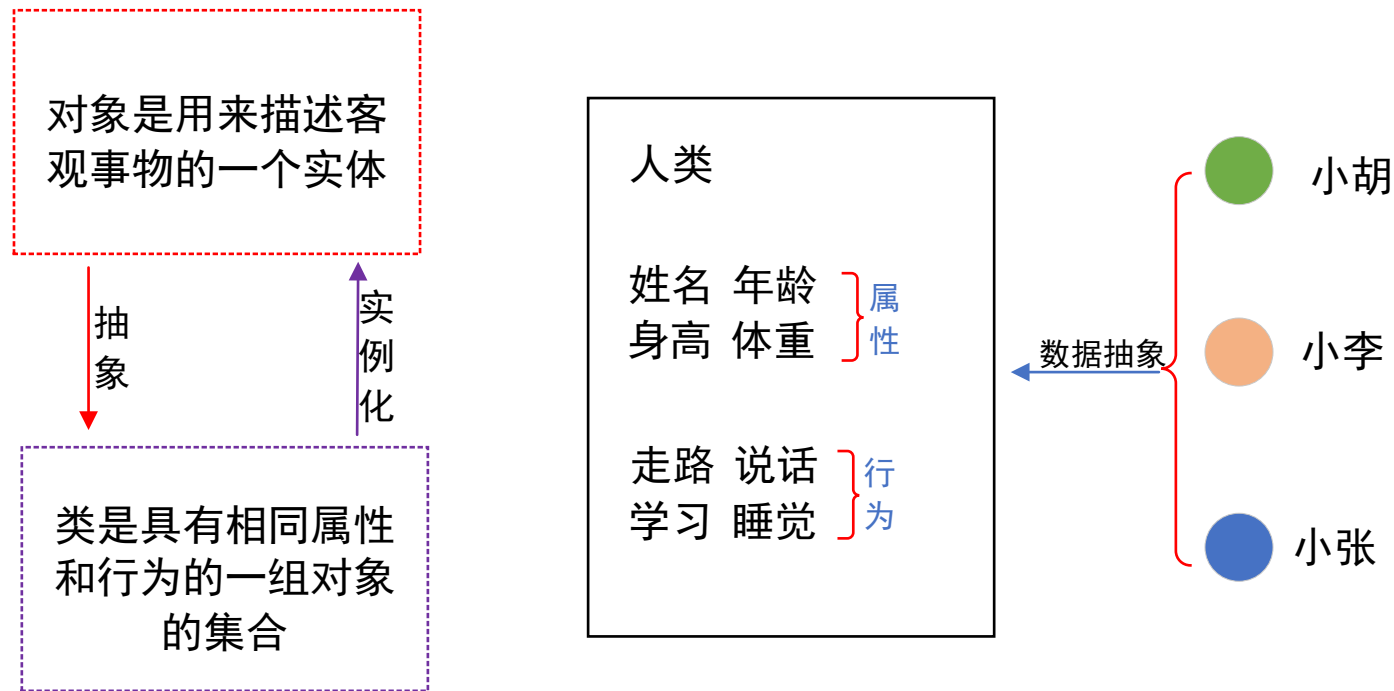


图1 类和对象的关系

面向对象程序设计将数据和数据的操作封装在一起，作为不可分割的整体，而各个对象之间通过传递不同的消息来实现相互协作。在C++中，对象是由数据和函数组成，调用对象的函数就是向该对象发送一条消息，要求该对象完成某一功能。

5.2 面向对象的特征

5.2.2 抽象

- 抽象(abstract)是对具体问题（对象）进行概括，抽出一类对象的公共性质并加以描述的过程
- 抽象是对复杂世界的简单表示，抽象不是对象的全部信息的描述，而只强调感兴趣的信息，忽略与主题无关的信息

例：学生成绩管理程序（姓名、学号和成绩等）↔学生健康管理程序（身高、体重）

- 面向对象程序设计中的抽象包括两个方面：**数据抽象**和**代码抽象**（或称为**行为抽象**）。前者描述某类对象的属性或状态，后者描述了某类对象的公共行为特征或具有的公共功能
- **例：**时钟程序----存储时间（时分秒-**数据抽象**）↔显示时间、设置时间等功能（**行为抽象**）

编写程序的目的就是描述和解决现实世界中的问题，将现实世界中的对象和类如实反映在程序中。

5.2 面向对象的特征

5.2.3 封装

- 封装(encapsulation)是将抽象得到的数据和行为放在一起，形成一个实体——对象，并尽可能隐蔽对象的内部细节。
- 对象之间相互独立，互不干扰。对象只留有少量的接口供外部使用，数据和方法是隐藏的。
- 封装具有以下特点：
 - 封装必须提供接口供外部使用，简化了对象的使用。
 - 封装在内部的数据和方法对外不可见，其他对象不能直接使用。
 - 封装可以通过继承机制实现代码重用。

5.2 面向对象的特征

5.2.4 继承

- 继承 (inheritance) 是指特殊类的对象拥有其一般类的全部属性与行为，被继承的类称为基类，通过继承得到的新类称为派生类。
- C++提供继承机制更符合现实世界的描述，继承机制具有传递性，可以被一层一层的不断继承下去，实现代码重用和可扩充性，减轻程序开发工作的强度，提高程序开发的效率。

货车类



SUV类



继承

继承

汽车类

5.2 面向对象的特征

5.2.5 多态

- 多态 (polymorphism) 是指不同的对象收到相同的信息时执行不同的操作。所谓消息是指对类的成员函数的调用，不同的操作是指不同的实现，也就是调用了不同的成员函数。
- C++语言支持两种多态性，即编译时的多态性（静态多态性）和运行时的多态性（动态多态性）。编译时的多态是在编译的过程中确定同名操作的具体操作对象，比如函数重载（包括运算符重载），根据参数的不同，执行不同的函数。运行时的多态性是在运行过程中才动态地确定所操作的具体对象，C++通过使用虚函数 (virtual) 来实现动态多态性。
- 这种确定具体操作对象的过程叫做绑定（也叫联编），绑定工作在编译链接阶段完成称为静态绑定，在程序运行阶段完成称为动态绑定。



移动



5.2 面向对象的特征

5.2.6 消息

一个对象向另一个对象发出的服务请求被称为“消息”，也可以说是一个对象调用另一个对象的函数。当对象接收到消息时，就会调用相关的方法，执行相应的操作。

● 消息具有以下三个性质：

- 同一个对象可以接收不同形式的多个消息，作出不同的响应；
- 相同形式的消息可以传递给不同的对象，所作出的响应可以不同；
- 对消息的响应并不是必须的，对象可以响应消息，也可以不响应。



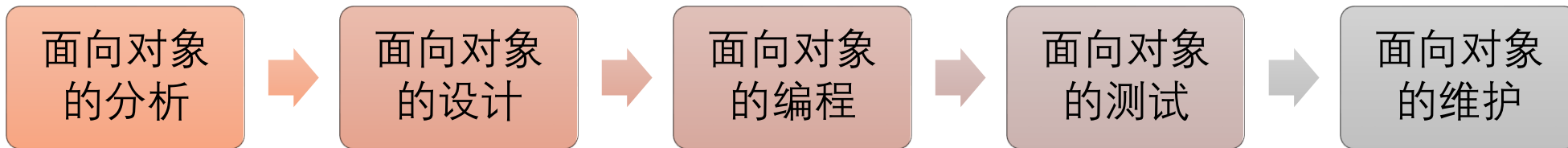
询问问题

消息



5.2 面向对象的特征

5.2.7 面向对象的软件开发



- 1、在**分析阶段**，首先从实际问题出发，用面向对象的方法分析用户需求，建立一个体现系统重要性的分析模型。该阶段应精确地抽象出系统应该包含哪些功能。
- 2、**设计阶段**包括两方面的工作：①把面向对象的分析模型运用到面向对象的设计，作为面向对象的设计的一部分；②针对具体实现中的人-机界面、数据存储、任务管理等因素添加一些与实现有关的内容，建立系统的设计模型
- 3、**编程**是面向对象的软件开发最终实现的重要阶段，程序员用面向对象的程序设计语言进行编程，实现软件系统。
- 4、**测试**的任务是发现软件中的错误。任何一款软件产品在交付前都要经过测试。
- 5、**维护**为了保障软件的正常使用。

5.3 类和对象

C++提供对数据结构和方法的封装、抽象，称为类。类是封装的基本单元，也可以将类理解为一种新的数据类型，它包含数据单元和对数据的操作。类与结构体的使用类似，只不过结构体不包含对数据的操作。先定义一个类类型，然后再使用该类定义若干个对象，对象实际上就是一种类类型变量。

5.3 类和对象

5.3.1 类和定义

类是一种用户自定义数据类型，类的定义包含两部分内容：**数据成员**和**成员函数**（又称**函数成员**），数据成员表示该类所具有的属性，成员函数表示该类的行为，一般是对数据操作的函数，也称为方法。其定义格式如下：

```
class 类名
{
    public:
        公有的数据成员和成员函数
    protected:
        受保护的数据成员和成员函数
    private:
        私有的数据成员和成员函数
};
```

5.3 类和对象

5.3.1 类和定义

说明：

- 定义类时使用关键字class，类名必须符合标识符命名规范，**一般类名的首字母大写**
- 一个类包含类头和类体两部分，class <类名>称为类头
- public（公有）、protected（受保护）与private（私有）为属性/方法的访问权限，用来控制类成员的存取。如果没有标识访问权限，默认为private
- 三种访问控制权限在类定义中可按任意顺序出现多次，但一个成员只有一种访问权限
- 结束部分的分号不能省略

5.3 类和对象

5.3.1 类和定义

根据上述类定义的规则，定义一个日期类Date。下面分析如何定义日期类。（1）日期都会具有三个属性：年、月、日，因此可以用整型数据表示这三个属性；（2）分析日期具有的行为，首先应该可以设置日期，比如2018年8月8日，其次可以显示该日期。因此，定义两个成员函数setDate()，show()来完成上述行为。Date类定义如下：

```
class Date
{
private: //以下是私有成员
    int year, month, day; //用三个整型表示属性：年、月、日
public: //以下是公有成员
    void setDate(int y, int m, int d) //函数成员：设置日期
    {
        year = y;
        month = m;
        day = d;
    }
    void show() //函数成员：显示日期
    {
        cout << year << "年" << month << "月" << day << "日" << endl;
    }
}; //类定义结束符
```

5.3 类和对象

5.3.1 类和定义

说明：

- 类体包含数据成员和函数成员的声明与实现， `setDate(int y, int m, int d)` 用三个参数初始化数据成员
- `show`函数显示三个数据成员
- 在类定义时，结尾处加上分号
- 类是一种抽象的数据类型，所以类定义不会分配内存空间，只有在使用类定义该类型的对象时，系统才会为对象分配具体的内存空间

5.3 类和对象

5.3.2 数据成员的实现

类定义中数据成员描述了类对象所具有的属性，数据成员的类型可以是**基本数据类型**，也可以是**用户自定义数据类型**，如数组、指针、引用、结构体等。当然也可以使用已经定义的类型表示数据成员，如：

```
enum color{red, white, blue}
class Car
{
private:
    int num;        \\基本数据类型
    color c;        \\枚举类型
    char *name;     \\指针
}
```



```
class Tool
{
    Car c;           //使用已经定义的类型
    Tool *t;         //正在定义类类型的指针
    Tool &s;          //正在定义类类型的引用
    Tool d;          //错误，使用未定义完整的类类型定义成员
}
```

5.3 类和对象

5.3.3 成员函数的实现

- 所有成员的声明都必须在类的内部，但是成员函数体的定义则既可以在类的内部也可以在类的外部。
- 当定义在类的外部时，函数名之前需要加上类名和作用域运算符 (::) 以显式的指出该函数所属的类。
- 在类外定义函数体的格式如下：

```
返回值类型 类名 :: 成员函数名(形参表)
{
    函数体;
}
```

- 在类的外部定义成员函数时，必须同时提供类名和函数名，函数参数列表要和类内函数声明一致。其中， :: 表示类的作用域分辨符，放在函数名之前类名之后，返回类型在类的作用域之外。

5.3 类和对象

5.3.3 成员函数的实现

Date类中的成员函数在类中声明：

```
class Date
{
private:
    int year, month, day;
public:
    void setDate(int y, int m, int d); //声明成员函数

    void show();                      //声明成员函数
};
```

在类外实现成员函数：

```
void Date::setDate(int y, int m, int d) //函数成员： 设置日期
{
    year = y;
    month = m;
    day = d;
}
void Date::show()
{
    cout << year << "年" << month << "月" << day << "日" << endl;
}
```

5.3 类和对象

5.3.3 成员函数的实现

【例 2】成员函数的实现

```
#include <iostream>
using namespace std;
class Date
{
private:
int year, month, day;
public:
void setDate(int y, int m, int d); //声明成员函数
void show(); //声明成员函数
};
void Date::setDate(int y, int m, int d) //函数成员： 设置日期
{
year = y;
month = m;
day = d;
}
void Date::show()
{
cout << year << "年" << month << "月" << day << "日" << endl;
}
int main()
{
Date da;
da.setDate(12, 4, 5);
da.show();
return 0;}
```

5.3 类和对象

5.3.4 对象的定义和使用

(1) 先定义类，再定义对象

```
class 类名  
{ 数据成员和成员函数 };  
类名 对象列表;
```



如使用已经定义的Date类来定义日期对象：
Date d1,d2,d3;

(2) 在声明类的同时定义类对象

```
class 类名  
{ 数据成员和成员函数 }对象列表;
```



```
class Date  
{  
.....//数据成员和函数成员的实现  
}d1,d2,d3; //声明类型的同时定义三个对象
```

5.3 类和对象

5.3.4 对象的定义和使用

在创建对象之后，就可以使用对象访问对象中的数据成员和成员函数，一般有三种方式可以访问对象中的成员：

- 通过对象名和成员运算符访问对象成员。
- 通过指向对象的指针和指针运算符访问对象中的成员。
- 通过对象的引用变量访问对象中的成员。

通过对象名和成员运算符访问对象成员的格式如下：

- 对象名. 数据成员名
- 对象名. 成员函数名

其中“.”是成员运算符，用来限定访问哪一个对象中的成员，例如：

```
Date d;
```

```
d.show(); //访问对象中的公有成员
```

```
d.year; //错误，类外不能访问私有成员
```

5.3 类和对象

5.3.5 成员的访问权限

C++通过 public、protected、private 三个关键字来控制数据成员和成员函数的访问权限，它们分别表示公有类型(public)、私有类型(private)和保护类型(protected)。三者的意义如下：

- 被public限定符所修饰的数据成员和函数可以被该类的函数、子类、友元函数访问，也可以由该类的任意对象访问，即可以使用成员运算符来访问，适用于完全公开的数据。这里的友元函数，可以是该类的友元函数，也可以是该类的友元类的成员函数。
- protected限定符修饰的数据成员和成员函数可以被该类的成员函数访问，但是不能被类对象所访问，即不能通过类对象的成员运算符来访问，属于半公开性质的数据。另外，这些成员可以被子类和友元函数访问。
- 被private限定符修饰的数据成员和成员函数只能被该类的函数和友元函数访问，子类无法访问。在这三个限定符中封装程度最高，一般来说，应该尽可能将类的成员声明为private，封装类的实现细节，提高类的安全性。

注意：C++ 中的 public、private、protected 只能修饰类的成员，不能修饰类，C++中的类没有公有私有之分。

5.3 类和对象

5.3.5 成员的访问权限

表1 访问权限表

访问权限	含义	可存取对象
public	公开	该类成员、子类、友元及所有对象
protected	受保护	该类成员及其子类、友元
private	私有	该类成员及友元

一般通过在类内定义的public成员函数对private成员进行存取，为private成员提供外界访问的接口。

5.4 构造函数与析构函数

5.4.1 构造函数的声明与使用

- ✓ 在创建对象时，经常需要自动的完成某些初始化工作，如数据成员初始化，C++提供了构造函数完成这项工作。(例如，学生对象，创建时姓名、年龄等属性值应被赋初值)
- ✓ C++的构造函数是一种特殊的成员函数，构造函数的功能是，在创建对象时，系统自动调用构造函数对数据成员初始化，不需要用户显式调用。构造函数定义格式如下：

```
class 类名
{
public:
    构造函数名 (参数列表)
    {
        函数体
    }
};
```


5.4 构造函数与析构函数

5.4.1 构造函数的声明与使用

构造函数是类的成员函数，除具有一般函数的特征外，还拥有以下特点：

- 构造函数是不需要用户显式调用，定义对象时被自动执行。
- 构造函数名与类名相同，无返回类型。注意什么也不写，也不能用void。
- 构造函数的访问属性应该是公有属性。
- 主要用于对数据成员进行初始化，一般不做初始化以外的工作。对象生存周期内，只调用一次构造函数。
- 构造函数可以访问类中所有成员，可以不带任何参数，可以带参数表和默认值用户，可以根据具体需要设计合适的构造函数。
- 构造函数还可以重载，类中可以有多多个构造函数，系统根据构造函数参数表进行区分，选择其中的一个执行。
- 如果用户没有定义构造函数，系统会自动定义一个无参的默认构造函数，它不对数据成员做任何操作，即函数体为空；如果用定义了构造函数，系统就不会为创建默认构造函数，而是根据对象的参数类型和参数个数从用户定义的构造函数中选择最合适的构造函数完成初始化。

5.4 构造函数与析构函数

5.4.1 构造函数的声明与使用

【例 3】定义Date类不带参数的构造函数

```
#include<iostream>
using namespace std;
class Date
{
private:
    int year, month, day;//用三个整型表示年月日
public:
    Date() //定义不带参数的构造函数
    {
        year = 2019; //初始化数据成员
        month = 5;
        day = 20;
        cout<< "调用构造函数" <<endl;
    }
//Date():year(2019), month(5), day(20) { cout <<
"调用构造函数" << endl; }
```

```
void setTime(int y, int m, int d)
{
    year = y;
    month = m;
    day = d;
}
void show()
{
    cout << year << "年" << month << "月" << day << "日" << endl;
}
};
//Date::Date() : year(2019), month(5), day(20) { cout <<
"调用构造函数" << endl; };
void main()
{
    Date d; //自动调用构造函数, 初始化数据成员
    d.show();
    d.setTime(2019, 9, 20);
    d.show();
}
```

5.4 构造函数与析构函数

5.4.1 构造函数的声明与使用

- ❑ 除了在函数体内对数据成员初始化，C++还提供另一种初始化方式：使用初始化列表来实现对数据成员的初始化，格式如下：

```
类名：：构造函数名（参数表）：初始化列表  
{  
    //构造函数其他代码  
}
```

- 初始化列表的形式如下：

数据成员1（参数名或常量），数据成员2（参数名或常量），数据成员3（参数名或常量）

- 例如，可将例3中的构造函数改成如下形式：

```
Date():year(2019), month(5), day(20) { }
```

5.4 构造函数与析构函数

5.4.2 重载构造函数

- 在一个类中可以有多多个构造函数，即可以重载构造函数，系统在调用构造函数时，根据参数类型和参数个数进行区分，选取合适的构造函数。

【例 4】重载Date类中的构造函数

```
#include<iostream>
using namespace std;
class Date
{
private:
    int year, month, day; //用三个整型表示年月日
public:
    Date(int y); //带一个参数的构造函数
    Date(int y,int m); //带两个参数的构造函数
    Date(int y,int m,int d); //带三个参数的构造函数
};
```

```
Date::Date(int y)
{
    year = y;
    month = 5;
    day = 20;
    cout << "调用构造函数1" << endl;
}
Date::Date(int y,int m)
{
    year = y;
    month = m;
    day = 10;
    cout << "调用构造函数2" << endl;
}
```

5.4 构造函数与析构函数

5.4.2 重载构造函数

【例 4】重载Date类中的构造函数（续）

```
Date::Date(int y, int m, int d)
{
    year = y;
    month = m;
    day = d;
    cout << "调用构造函数3" << endl;
}
void main()
{
    Date d1(2019);
    Date d2(2019, 2);
    Date d3(2019, 2, 12);
}
```

C:\Windows\system32\cmd.exe

```
调用构造函数1
调用构造函数2
调用构造函数3
请按任意键继续. . .
```

5.4 构造函数与析构函数

5.4.3 带默认参数值的构造函数

- 前面介绍的带参的构造函数，在创建对象时必须传递相应的实参，构造函数才被执行。但在实际应用中，对象经常会有一些默认初始值：职工的性别默认为“男”，点的坐标默认为（0,0）等。C++允许使用带默认参数值的构造函数，一般情况下，对象的数据成员为默认值，当然用户也可以根据需求自行设置。
- 【例 5】设计学生类Student，带默认参数值的构造函数**

```
1. #include<iostream>
2. using namespace std;
3. class Student
4. {
5. private:
6.     char name[20]; //姓名
7.     char sex[10];  //性别
8.     int age;       //年龄
9. public:
10.    Student()      //无参的构造函数
11.    {
12.        //不作任何初始化工作
13.        cout << "调用无参构造函数" << endl;
14.    }
```


5.4 构造函数与析构函数

5.4.3 带默认参数值的构造函数

【例 5】设计学生类Student，带默认参数值的构造函数（续13ban）

```
15. Student(char n[], char s[10] = "女", int a = 18) //带默认参数的构造函数
    数
16. {
17.     strcpy_s(name, strlen(n)+1, n);
18.     //使用strcpy_s函数实现字符串的复制
19.     strcpy_s(sex, strlen(s) + 1, s);
20.     age = a;
21.     cout << "调用带默认参数值的构造函数" << endl;
22. }
23.};
24. void main()
25. {
26.     Student s1;
27.     Student s2("张华");
28.     Student s3("李明", "男", 12);
29. }
```

C:\Windows\system32\cmd.exe

调用无参构造函数
调用带默认参数值的构造函数
调用带默认参数值的构造函数
请按任意键继续. . .

5.4 构造函数与析构函数

5.4.3 带默认参数值的构造函数

- ◆ 程序中创建对象s1调用无参构造函数；创建对象s2时，使用带默认参数的构造函数的两个默认值“女”、“18”来构造对象，第一个参数是“张华”。第28行语句仍调用带默认参数的构造函数，未采用默认值。注意，对象参数类型和构造函数参数要逐一匹配。
- ◆ 如果将程序中的带默认参数的构造函数改为如下格式：

```
Student(char n[]="小华", char s[10] = "女", int a = 18)
{
    strcpy_s(name, strlen(n)+1, n);
    strcpy_s(sex, strlen(s) + 1, s);
    age = a;
    cout << "调用带默认参数值的构造函数" << endl;
}
```

- 则程序执行到26行，出现对重载函数调用不明确的问题。初始化对象s1时，既可以调用无参构造函数，也可以调用带默认值的构造函数（形参到默认值，可以不用传值），因此出现二义性问题。因此，在定义带默认参数的构造函数时，要避免此类问题的出现。

5.4 构造函数与析构函数

5.4.4 析构函数

- 析构函数也是类的一种特殊成员函数，它的作用与构造函数相反，一般是执行对象的清理工作。当对象的生命周期结束时，通常需要做一些善后工作，例如：构造对象时，通过构造函数动态申请了一些内存单元，在对象消失之前就要释放这些内存单元。
- C++机制提供析构函数来完成这项工作，系统会自动的调用析构函数释放为对象分配的资源。析构函数的定义格式如下：

```
类名：~析构函数名()  
{  
    函数体  
}
```

5.4 构造函数与析构函数

5.4.4 析构函数

- ❑ 析构函数名前加上一个逻辑非运算符，表示与构造函数相反，相当于“逆构造函数”。析构函数具有如下特点：
 - 析构函数名与类名相同，但在类名前加“~”字符。
 - 析构函数没有返回值，没有参数，不能被重载，一个类仅有一个析构函数。
 - 析构函数一般由用户自定义，对象消失时系统自动调用。如果用户没有定义析构函数，系统将自动生成一个不作任何工作的析构函数。
- ❑ 以下情况，析构函数将会自动调用：
 - 如果一个对象被定义在一个函数体内，则当该函数结束时，该对象的析构函数被自动调用。
 - 若一个对象是使用new运算符动态创建的（new调用构造函数），在使用delete运算符释放对象时，delete将会自动调用析构函数。

注意：对象消失时的清理工作并不是由析构函数完成，而是由用户在析构函数中添加的清理语句完成。

5.4 构造函数与析构函数

5.4.4 析构函数

【例 6】带析构函数的Student类

```
#include<iostream>
using namespace std;
class Student
{
private:
    char name[20];//姓名
    char sex[10];//性别
    int age;//年龄
public:
    Student(char n[], char s[10], int a )
    {
        strcpy_s(name,strlen(n)+1, n);
        strcpy_s(sex, strlen(s) + 1, s);
        age = a;
        cout << "调用构造函数:"<<name << endl;
    }
}
```

5.4 构造函数与析构函数

5.4.4 析构函数 【例 6】带析构函数的Student类（续）

```
~Student() //定义析构函数
{
    cout << "调用析构函数: " << name << endl;
}
};
void fun() //普通函数
{
    Student s1("小王", "男", 20);
}
void main()
{
    fun();
    Student s2("李明", "男", 12);
    Student s3("小张", "男", 15);
}
```

➤ 构造函数在对象创建时自动调用，调用顺序与对象定义顺序相同。当主函数执行结束，对象生命周期结束，调用析构函数，析构函数与构造函数顺序相反。对于同一存储类别的对象，先构造的对象后析构，后构造的对象先析构。

C:\Windows\system32\cmd.exe

```
调用构造函数:小王
调用析构函数: 小王
调用构造函数:李明
调用构造函数:小张
调用析构函数: 小张
调用析构函数: 李明
请按任意键继续. . .
```

5.5 拷贝构造函数

□ 拷贝构造函数是一种特殊的构造函数

- 当程序中需要用一种已经定义的对象去创建另一个对象时，或将一个对象赋值给另一个对象，就需要用到拷贝构造函数。
- 拷贝构造函数的名称必须和类名称一致，没有返回类型，只有一个参数，该参数是该类型对象的引用。
- 拷贝构造函数的定义格式如下：

```
拷贝构造函数名（类名& 对象名）  
{  
    函数体 ...  
}
```

- 拷贝构造函数一般由用户自定义，如果用户没有定义拷贝构造函数，系统将自动生成一个默认的拷贝构造函数进行对象之间的拷贝。
- 如果用户自定义了拷贝构造函数，则在用一个类的对象初始化该类的另外一个对象时，自动调用自定义的拷贝构造函数。
- 拷贝函数的功能就是把有初始值对象的每个数据成员依次赋值给新创建的对象。

5.5 拷贝构造函数

□ 以下三种情况会调用拷贝构造函数：

- 当函数的形参为类的对象，将对象作为函数实参传递给函数的形参时。
- 当函数的返回值是类的对象，创建临时对象时。
- 当用类的一个对象初始化另外一个对象时。

5.5 拷贝构造函数

【例 7】带拷贝构造函数的Student类

```
#include<iostream>
using namespace std;
class Student
{
private:
    char name[20]; //姓名
    char sex[10];  //性别
    int age;       //年龄
public:
    Student(char n[], char s[10], int a ) //带参数的构造函数
    {
        strcpy_s(name, strlen(n)+1, n);
        strcpy_s(sex, strlen(s) + 1, s);
        age = a;
        cout << "调用构造函数:" << name << endl;
    }
}
```

5.5 拷贝构造函数

【例 7】带拷贝构造函数的 Student 类（续）

C:\Windows\system32\cmd.exe

```
调用构造函数: Jane
调用拷贝构造函数: Jane
调用构造函数: July
调用拷贝构造函数: July
调用拷贝构造函数: July
调用析构函数: July
调用析构函数: July
调用析构函数: July
调用析构函数: Jane
调用析构函数: Jane
请按任意键继续. . .
```

```
Student(Student & s)//定义拷贝构造函数 14ban
{
    strcpy_s(name, strlen(s.name) + 1, s.name);
    strcpy_s(sex, strlen(s.sex) + 1, s.sex);
    age = s.age;
    cout << "调用拷贝构造函数:" << name << endl;
}

~Student()//定义析构函数
{
    cout << "调用析构函数: " << name << endl;
};

Student fun( Student a) //返回值为类类型的普通函数
{
    return a;
}

void main()
{
    Student stu1("Jane","女",20);
    Student stu2 = stu1;
    Student stu3("July", "女", 15);
    fun(stu3);
}
```

5.5 拷贝构造函数

- 上例中，用户自定义了一个拷贝构造函数，**当用一个对象初始化另一个对象时**，调用拷贝构造函数。当**对象作为函数的返回值时**需要调用拷贝构造函数，此时C++将从堆中动态建立一个临时对象，将函数返回的对象拷贝给该临时对象，并把该临时对象的地址存储在寄存器里，从而由该临时对象完成函数返回值的传递。
- 如果用户未定义拷贝构造函数，系统也会完成工作。C++把这种对象之间数据成员的简赋值称为“**浅拷贝**”，默认拷贝构造函数执行的也是浅拷贝。大多情况下“浅拷贝”已经能很好地工作，但是一旦对象存在了动态成员，那么浅拷贝就会出问题，通过如下一段代码加以说明。

5.5 拷贝构造函数

【例 8】设计一个图书类 **Book**，两个数据成员 **name**，**price** 分别表示图书的名称和价格。**show()** 函数显示图书信息

```
#include<iostream>
#include<assert.h>
using namespace std;
class Book
{
private:
    char *name;        //图书名称
    double price;      //图书价格
public:
    Book(char *n,double p)
    {
        int length = strlen(n);
        name = new char[length + 1];
        strcpy_s(name,length+1 ,n );
        price = p;
        cout << "调用构造函数: " << name << endl;
    }
}
```

5.5 拷贝构造函数

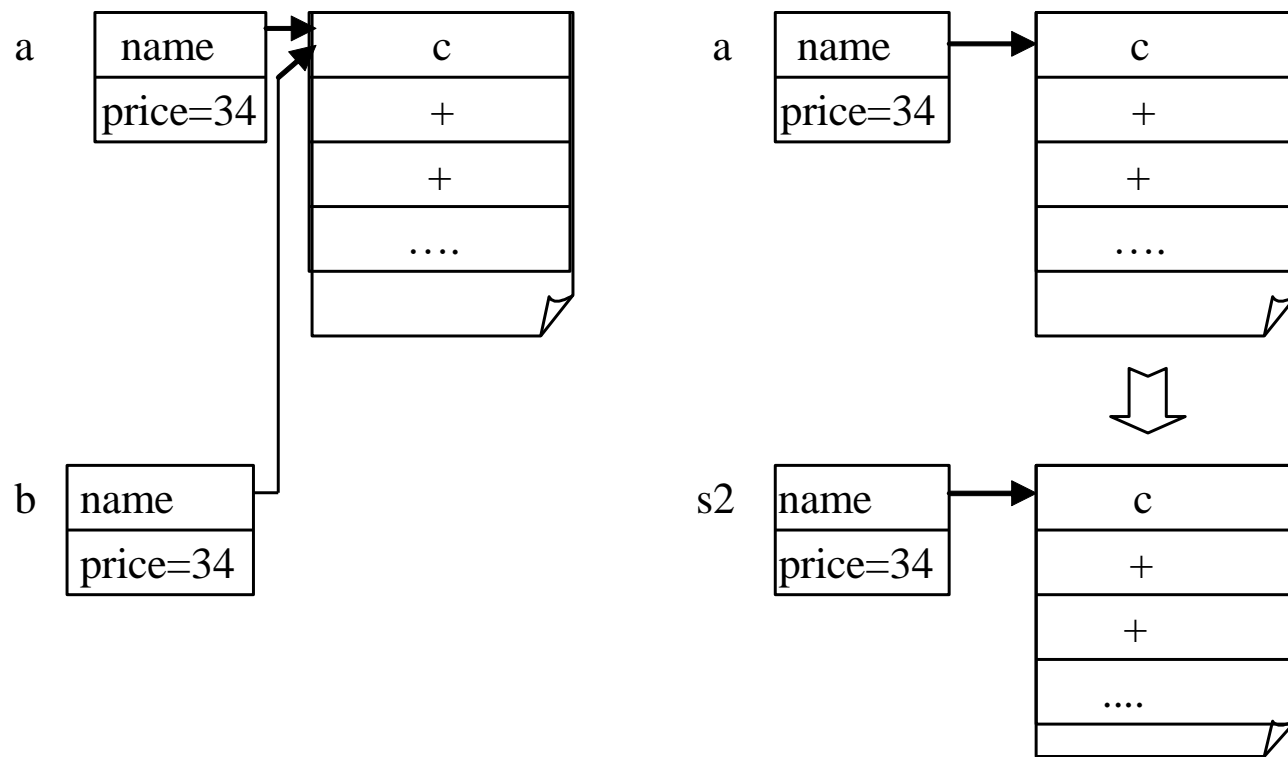
【例 8】设计一个图书类 **Book**，两个数据成员 **name**，**price** 分别表示图书的名称和价格。**show()** 函数显示 图书信息(续)

```
~Book()
{
    if(name!=NULL)
    {
        cout << "调用析构函数: " << name << endl;
        delete[] name;//释放分配的内存资源
        name= NULL;
    }
};
int main()
{
    Book a("c++程序设计",34);
    Book b(a);
    return 0;
}
```

! 在这段代码运行结束之前，会出现一个运行错误。原因在于默认拷贝构造函数在进行对象复制时，执行的是浅拷贝，只是将数据成员的值进行赋值，即执行下列操作：
`b.name = a.name;` `b.price = a.price;`

5.5 拷贝构造函数

这两个指针指向了堆里的同一个内存空间，如下图2 (a)所示。对象b析构，释放内存，然后对象a析构，由于a.name 和b.name所占用的是同一块内存，而同一块内存不可能释放两次，所以当对象a析构时，程序出现异常，无法正常执行和结束。



(a) 执行 `Book b(a);` 进行浅拷贝

(b) 执行 `Book b(a);` 进行深拷贝

图2 浅拷贝与深拷贝说明

5.5 拷贝构造函数

- 当类中的数据成员是**指针类型**时，必须定义一个特殊的拷贝构造函数，该拷贝构造函数不仅可以实现对象之间数据成员的赋值，而且可以为新对象单独分配内存空间，这就是“深拷贝”。

在例8中加入带深拷贝的构造函数：

```
Book(Book &b)
{
    int length = strlen(b.name);
    name = new char[length+1];
    if(name!=NULL)
        strcpy_s(name, length+ 1, b.name);
    price = b.price;
    cout << "调用拷贝构造函数： " << name << endl;
}
```

C:\Windows\system32\cmd.exe

```
调用构造函数： c++程序设计
调用拷贝构造函数： c++程序设计
调用析构函数： c++程序设计
调用析构函数： c++程序设计
请按任意键继续. . .
```

5.6 对象的使用

- ❑ 对象与普通变量类似，除了可以定义简单对象之外，还可以定义对象指针、对象数组、对象引用、动态对象等。
- ❑ 对象一旦声明就为其成员分配存储空间，并调用其构造函数进行初始化，并在生存期结束后自动调用析构函数以释放对象占用的内存空间。
- ❑ 程序员无法控制何时调用对象的构造函数，也无法决定何时释放对象占用的存储空间。如果在程序中需要使用许多对象，并且这些对象均占用大量存储空间，就会造成内存紧张。
- ❑ 一种比较好的解决途径是先声明这些对象，但并不立即分配存储空间和调用构造函数，在需要这些对象时才调用构造函数。对象使用完立即调用析构函数，并释放占用的内存空间，而不是等到对象生存期结束后才由系统自动回收存储空间。**解决方法就是使用指向对象的指针。**

5.6 对象的使用

5.6.1 对象指针

- 对象如同一般变量，占用一块连续的内存区域，因此可使用一个指向对象的指针来访问对象，即对象指针，它指向存放该对象的地址。
- 对象指针的定义与一般变量指针类似，格式如下：

类名 *对象指针名;

- 对象指针通过“->”运算符访问公有数据成员和成员函数。访问对象成员的格式如下：

对象指针名->数据成员名
对象指针名->成员函数名(参数表)

注意：对象指针在使用之前要进行初始化，让它指向一个已经声明过的对象，然后再使用，并且定义对象指针不调用构造函数。

5.6 对象的使用

5.6.1 对象指针

例如：

```
Student s1("jane",1002);
```

```
Student *s2=&s1;//定义对象指针并初始化
```

□ 使用对象指针作函数形参有如下两点好处：

- 实现地址传递。通过在调用函数时，将实参的地址传递给形参，使得实参和形参指向同一块内存空间。则在被调用函数中改变形参的值，实参对象的值也会随着改变，实现函数之间的信息传递。
- 程序运行效率高。使用对象指针形参仅将对象的地址值传给形参，而不进行副本的拷贝，这样可以提高运行效率，减少时空开销。

5.6 对象的使用

5.6.1 对象指针

【例 9】对象指针作为函数参数的使用

```
#include<iostream>
using namespace std;
class A
{
public:
    A()
    {
        x = y = 0;
        cout << "调用构造函数1" << endl;
    }
    A(int i, int j)
    {
        x = i; y = j;
        cout << "调用构造函数2" << endl;
    }
    ~A()
    {
        cout << "调用析构函数" << endl;    }
```


5.6 对象的使用

5.6.1 对象指针 【例 9】对象指针作为函数参数的使用（续）

```
void setValue(int i, int j) { x = i; y = j; }  
void print() { cout << x << ", " << y << endl; }  
  
private:  
    int x, y;  
};  
void fun(A m1, A *m2)    //对象指针作为形参  
{  
    m1.setValue(10, 5);  
    m2->setValue(23, 12);  
}  
void main()  
{  
    A p(5, 7), *q, d(2, 4);    //定义对象指针q  
    q = &p;                    //对象指针初始化, 指向对象p  
    q->print();                 //通过对象指针访问成员函数  
    fun(p, &d);                //调用函数  
    p.print();  
    d.print();  
}
```

C:\Windows\system32\cmd.exe

```
调用构造函数2  
调用构造函数2  
5, 7  
调用析构函数  
5, 7  
23, 12  
调用析构函数  
调用析构函数  
请按任意键继续. . .
```

5.6 对象的使用

5.6.2 对象引用

- ❑ 对象引用与普通变量引用的定义相同，引用对象就是对已经存在的对象起“别名”。其实质就是通过将被引用对象的地址赋给引用对象，使二者指向同一内存空间。
- ❑ 定义一个对象引用，并同时指向一个对象的格式为：

类名 & 对象引用名=被引用对象；

注意：

- 引用对象与被引用对象必须是相同类型；
- 对象引用在定义时必须初始化（除了作为函数参数与函数返回值），并且被引用对象必须已经定义；
- 定义对象引用并不会分配内存空间，也不会调用构造函数。

5.6 对象的使用

5.6.2 对象引用

使用对象引用访问对象成员的格式如下：

对象引用名.数据成员名
对象引用名.成员函数名(参数表)

【例 10】对象引用作为函数参数

```
#include<iostream>
using namespace std;
class A
{
public:
    A()
    {
        x = y = 0;
        cout << "调用构造函数1" << endl;
    }
}
```

5.6 对象的使用

5.6.2 对象引用 **【例 10】** 对象引用作为函数参数 (续)

```
A(int i, int j)
{
    x = i; y = j;
    cout << "调用构造函数2" << endl;
}
~A()
{
    cout << "调用析构函数" << endl;
}
void setValue(int i, int j) { x = i; y = j; }
void print()
{ cout << x << ", " << y << endl; }
private:
    int x, y;
};
```

```
void fun(A m1, A &m2)    //对象引用作为形参
{
    m1.setValue(10, 5);
    m2.setValue(23, 12);
}
void main()
{
    A p(5, 7), *q, d(2, 4);    //定义对象指针q
    q = &p;                    //对象指针初始化, 指向对象p
    q->print();                 //通过对象指针访问成员函数
    fun(p, d);                 //调用函数
    p.print();
    d.print();
}
```

5.6 对象的使用

5.6.3 对象数组

- ❑ 对象数组中每一个数组元素都是对象，不仅具有的数据成员，而且还有函数成员。
- ❑ 定义一个一维对象数组的格式如下：

类名 对象数组名[常量表达式];

- 其中，类名指出该数组元素所属的类型，常量表达式给出一维数组元素的个数。
- 与结构数组不同，对象数组初始化需要调用构造函数完成，以一个大小为n的一维数组为例，对象数组的初始化格式如下：

```
类名 数组名[n]={ 类名(数据成员1初值, 数据成员2初值,...),  
                  类名(数据成员1初值, 数据成员2初值,...),  
                  ...  
                  类名(数据成员1初值, 数据成员2初值,...)};
```

5.6 对象的使用

5.6.3 对象数组

- 如果在定义对象数组时未给出初始化表，将调用不带参的构造函数完成对象的初始化工作。
- 引用对象数组元素中的公有成员，其一般格式如下：

数组名[下标表达式].数据成员名
数组名[下标表达式].成员函数名（参数表）

5.6 对象的使用

5.6.3 对象数组

【例11】 定义一个汽车类**Automobile**，记录多辆汽车的基本信息，信息包括车的型号（字符串表示）、颜色（枚举类型表示）、售价（整数表示）、里程（实数表示）等

```
#include<iostream>
using namespace std;
enum Color { Red, White, Black, Blue, Yellow, Green }; //定义枚举类型Color
class Automobile //定义汽车Automobile类
{
private:
    char name[20]; //表示车的型号
    Color c; //车的颜色
    int sale; //车的售价,单位: 万
    double distance; //里程, 单位: 公里
public:
    Automobile(char name[20], Color c = Red, int sale = 0, double distance= 0); //带默认值的构造函数
    ~Automobile();
    void Show();//显示汽车信息
};
```

5.6 对象的使用

5.6.3 对象数组

【例11】 定义一个汽车类**Automobile**，记录多辆汽车的基本信息，信息包括车的型号（字符串表示）、颜色（枚举类型表示）、售价（整数表示）、里程（实数表示）等（续）

```
Automobile::Automobile(char n[20], Color c, int sale, double distance)
{
    strcpy_s(this->name, strlen(n) + 1, n);
    this->c = c;
    this->sale = sale;
    this->distance = distance;
    cout << "构造函数:" << name << endl;
}
Automobile::~~Automobile()
{
    cout << "析构函数:" << name << endl;
}
```

5.6 对象的使用

5.6.3 对象数组

【例11】 定义一个汽车类**Automobile**，记录多辆汽车的基本信息，信息包括车的型号（字符串表示）、颜色（枚举类型表示）、售价（整数表示）、里程（实数表示）等（续）

```
void Automobile::Show()
{
    cout << "汽车型号: " << name << endl;
    cout << "汽车颜色:" << c << endl;
    cout << "汽车售价: " << sale << endl;
    cout << "汽车里程: " << distance << endl;
}
int main()
{
    Automobile c1[3] = { Automobile("SL600",Red,30,1300),
                        Automobile("C500",Black,50,800),
                        Automobile("M600",Blue,45,1500) }; //定义对象数组
    c1[0].Show();//对象数组元素访问成员函数
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
构造函数:SL600
构造函数:C500
构造函数:M600
汽车型号: SL600
汽车颜色:0
汽车售价: 30
汽车里程: 1300
析构函数:M600
析构函数:C500
析构函数:SL600
请按任意键继续. . .
```

5.6 对象的使用

5.6.4 动态对象

□ C++中，可以用new运算符动态建立对象，用delete运算符删除对象。new运算符创建对象时会自动调用构造函数，而delete运算符删除对象时会自动调用析构函数。使用new定义动态对象的格式如下：

```
类名 * 指针名;  
对象指针=new 类名(参数表);
```

5.6 对象的使用

5.6.4 动态对象

说明：

- 动态对象存储在new语句从堆申请的空间中。
- 建立动态对象时要调用构造函数，当初值表缺省时调用默认的构造函数。
- 因为new运算符返回的是内存地址，所以必须赋值给同类型的指针，用户可以根据new的返回值是否为NULL，判断分配空间是否成功。
- 创建动态对象时，参数的个数与该类定义的构造函数参数相匹配。
- 将动态对象赋值给对象指针，通过该指针可以引用动态对象。
- 通过new运算符创建的动态对象不会自动消失，需要通过delete语句删除，删除对象时，会释放对象所占用的内存空间。语法格式为：

```
delete 对象指针;
```

5.6 对象的使用

5.6.4 动态对象

通过new运算符还可以创建动态对象数组，建立一维动态对象数组的格式为：

```
对象指针=new 类名[下标表达式];
```

删除一个动态对象数组的格式为：

```
delete [] 对象指针;
```

注意：在建立动态对象数组时，要调用构造函数，调用的次数与数组的大小相同；删除对象数组时，要调用析构函数，调用次数与数组的大小相同。

5.6 对象的使用

5.6.4 动态对象

【例12】 动态数组实现例11

```
#include<iostream>
using namespace std;
enum Color { Red, White, Black, Blue, Yellow, Green }; //定义枚举类型Color
class Automobile //定义汽车Automobile类
{
private:
    char name[20]; //表示车的型号
    Color c; //车的颜色
    int sale; //车的售价,单位: 万
    double distance; //里程, 单位: 公里
public:
    Automobile(char name[20] = "", Color = Red, int = 0, double = 0); //带默认值的构造函数
    ~Automobile();
    void Assign(char n[20], Color c, int sale, double distance); //为对象赋值
    void Show(); //显示汽车信息
};
```

5.6 对象的使用

5.6.4 动态对象

【例12】 动态数组实现例11（续）

```
Automobile::Automobile(char n[20], Color c, int sale, double distance)
{
    strcpy_s(this->name, strlen(n) + 1, n);
    this->c = c;
    this->sale = sale;
    this->distance = distance;
    cout << "构造函数:" << name << endl;
}
Automobile::~~Automobile()
{
    cout << "析构函数:" << name << endl;
}
void Automobile::Assign(char n[20], Color c, int sale, double distance)
{
    strcpy_s(this->name, strlen(n) + 1, n);
    this->c = c;
    this->sale = sale;
    this->distance = distance;
}
void Automobile::Show()
{
    cout << "汽车型号: " << name << endl;
    cout << "汽车颜色:" << c << endl;
    cout << "汽车售价: " << sale << endl;
    cout << "汽车里程: " << distance << endl;
}
```

5.6 对象的使用

5.6.4 动态对象

【例12】 动态数组实现例11（续）

```
int main()
{
    Automobile *c2 = new Automobile("BM600", Red, 50, 100); //建立动态对象
    Automobile *c3 = new Automobile[2]; //定义动态对象数组
    c3[0].Assign("SL600", Red, 30, 1300); //调用成员函数Assign为动态对象数组元素赋值
    c3[1].Assign("C500", Black, 50, 800);
    c3[1].Show();
    delete c2; //删除动态对象
    delete [] c3; //删除动态对象数组
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
构造函数:BM600
构造函数:
构造函数:
汽车型号: C500
汽车颜色:2
汽车售价: 50
汽车里程: 800
析构函数:BM600
析构函数:C500
析构函数:SL600
请按任意键继续. . .
```

5.6 对象的使用

5.6.5 成员对象

- 类内嵌其他类对象的类称为组合类，内嵌的对象称为成员对象。在声明该成员对象之前，成员类应该已经定义。

例：

```
class Time{ ...}; //创建一个时间类Date，类体略
class Date
{ Time t1 , t2;}; //创建一个日期类Date，该类包含两个数据成员d1和d2，这两个成员是Time类对象
```

5.6 对象的使用

5.6.5 成员对象

- 在创建对象时，成员对象作为对象的一部分也会自动创建。在C++中通过使用构造函数初始化列表来为组合类的成员对象初始化，其定义格式如下：

```
类名::构造函数（参数表）:成员对象(子参数表1)，成员对象(子参数表2).....  
{  
    构造函数体  
}
```

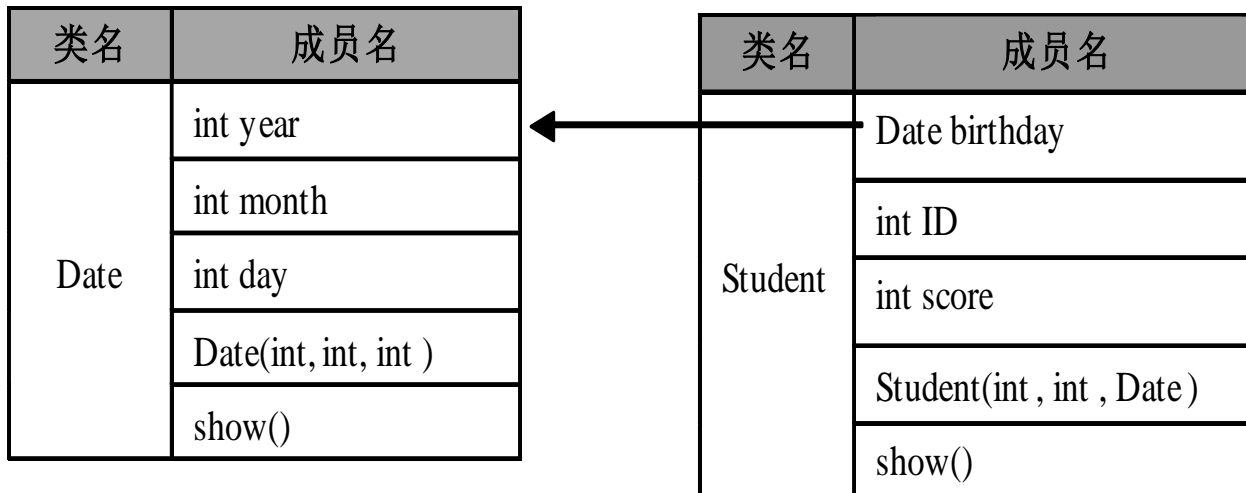
注意：

- 类中有成员对象时，该类的构造函数要包含对成员对象的初始化。如果构造函数的成员初始化列表没有对成员对象初始化，则使用成员对象的缺省构造函数。
- 建立一个类的对象时，应先调用其构造函数。但是如果这个类有成员对象，则要先执行成员对象自己所属类的构造函数，成员对象的初值从初始化列表获得。当全部成员对象都执行了自身类的构造函数后，再执行当前类的构造函数。

5.6 对象的使用

5.6.5 成员对象

【例13】设计一个学生类，学生信息包括学号，成绩，生日



5.6 对象的使用

5.6.5 成员对象

【例13】设计一个学生类，学生信息包括学号，成绩，生日（续）

```
#include<iostream>
#include<string>
using namespace std;
class Date //定义日期类
{
private:
    int year;
    int month;
    int day;
public:
    Date(int y, int m, int d)
    {
        year = y;
        month = m;
        day = d;
        cout << "调用日期类构造函数"<<endl;
    }
}
```

5.6 对象的使用

5.6.5 成员对象

【例13】设计一个学生类，学生信息包括学号，成绩，生日（续）

```
~Date()
{
    cout << "调用日期类析构函数" << endl;
}
void show()
{
    cout << "生日:";
    cout << year << "-" << month << "-" << day << endl;;
}
};
class Student //定义学生类
{
private:
    int ID; //学号
    int score; //成绩
    Date birthday; //生日（成员对象）
public:
```

5.6 对象的使用

5.6.5 成员对象

【例13】设计一个学生类，学生信息包括学号，成绩，生日（续）

```
Student(int ID, int score, Date bir) : birthday(bir) //成员对象初始化
{
    this->ID = ID;
    this->score = score;
    cout << "调用学生类构造函数" << endl;
}
~Student()
{
    cout << "调用学生类析构函数" << endl;
}
void show()
{
    cout << "The basic information:" << endl;
    cout << "学号:" << ID << "\t" << "成绩: " << score << "\n";
    birthday.show(); //通过函数调用显示生日日期
}
};
```

5.6 对象的使用

5.6.5 成员对象

【例13】设计一个学生类，学生信息包括学号，成绩，生日（续）

```
int main()
{
    Student stu(001, 67, Date(2000, 12, 23));
    stu.show();
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
调用日期类构造函数
调用学生类构造函数
调用日期类析构函数
调用日期类析构函数
The basic information:
学号:1 成绩: 67
生日:2000-12-23
调用学生类析构函数
调用日期类析构函数
请按任意键继续. . .
```

5.7 this指针

□ this指针的作用

- 指针存在于类的成员函数中，指向被调用函数类实例的地址。一个对象的this指针并不是对象本身的一部分，不会影响sizeof（）的结果。
- this指针的作用域是在类内部，当在类的非静态成员函数中访问类的非静态成员的时候，编译器会自动将对象本身的地址作为一个隐含参数传递给函数。

□ this指针的特点

- this指针只能在成员函数中使用；
- this 指针在成员函数开始前构造，在成员函数结束后清除，生命周期和其他函数参数一样。当调用一个类的成员函数时，编译器将类的指针作为函数的this参数传递进去。
- this指针会因编译器不同而有不同的放置位置。可能是栈，也可能是寄存器，甚至全局变量。

5.7 this指针

【例14】设计日期类Date，展示this指针的用法

```
#include<iostream>
using namespace std;
class Date
{
public:
    Date()
    {
        _year = 0;           //相当于this->_year=0
        _month = 0;
    }
    void Display()
    {
        cout << _year << "\t" ;
        cout << _month << endl;
    }
}
```


5.7 this指针

【例14】设计日期类Date，展示this指针的用法（续）

```
void SetDate(int year ,int month)
{
    _year = year;           //相当于this->_year=year
    _month = month;
}
private:
    int _year ,_month;
};
int main()
{
    Date Date1, Date2;
    Date1.SetDate(2019,8);
    Date2.SetDate(2019,2);
    Date1.Display();
    Date2.Display();
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
2019      8
2019      2
请按任意键继续. . .
```

5.7 this指针

在对象调用Date1.SetDate()时，成员函数除了接收2个参数外，还接收了一个对象Date1的地址。这个地址被隐含的形参this指针获取，它就等于执行了this=&Date1。所以对数据成员的访问都隐含的加上了前缀this->。若将SetDate()函数修改如下，将会产生疑义：

```
void SetDate(int _year,_month)
{
    _year = _year;
    _month=_month;
}
```

原因在于传入函数的形参与成员变量同名，编译器无法区分哪一个是成员变量，哪一个是参数，因此无法赋值。这时，必须加上this指针进行区别。

将程序修改如下：

```
void SetDate(int _year,_month)
{
    this->_year = _year;
    this->_month=_month;
}
```

5.7 this指针

注意：

构造函数比较特殊，没有隐含this形参。this指针是编译器自己处理的，编程者不能人为的在成员函数的形参中添加this指针的参数定义，也不能在调用时显示传递对象的地址给this指针。

5.8 友元

类具有封装和信息隐藏的特性，只有类的成员函数才能访问类的私有成员和受保护成员，类外的其他函数无法访问私有成员和受保护成员。

但有些情况下必须要访问某一些类的私有成员，此时就会为了这些特殊的少部分访问操作要把数据成员公有化，破坏私有成员的隐藏性。

为解决上述问题，C++提出一种使用友元（friend）的方案。

友元不是类的成员。友元是一种定义在类外部的普通函数或类，友元不是成员函数，但是它可以访问类中的保护和私有成员。

友元可以是另一个类的成员函数或者不属于任何一个类的普通函数，该函数被称为友元函数；友元也可以是一个类，该类被称为友元类。

5.8 友元

5.8.1 声明友元函数

- 友元函数可以是全局函数和其他类的成员函数，友元函数定义在类的外部，但它需要在类体内进行声明。为了与该类的成员函数加以区别，在声明时前面加以关键字friend修饰。
- 将全局函数声明为友元的写法如下：

```
friend 返回值类型 友元函数名(参数表);
```

- 将其他类的成员函数声明为友元的写法如下：

```
friend 返回值类型 其他类的类名::成员函数名(参数表);
```

注意：

- 友元函数是一个普通的函数，它不是本类的成员函数，因此在调用时不能通过对象调用。
- 友元函数可以在类内声明，类外定义。
- 友元函数对类成员的存取与成员函数一样，可以直接存取类的任何存取控制属性的成员；
- private、protected、public访问权限与友元函数的声明位置无关，因此原则上，友元函数声明可以放在类体中任意部分，但为程序清晰，一般放在类定义的后面。
- 不能把其他类的私有成员函数声明为友元。

5.8 友元

5.8.1 声明友元函数

【例15】定义Boat与Car两个类，二者都有weight属性，定义它们的一个友元函数totalWeight()，计算二者的重量和

```
#include<iostream>
using namespace std;
class Boat
{
public:
    Boat(double w)
    {
        weight=w;
    }
    friend double totalWeight(Boat a)
    {return a.weight;}
private:
    double weight;
};
```


5.8 友元

5.8.1 声明友元函数

【例15】定义Boat与Car两个类，二者都有weight属性，定义它们的一个友元函数totalWeight()，计算二者的重量和（续）

C:\Windows\system32\cmd.exe

totalweight:700

请按任意键继续. . .

```
class Car
{
public:
    Car(double w)
    {
        weight=w;
    }
    friend double totalWeight(Car b)
    { return b.weight; }

private:
    double weight;
};

void main()
{
    Boat aa(300);
    Car bb(400);
    cout<<"totalweight:"<<totalWeight(aa)+totalWeight(bb)<<endl;
}
```

5.8 友元

5.8.1 声明友元函数

一个类的成员函数可以是另一个类的友元。例如，教师可以修改学生成绩（访问学生的私有成员），则可以将教师的成员函数声明为学生类的友元函数。

【例16】 定义教师类和学生类，教师可以修改学生成绩

```
#include<iostream>
#include<string>
using namespace std;
class Student;//前向引用声明
class Teacher
{
public:
    void changeGrade(Student *s);//教师成员函数，修改学生成绩
};
```

5.8 友元

5.8.1 声明友元函数

【例16】定义教师类和学生类，教师可以修改学生成绩(续)

```
class Student {  
public:  
    Student(string name, int num, int grade[5]); //构造函数  
    void show();  
private:  
    string name;  
    int num;  
    int grade[5]; //五门功课成绩  
    friend void Teacher::changeGrade(Student *s); //将教师的成员函数声明为学生类的友元函数  
};  
Student::Student(string name, int num, int grade[])  
{  
    this->name = name,  
    this->num = num;  
    for (int i = 0; i < 5 ; i++)  
        this->grade[i] = grade[i];  
}
```

5.8 友元

5.8.1 声明友元函数

【例16】定义教师类和学生类，教师可以修改学生成绩（续）

```
void Student::show()
{
    cout << name << "的成绩为: ";
    for (int i = 0; i < 5; i++)
        cout << grade[i] << " ";
}
void Teacher::changeGrade(Student *s)
{
    int n, g; // 分别表示要修改的课程编号和成绩
    cout << "\n请输入想要修改的课程编号: (1-5) ";
    cin >> n;
    cout << "请重新输入成绩: ";
    cin >> g;
    s->grade[n-1] = g; // 友元函数访问私有成员
    cout << "修改成功! \n";
}
```

5.8 友元

5.8.1 声明友元函数

【例16】定义教师类和学生类，教师可以修改学生成绩（续）

```
int main()
{
    int grade[5];
    cout << "请输入五门功课的成绩: ";
    for (int i = 0; i < 5; i++)
        cin >> grade[i];
    Teacher T1;
    Student S1("July", 90, grade);
    S1.show();
    T1.changeGrade(&S1);
    S1.show();
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
请输入五门功课的成绩: 67 77 89 75 98
July的成绩为: 67 77 89 75 98
请输入想要修改的课程编号: (1-5) 2
请重新输入成绩: 100
修改成功!
July的成绩为: 67 100 89 75 98 请按任意键继续. . .
```

5.8 友元

5.8.2 声明友元类

- ❑ 友元除了前面讲过的函数以外，友元还可以是类。
- ❑ 如果把一个类声明为另一个类的友元，则称该类为友元类。
- ❑ 一个类 A 可以将另一个类 B 声明为友元，则类 B 的所有成员函数就都可以访问类 A 的私有成员，这就意味着 B 类的所有成员函数都是 A 类的友元函数。
- ❑ 友元类的声明需要在类名前加关键字 friend，在类定义中声明友元类的写法如下：

```
class A{  
    ....  
    friend class B;//声明B为A类的友元类  
};
```

注意：在声明一个友元类时，该类必须已经存在。

5.8 友元

5.8.2 声明友元类

【例17】 定义一个日期类，包括年、月、日、和时、分、秒

```
#include<iostream>
#include<string>
using namespace std;
class Date;
class Time //定义时间类
{
private:
    int hour;
    int minute;
    int second;
public:
    Time(int hour, int minute, int second)
    {
        this->hour = hour;
        this->minute = minute;
        this->second = second;
    }
}
```

5.8 友元

5.8.2 声明友元类

【例17】 定义一个日期类，包括年、月、日、和时、分、秒

```
friend class Date;//声明友元类
};
class Date //定义日期类
{
private:
    int year;
    int month;
    int day;
    Time t;
public:
    Date(int y, int m, int d,int h,int mi,int s):t(h,mi,s)//通过初始化列表对成员对象赋值
    {
        year = y;
        month = m;
        day = d;
    }
}
```

5.8 友元

5.8.2 声明友元类

【例17】 定义一个日期类，包括年、月、日、和时、分、秒

```
void showDate()//Time的友元函数
{
    cout << "当前日期为: ";
    cout << year << "-" << month << "-" << day << "\t";
    cout << t.hour << "时" << t.minute << "分" << t.second << "秒" <<
endl;//访问私有成员
}
};
int main()
{
    Date date1(2019,8,10,11,56,35);
    date1.showDate();
    return 0;
}
```

C:\Windows\system32\cmd.exe

当前日期为: 2019-8-10 11时56分35秒
请按任意键继续. . .

5.8 友元

5.8.2 声明友元类

友元关系在类之间不能传递，即类A是类B的友元，类B是类C的友元，并不能导出类A是类C的友元。“咱俩是朋友，所以你的朋友就是我的朋友”这句话在C++的友元关系上不成立。

使用友元类时注意：

- 友元关系不能被继承，B类是A类的友元，C类是B类的友元，C类与A类之间，如果没有声明，就没有任何友元关系，不能进行数据共享。
- 友元关系是单向的，不具有交换性。若类B是类A的友元，类A不一定是类B的友元。
- 友元关系不具有传递性。若类B是类A的友元，类C是B的友元，类C不一定是类A的友元。
- 友元提高了数据的共享性，增强了函数与函数之间，类与类之间的相互联系，极大提高了程序的运行效率。但友元的存在破坏了类中数据的隐蔽性和封装性，使得程序的可维护性降低。

5.9 静态成员

C++提供static声明的静态成员，用以解决同一个类的不同对象之间数据成员和函数的共享问题。类的静态成员有两种：**静态数据成员**和**静态函数成员**。

5.9.1 静态数据成员

□ 静态数据成员的特点

- 静态数据成员在内存中只保留一份拷贝，由该类的所有对象共同维护和使用，从而实现同一类中所有对象之间的数据共享。
- 静态数据成员的值可以被更新，一旦某一对象修改静态数据成员的值之后，其他对象再访问的是更新过之后的值。
- 静态数据成员属于类属性（class attribute），类属性是类的所有对象共同拥有的一个数据项，对于任何对象实例，它的属性值相同。

5.9 静态成员

5.9.1 静态数据成员

□ 静态数据成员的声明和初始化

- 静态成员的定义或声明要加关键词static，并且必须在类内声明，类外初始化。
- 类内声明静态数据成员的格式：

```
static 类型标识符 静态数据成员名;
```

- 在类外进行初始化格式为：

```
类型标识符 类名::静态数据成员名=初始值;
```


5.9 静态成员

5.9.1 静态数据成员

□ 静态数据成员的声明和初始化

说明:

- 静态数据成员是所有对象所公有，不属于任何一个对象，独立占用一份内存空间。
- 静态数据成员的访问属性同普通数据成员一样，可以为public、private和protected。
- 静态数据成员是一种特殊的数据成员，它表示类属性，而不是某个对象单独的属性。
- 静态数据成员使用之前必须初始化。
- 静态数据成员是静态存储的，它是静态生存周期。程序开始时就分配内存空间，而不是某个对象创建时分配；不随对象的撤销而释放，而是在程序结束时释放内存空间。
- 静态数据成员只是在类的定义中进行了引用性声明，因此必须在文件作用域的某个地方对静态数据成员用类名限定进行定义并初始化，即应在类体外对静态数据成员进行初始化。
- 静态数据成员初始化时前面不加static关键字，以免与一般静态变量或对象混淆。
- 静态数据成员初始化时必须使用类作用域运算符::来标明它所属的类。

5.9 静态成员

5.9.1 静态数据成员

□ 静态数据成员的访问

- 静态数据成员本质上是全局变量，在程序中，即使不创建对象其静态数据成员也存在。因此，可以通过类名对其直接访问，一般格式为：

```
类名::静态数据成员;
```

- 静态数据成员在类内可以随意访问，如果在类外，通过类名与对象只能访问控制属性为public的成员。

5.9 静态成员

5.9.2 静态成员函数

- 静态成员函数和静态数据成员一样，都属于类的成员，不属于某一个对象，是所有对象共享的成员函数，只要类存在，就可以使用静态成员函数。
- 静态成员函数是在函数声明前加关键字static，声明格式如下：

```
static 返回类型 静态成员函数名（形参表）；
```

- 静态成员函数可以在类内定义，也可以在类内声明，类外定义。在类外定义时不能在使用static作为关键字。
- 静态成员函数的调用形式有以下两种：
 - 通过类名直接调用静态成员函数：

```
类名::静态成员函数名（参数表）；
```

- 通过对象调用静态成员函数：

```
对象.静态成员函数（参数表）；
```

5.9 静态成员

5.9.2 静态成员函数

说明:

- 公有的静态成员函数可通过类名或对象名进行调用，一般非静态成员函数只能通过对象名调用。
- 通过对象访问静态成员函数之前，对象已经创建。
- 静态成员函数可以直接访问类中静态数据成员和静态成员函数，但不能直接访问类中的非静态成员。
- 由于静态成员是独立于对象而存在的，因此静态成员没有this指针。
- 类的非静态成员函数可以调用静态成员函数，反之不可以。

5.10 静态成员

5.9.2 静态成员函数

【例18】静态成员函数访问非静态数据成员

```
#include <iostream>
using namespace std;
class CRectangle
{
private:
    int w, h;           //矩形的长和高
    static int totalArea; //矩形总面积
    static int totalNumber; //矩形总数

public:
    CRectangle(int w_, int h_);
    ~CRectangle();
    CRectangle(CRectangle &r);
    static void PrintTotal(CRectangle r);
};

CRectangle::CRectangle(int w_, int h_)
{
    w = w_; h = h_;
    totalNumber++; //有对象生成则增加总数
    totalArea += w * h; //有对象生成则增加总面积
}

CRectangle::~CRectangle()
{
    totalNumber--; //有对象消亡则减少总数
    totalArea -= w * h; //有对象消亡则减少总面积
}
```

5.10 静态成员

5.9.2 静态成员函数 【例18】 静态成员函数访问非静态数据成员（续）

```
CRectangle::CRectangle(CRectangle & r)
{
    totalNumber++;
    totalArea += r.w * r.h;
    w = r.w; h = r.h;
}
void CRectangle::PrintTotal(CRectangle r)
{
    cout << r.w << ", " << r.h << endl;
    cout << totalNumber << ", " << totalArea << endl;
}
int CRectangle::totalNumber = 0;
int CRectangle::totalArea = 0;
//必须在定义类的文件中对静态成员变量进行一次声明
int main()
{
    CRectangle r1(4, 6), r2(2, 5);
    //cout << CRectangle::totalNumber; //错误, totalNumber 是私有
    CRectangle::PrintTotal(r1);
    r1.PrintTotal(r1);
    return 0;
}
```


5.9 静态成员

5.9.2 静态成员函数

【例19】设计一个学生类**Student**，统计学生人数

分析：创建学生类对象时，学生人数加1，析构对象时人数减1，因此可以在类内定义一个静态成员变量来统计学生人数，每次调用构造函数对静态成员变量值加1，调用析构函数对静态成员变量减1。同时定义静态成员函数，获取学生人数。

```
#include<iostream>
#include<string>
using namespace std;
class Student {
private:
    string name;//姓名
    int num;//学号
    static int count;//学生人数
public:
    static int getNum();//静态成员函数，返回学生总人数
    Student(string, int);
    ~Student();
```

5.9 静态成员

5.9.2 静态成员函数

【例19】设计一个学生类Student，统计学生人数（续）

```
void print();//输出学生信息
};
//在类外对静态成员初始化:
int Student::count = 0;
Student::Student(string name, int num)
{
    this->name = name;
    this->num = num;
    //调用构造函数，学生人数加1
    count++;
    cout << "当前学生人数为: " << count << endl;
}
Student::~~Student()
{
    //释放对象，学生人数减1
    count--;
}
```

5.9 静态成员

5.9.2 静态成员函数

【例19】设计一个学生类Student，统计学生人数（续）

```
int Student::getNum()//静态成员函数定义
{
    return count;
}
void Student::print()
{
    cout << "学生姓名: " << name << "\t学生学号: " << num << endl;
}
void fun()//普通函数
{
    Student s3("李颖", 1003);
    s3.print();
}
```

5.9 静态成员

5.9.2 静态成员函数

【例19】设计一个学生类Student，统计学生人数（续）

```
int main()
{ //通过类名可直接访问静态成员函数,或静态数据成员
    cout << "当前学生总数为: " << Student::getNum() << endl;
    Student s1("章华", 1001);
    s1.print();
    Student s2("李明", 1002);
    s2.print();
    fun();
    cout << "当前学生人数为: " << Student::getNum() << endl;
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
当前学生总数为: 0
当前学生人数为: 1
学生姓名: 章华  学生学号: 1001
当前学生人数为: 2
学生姓名: 李明  学生学号: 1002
当前学生人数为: 3
学生姓名: 李颖  学生学号: 1003
当前学生人数为: 2
请按任意键继续. . .
```

5.10 常成员与常对象

5.10.1 常对象

- ❑ 如果想要共享某一数据，但又不希望它被修改，可以用const修饰，表示该数据为常量。如果某个对象不允许被修改，也可以用const修饰，则该对象称为**常对象**。
- ❑ const用来**限定类的数据成员和成员函数**，分别称为**类的常数据成员**和**常函数成员**。
- 常对象可以调用常成员函数，不能调用非const成员函数；非const对象，可以调用普通成员函数和常成员函数。
- 定义常对象的一般格式如下：

类名 const 对象名（实参表）；

或者

const 类名 对象名（实参表）；

例如，定义一个日期对象始终为2019年8月11号，则可以定义为：
Date const d1(2019,8,11); //d1为常对象

5.10 常成员与常对象

5.10.1 常对象

- ❑ d1是常对象，d1中的所有成员值都不能被修改。定义常对象时必须赋初值，以下语句是错误的：

Date const d2; //d2为常对象

Date d3(2019,8,15); //d3为普通对象

d2=d3; //错误，常对象定义时必须赋初值

- ❑ 常对象不能调用非常成员函数（除了系统自动调用的构造函数和析构函数），目的是防止这些函数修改对象中数据成员的值。如果有下面的语句：

Date const d4 (2019,8,12) ;

d4.showDate();//错误，试图调用常对象中的非常成员函数

5.10 常成员与常对象

5.10.1 常对象

- 常成员函数可以访问常对象中的数据成员，但不允许其修改常对象中数据成员的值。如果想要修改常对象中某个数据成员的值，可以用mutable进行声明，例如：

mutable int year;

- 把year声明为可变的数据成员，就可以用常成员函数对其值进行修改。

注意：

- 常对象的成员函数不一定是常成员函数；同样的常对象的数据成员不一定是常数据成员。
- 常对象一旦初始化，常对象的数据成员便不允许修改，而不是说常对象的数据成员都是常数据成员。
- 常对象可以调常成员函数和静态成员函数。

5.10 常成员与常对象

5.10.2 常数据成员

- ❑ 用const修饰的数据成员称为常数据成员，其用法与常变量相似。常数据成员在定义时必须进行初始化，并且其值不能被更新（除非数据成员被mutable修饰时，可以被修改）。
- ❑ 常数据成员的声明格式：

数据类型 const 数据成员名;

或者

const 数据类型 数据成员名;

说明：

- 任何函数都不能对常数据成员赋值。
- 构造函数对常数据成员进行初始化时也只能通过初始化列表进行。
- 常数据成员在定义时必须赋值或称为必须初始化。
- 如果有多个构造函数，必须都初始化常数据成员。

5.10 常成员与常对象

5.10.2 常数据成员

注意：

常数据成员不能在声明时赋初始值，必须在构造函数初始化列表进行初始化；普通数据成员在初始化列表和函数体中初始化均可。

5.10 常成员与常对象

5.10.2 常数据成员

【例20】 Student中常数据成员的使用

```
#include<iostream>
#include<string>
using namespace std;
class Student {
private:
    const string  name;    //常数据成员
    const int num;        //常数据成员
    static const  int count; //静态常数据成员
public:
    Student(string i,int a):name(i), num(a)
    {
        cout << "constructor!" << endl;
    }
}
```

5.10 常成员与常对象

5.10.2 常数据成员

【例20】 Student中常数据成员的使用（续）

```
void display()
{
    cout << name << "," << num << "," << count << endl;
}
};
const int Student::count = 0; //静态常数据成员在类外说明和初始化
int main()
{
    Student s1("jane",1001);
    s1.display();
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
constructor!
jane, 1001, 0
请按任意键继续. . .
```

5.10 常成员与常对象

5.10.3 常成员函数

- 用const修饰的成员函数称为**常成员函数**。常成员函数**只能引用本类中的数据成员**（非const数据成员和const数据成员），**而不能修改它们的值**。只有常成员函数才能访问常对象和常量，非常成员函数不能操作常对象。常成员函数的声明格式如下：

```
返回类型 函数名（参数列表） const;
```

- const是加在函数说明后的类型修饰符，它是函数类型的一部分，在实现部分也要带该关键字，调用时不加const关键字。
- const关键字可以用于对重载函数的区分。
- 常成员函数不能更新任何数据成员，也不能调用该类中没有用const修饰的成员函数，只能调用常成员函数和常数据成员。
- 非常对象也可以调用常成员函数，但是当常成员函数与非常成员函数同名时（可以视为函数重载），非常对象是会优先调用非常成员函数。

5.10 常成员与常对象

5.10.3 常成员函数

【例21】常成员函数的使用

```
#include<iostream>
using namespace std;
class A {
private:
    int w, h;
public:
    int getValue() const;//常成员函数
    int getValue();//普通函数
    A(int x, int y)
    {
        w = x, h = y;
    }
    A() {}
};
```

5.10 常成员与常对象

5.10.3 常成员函数

【例21】常成员函数的使用（续）

```
C:\Windows\system32\cmd.exe
12
100
请按任意键继续. . .
```

```
int A::getValue() const //实现部分也带该关键字
{
    //w=10,h=10;        //错误，因为常成员函数不能更新任何数据成员
    return w*h;
}
int A::getValue()
{
    w = 10, h = 10;      //可以更新数据成员
    return w*h;
}
int main()
{
    A const a(3, 4);      //定义常对象
    A c(2, 6);
    cout << a.getValue() << "\n" << c.getValue() << endl;
    return 0;
}
```

5.11 作业

填空题

1. 面向对象的基本特征是：____、____、____、____。
2. 一个对象向另一个对象发出的服务请求称为____。
3. 类定义的关键字是____。类的数据成员通常表示类的____，类的函数成员通常指对类数据的操作，又称为____。
4. 类的访问限定符包括____、____和____。类成员默认访问方式是____。
5. 在创建对象时由系统自动调用的函数称为____，其访问属性应为____，其名和____相同。对象生命周期结束时，系统自动调用的函数称为____。
6. 动态对象的创建通过使用关键字____，建立对象时自动调用____。动态对象撤销使用关键字____，同时自动调用____。
7. 类中所有对象共享的数据称为____，使用关键字____修饰，初始化只能在____进行。

5.11 作业

编程题

- 1、定义一个立方体类Box，计算输出立方体的体积和面积。
- 2、设计一个人员类(Person)。数据成员包括身份证号(IdPerson)、姓名(Name)、性别(Sex)、生日(Birthday)和家庭住址(HomeAddress)。成员函数包括人员信息的录入和显示，还包括构造函数与拷贝构造函数。