# 第3章　类与对象

本章主要内容：

(1) 面向对象程序设计的基本思想。

(2) 类与对象、抽象、封装、继承与多态的含义。

(3) 类、对象的定义和使用。

(4) 类的构造函数、析构函数与复制构造函数。

(5) 静态成员、常成员、友元的应用。

(6) 对象数组、类的组合应用。

从20世纪90年代开始，面向对象程序设计(Object-Oriented Programming，OOP)已成为程序设计的主流技术，该技术提高了大型软件的开发效率，并使程序易于维护。C++语言既支持面向过程的程序设计方法，也支持面向对象的程序设计方法。前面实现的面向过程的程序设计中，编写的程序是由一个或多个函数组成的。从本章开始，我们将学习面向对象的程序设计，在程序中定义类和对象，应用封装、继承和多态实现程序设计。

## 3.1　面向对象程序设计概述

### 3.1.1　面向对象的基本概念

前面介绍的结构化程序设计(Structured Programming，SP)是以解决问题的过程作为出发点，其方法是面向过程的。结构化程序设计是以功能为核心的，基本方法是将问题分解成模块，每个模块尽可能相对独立地解决一个子问题，整个程序是模块功能的集合。在C语言中，一个模块可用一个函数描述，多个函数构成一个源程序，多个源程序构成一个完整的C程序，实现问题的求解。所以，C语言的源程序是由一个个函数构成的。因为各模块可以分别编程，使程序易于阅读、开发和维护。

但是，这种结构化程序设计把程序看成是“数据结构＋算法”，程序中数据与处理这些数据的算法(过程)是分离的。这样，当数据结构改变时，所有相关的处理过程都要进行相应的修改。同时，由于这种分离，导致了数据可能被多个模块使用和修改，难以保证数据的安全性和一致性。另外，当前广泛应用的图形用户界面的应用程序，很难用过程来描述和实现，开发和维护也都很困难。所以，结构化程序设计方法难以适应大型软件和图形界面的应用软件开发。

面向对象程序设计是应用面向对象的思想指导软件开发的过程，简称OO(Object-Oriented)方法，是建立在“对象”概念基础上的方法学。面向对象的思想认为客观世界是由各种各样的对象组成的，每种对象都有各自的内部状态和运动规律，不同对象间的相互作用和联系就构成了各种不同的系统，构成了客观世界。由此，解决现实世界问题的计算机程序也与此相对应，是由一个个对象组成的，这些程序就称为面向对象的程序。

面向对象编程的关注点在于对象本身，对象包含对象的属性和行为两个构成要素。C语言中结构类型的变量包含不同数据类型的成员，用于描述事物的属性，实现了把各种数据的集

合用于表示一个事物，而对数据操作的函数（事物的行为）需要单独定义在结构之外，如果将数据和操作的函数都封装为一个整体，就是 C++ 中的对象。在面向对象程序设计中，对象是构成软件系统的基本单元，并从相同类型的对象中抽象出类，对象是类的实例。类的成员中不仅包含描述类对象属性的数据，还包含对这些数据进行处理的程序代码（这些程序代码被称为对象的行为或操作）。面向对象程序设计是把数据及其操作封装为一个整体对待，数据本身不能被外部函数直接存取。

面向对象程序设计的程序一般由类的定义和类的使用两部分组成，主程序中定义各个类的对象并规定它们之间传递消息的规律，程序中的一切操作都通过向对象发送消息来实现，对象接收到消息后，调用有关对象的行为来完成相应的操作。面向对象方法更接近于人类的自然思维方式，用这种方法开发的软件可维护性和可复用性更高。

面向对象程序设计中的概念主要包括对象、类、抽象、封装、继承、多态、消息等。通过这些概念面向对象的思想得到了具体的体现。

**1. 对象**

1）现实世界的对象

在现实世界中，我们所见到的任何事物都可以看成对象，如一个人、一个工厂、一只狗、一辆汽车、一台计算机等都是对象，这些是有形的具体存在的事物；对象也可以是一个无形的抽象的行为，如一次演出、一次出差等。

对象是现实世界中的实体。对象多种多样，各种对象具有不同的属性特征。有的对象有生命，有的对象没有生命，有的对象有固定的形状，有的对象没有固定的形状……例如，球都具有圆形、半径的属性，而人具有姓名、性别、年龄的属性，即使同一类对象其属性值也是不同的。各个对象也有自己的行为。例如，球的滚动、弹跳，人的走路、眨眼、学习，汽车的加速、刹车和转弯，等等。所以，一个对象是由一组静态的属性和一组动态的行为组成的。对象可以很简单，也可以很复杂，复杂的对象可由若干简单对象组成。

总的来说，现实世界中的对象，具有以下特性。

（1）每个对象都有一个用于与其他对象相区别的名字。

（2）具有某些特征，称它为属性或状态。

（3）有一组行为，决定了对象能干什么。

（4）对象的行为可以分为两类：一类是作用于自身的行为，另一类是作用于其他对象的行为。

2）面向对象中的对象

面向对象中的对象是对现实世界的对象的映射，是由描述其属性的数据和定义在数据上的一组操作组成的实体，即将数据和对数据的操作封装在一起构成一个整体。在C++ 中，每一个对象都是由数据和函数两部分组成的，函数用来实现对数据的操作。

例如，学生“李明”是一个对象，他的数据和他能提供的一组操作表示如下：

对象名：李明

对象的属性：

年龄：20

性别：男

身高：175cm

专业：信息管理与信息系统

对象的操作：

运动

上课

这里的属性说明了李明这个对象的特征，操作说明了李明能做什么。

**2. 类**

1）现实世界中的类

在现实世界中，人们是通过研究对象的属性和观察它们的行为而认识对象的。人们可以把对象分成很多类。类是对一组具有共同属性和行为的对象的抽象。例如，学生这个类是对小学生、中学生、大学生、研究生等学生群体的统称，具体的学生个体则是这个类的一个实例，就是一个学生对象。类和对象是抽象和具体的关系。

学生类还可以再分成小学生类、中学生类……即每一大类中还可再分成若干小类，也就是说，类是分层的。同一类的对象具有许多相同的属性和行为。

2）面向对象中的类

面向对象中的类是一组对象的抽象，为属于该类的全部对象提供了抽象的描述，包括属性和行为两个主要部分。类是创建对象的模板，它没有具体的值和具体的操作，只有以它为模板创建的对象才有具体的值和操作。类用类名来相互区别。对象是类的一个实例，有了类才能创建对象。

在 C++ 中，就是用类来描述对象的，类是对现实世界的对象进行抽象得到的。例如，在现实世界中，同是学生类的“张平”和“李平”，有许多共同点，但肯定也有许多不同点。当用 C++ 描述时，对“张平”和“李平”这两个同类的对象进行抽象，得到相同的属性和行为，然后描述为学生类的两部分：数据（相当于属性）和对数据的操作（相当于行为）。例如，数据可以是姓名、性别、年龄、住址等，而对数据的操作可以是读取或设置其名字、年龄等。抽象出了学生类以后，就可以在程序中描述具体的“张平”和“李平”这两个对象。

从程序设计的观点来说，类就是数据类型，是用户自定义的数据类型。这种类型是用户根据具体问题的需要而定义的，也就是说，类与具体问题相适应。我们可以通过定义所需要的类，来扩展程序设计语言解决问题的能力。

在 C++ 中，把描述类的属性的数据称为数据成员，把描述行为的操作称为成员函数。

**3. 消息与方法**

在软件系统中，对象与对象之间存在一定的联系，这种联系通过消息的传递来实现。一个对象向另一个对象发出的请求称为消息。当某个行为作用于对象时，就称该对象执行了一个方法，方法定义了一系列的计算步骤，在 C++ 中称为成员函数。简单理解，这种消息传递机制就是面向过程程序设计的过程调用。消息传递的实质就是方法的调用，即向对象发送消息就是调用对象的方法。

消息的特性为：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，所做出的响应也可以是不同的；消息的发送可以不考虑具体的接收者；对象可以响应消息，也可以不响应。

### 3.1.2 面向对象的基本特征

**1. 抽象**

抽象（Abstraction）是人类认识客观世界的基本手段，它是从许多事物中舍弃个别的、非

本质性的特征，抽取共同及本质性的特征的过程。抽象是人类对事物进行分类的最基本的方法和手段。面向对象程序设计中的抽象是对一类对象进行分析和认识，经过概括，抽出一类对象的公共性质，并加以描述的过程。

对一个事物的抽象一般包括两方面：数据抽象和行为抽象。数据抽象是对对象的属性和状态的描述，即对使对象之间相互区别的特征量的描述。行为抽象是对数据需要进行的处理的描述，它描述了一类对象的共同行为特征，使一类对象具有共同的功能，所以，又称行为抽象为代码抽象。

例如，要设计绘制圆的图形的程序，通过分析可知，圆是这个问题中的唯一事物。对于具体的圆，有的大些，有的小些，圆的位置也不尽相同，但可用三个数据（即圆心的横、纵坐标和圆的半径）描述圆的位置和大小，这就是对圆这个事物的数据抽象。由于抽象后没有具体的数据，它不能是一个具体的圆，只能代表一类事物，即圆类。要能画出圆，该程序还应有设置圆形位置、半径大小、绘制圆形的功能，这就是对圆这个事物的行为抽象。

由上面的例子可以看出，类的数据成员的实质就是解决问题所需要的数据，它是数据抽象的结果；而成员函数的实质是完成对类中的这些数据进行加工处理的代码，它是类的行为，用行为抽象来描述。所以，抽象性是面向对象的核心。

在面向对象的分析过程中，抽象原则具有两方面的含义。

（1）尽管问题域中的事物很复杂，但在分析过程中并不需要了解和描述它们的全部，只需要分析研究其中与系统目标有关的事物及其本质性特征。对于那些与系统目标无关的特征和许多具体的细节，即使有所了解，也应该舍弃。

（2）通过舍弃个体事物在细节上的差异，抽取其共同特征而得到一批事物的抽象概念，即抽象出类的概念。

抽象是面向对象方法中使用最为广泛的原则，例如，程序中的对象是对现实世界中事物的抽象：类是对象的抽象；数据成员是事物静态特征的抽象；成员函数是事物动态特征的抽象等。

### 2. 封装

封装（Encapsulation）就是把一个事物包装起来，使外界不了解它的内部的具体情况。在面向对象的程序设计中，封装就是把相关的数据和代码组合成一个有机的整体，形成数据和操作代码的封装体，对外只提供一个可以控制的接口，内部大部分的实现细节对外隐蔽，达到对数据访问权的合理控制。封装使程序中各部分之间的相互联系达到最小，提高了程序的安全性，简化了程序代码的编写工作，是面向对象程序设计的重要原则。

面向对象程序设计的封装机制是通过对象实现的。对象中的成员不仅包含数据，也包含对这些数据进行处理的操作代码。对象中的成员可以根据需要定义为公有的或私有的，私有成员在对象中被隐蔽起来，对象以外的访问被拒绝；公有成员提供了对象与外界的接口，外界只能通过这个接口与对象发生联系。可见，对象有效地实现了封装的两个目标：对数据和行为的结合及信息隐蔽。

抽象和封装是互补的。一个好的抽象有利于封装，封装的实体则帮助维护抽象的完整性，但抽象先于封装。

### 3. 继承

继承（Inheritance）是指通过继承已有类的成员而定义生成新的类。其中，已有的类称为基类或父类，新定义的类称为派生类或子类。面向对象程序设计提供了类的继承性，给创建派

生类提供了一种方法：创建派生类时，不必重新描述基类的所有成员（数据和操作），只需让它继承基类的成员，然后描述与基类不同的那些成员。也就是说，派生类的成员由继承来的和新添加的两部分组成，继承允许派生类使用基类的数据和操作，还可以拥有自己的数据和操作。所以，继承是通过对已有类增添不同的特性来派生出多种不同的特殊类，从而使得类与类之间建立了层次结构关系，为软件复用提供了有效的途径。

在某些情况下，一个类会有多个派生类，派生类比原有的类更加具体化。继承避免了对基类和派生类之间共同属性和行为进行重复的描述，简化了人们对事物的认识和描述，通过软件复用提高软件的开发效率。

**4. 多态性**

多态性（Polymorphism）是面向对象程序设计的重要特性之一，是指不同的对象收到相同的消息时产生不同的操作行为，或者说同一个消息可以根据发送消息的对象的不同而采用多种不同的操作行为。例如，运算符 & 既可表示取地址运算符，也可表示引用运算符，系统会根据运算符出现的位置，判断出代表的操作；当单击不同的对象时，各对象就会根据自己的理解做出不同的动作，产生不同的行为，这就是多态性。简单地说，多态性就是一个接口，有多种方式。

C++ 语言支持两种多态性：编译时的多态性和运行时的多态性。编译时的多态性通过重载实现。运行时的多态性通过虚函数实现。

### 3.1.3 面向对象的程序设计语言简介

面向对象程序设计达到了软件工程的三个主要目标：重用性、灵活性和扩展性。它克服了传统的结构化方法在建立问题系统模型和求解问题时存在的缺陷，提供了更合理、更有效、更自然的方法，更适合于大型软件的开发。

面向对象程序设计语言的鼻祖是 20 世纪 60 年代开发的 Simula 67，它提供了对象、类、继承等概念，提出了面向对象的术语，奠定了面向对象语言的基础。它的主要用途是进行建模仿真。

20 世纪七八十年代期间的 Smalltalk 语言，是最有影响的面向对象语言之一。它包括了 Simula 的面向对象的所有特征，而且数据封装比 Simula 更严格。

Object-C 是在 1983 年以后开发的、对 C 进行扩充而形成的面向对象的语言，但它的语法更像 Smalltalk 语言。它的扩充主要是新引入的构造和运算符，用它们来完成类定义和消息传递。

C++ 语言在 C 语言的基础上扩充了对面向对象的支持，既支持面向过程的设计方法，又支持面向对象的设计方法，还有丰富的开发环境，因而得到广泛的应用。

C#语言是微软公司于 2000 年推出的一种纯面向对象的语言，是基于.NET Framework 开发的核心语言。.NET Framework 是微软公司的新一代技术平台，便于开发者建立 Web 应用程序和 Web 服务。C#不仅继承了 C/C++ 语言的强大功能，并且在语法上保持一致，使学习者容易入门。

Java 语言是原 Sun 公司于 1995 年推出的适用于分布网络环境的面向对象语言，它采用与 C++ 语法基本一致的形式，将 C++ 中与面向对象无关的部分去掉，使其语义成为纯面向对象的语言。它使应用程序独立于异构网络上的多种平台，具有能编译或解释执行、连接简单、支持语言级的多线程等特点，也是一种广泛使用的面向对象的语言。

## 3.2 类与对象的定义

类是面向对象程序设计的基础和核心，也是实现数据抽象的工具。类实质上是用户自定义的一种特殊的数据类型，与一般的数据类型相比，它不仅包含相关的数据，还包含能对这些数据进行处理的函数，同时，这些数据具有隐蔽性和封装性。

### 3.2.1 类的定义

#### 1. 从结构到类

问题引入：在学生信息管理中描述一个学生的信息包括学号、姓名、年龄等。这些信息具有不同的数据类型，并且作为一个整体进行描述，就要用到结构类型。结构是用户自定义的数据类型，可由不同类型的数据成员组成。结构变量在内存中占有一片连续的存储空间。

下面用 struct 定义一个描述学生基本信息的结构类型 Student，并使用该结构定义两个学生变量 stu1 和 stu2。

```
struct Student
{
    int num;
    char name[10];
    int age;
};
Student stu1, stu2;
```

对结构变量成员访问有两种形式。

(1) 用成员(圆点)运算符。

```
结构变量名.成员
```

(2) 用指针的间接访问(*)或指向运算符(->)。

```
(*指针).成员
```

或

```
指针->成员
```

例如：

```
stu1.num=20120101;
Student *pstu=&stu2;
(*pstu).num=20120102;
pstu->age=20;
```

实际上，在 C++ 中的结构类型不仅可以包含各种数据成员，还可以包含函数。如例 3-1 所示，就是在结构类型 Student 中增加对数据操作的函数 Init()和 Disp()。

【例 3-1】 学生结构类型示例。

```
//D:\QT_example\3\Example3_1.cpp
#include <iostream>
using namespace std;
struct Student
{
    //数据成员
    int num;
```

```
    char name[10];
    int age;
    //成员函数
    void Init()
    {   cout<<"Please input the student's number:";
        cin>>num;
        cout<<"Please input the student's name:";
        cin>>name;
        cout<<"Please input the student's age:";
        cin>>age;
    }
    //成员函数
    void Disp()
    {   cout<<"Information of Student:"<<endl;
        cout<<"Student's number:"<<num<<endl;
        cout<<"Student's name:"<<name<<endl;
        cout<<"Student's age:"<<age<<endl;
        cout<<endl;
    }
};
int main()
{
    Student stu1, stu2;
    stu1.Init();                //结构变量调用成员函数实现数据成员的输入
    stu1.Disp();                //结构变量调用成员函数实现数据成员的输出
    Student *pstu=&stu2;
    pstu->Init();               //指针方式访问结构变量的成员函数
    pstu->Disp();               //指针方式访问结构变量的成员函数
    return 0;
}
```

程序运行结果为：

```
Please input the student's number:201701
Please input the student's name:liping
Please input the student's age:19
Information of Student:
student's number.:201701
student's name:liping
student's age:19

Please input the student's number:201702
Please input the student's name:wangli
Please input the student's age:18
Information of Student:
student's number:201702
student's name:wangli
student's age:18
```

例 3-1 结构中的数据和函数都是结构的成员，通常称为数据成员和成员函数。对成员函数的访问形式与数据成员类似，只是应按照函数的调用形式，具体如下：

```
结构变量.成员函数名(实参表);
```

或

```
(*结构变量指针).成员函数名(实参表);
```

或

```
结构变量指针->成员函数名(实参表);
```

### 2. 类的定义

C++ 提供了一种比结构类型更安全的数据类型——类，类与上面包含成员函数的结构定义类似。

【例 3-2】 用学生类实现例 3-1。

```
//D:\QT_example\3\Example3_2.cpp
#include <iostream>
using namespace std;
class Student
{
    //数据成员
    int num;
    char name[10];
    int age;
    //成员函数
    void Init()
    {   cout<<"Please input the student's number:";
        cin>>num;
        cout<<"Please input the student's name:";
        cin>>name;
        cout<<"Please input the student's age:";
        cin>>age;
    }
    //成员函数
    void Disp()
    {   cout<<"Information of Student:"<<endl;
        cout<<"Student's number:"<<num<<endl;
        cout<<"Student's name:"<<name<<endl;
        cout<<"Student's age:"<<age<<endl;
        cout<<endl;
    }
};
int main()
{
    Student stu1, stu2;                 //用类 Student 定义对象 stu1 和 stu2
    stu1.Init();                        //对象调用成员函数,但出现编译错误
    stu1.Disp();                        //编译错误
    Student *pstu=&stu2;                //定义指向对象的指针
    pstu->Init();                       //编译错误
    pstu->Disp();                       //编译错误
    return 0;
}
```

例 3-2 中，关键字 struct 改为 class 可用来定义一个类，有了类就可以定义具体的对象 stu1 和 stu2。就如同结构类型与结构变量一样，先有数据类型再有变量，类就是一种自定义的数据类型，而对象就相当于该类型的一个变量。在类外用对象 stu1 和 stu2 访问成员 Init() 和 Disp()，会出现如下编译错误：

```
E:\QT_example\3\Example3_2…10: error: 'void Student::Init()' is private
E:\QT_example\3\Example3_2…19: error:'void Student::Disp()' is private
```

这是因为，类内成员在不指定访问权限时默认为私有(private)，即不允许在类外直接访问，这是类与结构的不同，结构类型默认的访问权限是公有的(public)。所以，该程序中需指定成员函数的公有访问权限，修改后如例 3-3。

【例 3-3】 修改例 3-2,声明公有成员函数。

```
//D:\QT_example\3\Example3_3.cpp
#include <iostream>
using namespace std;
class Student
{
    //数据成员
    int num;
    char name[10];
    int age;
    //公有成员函数
public:
    void Init()
    {   cout<<"Please input the student's number:";
        cin>>num;
        cout<<"Please input the student's name:";
        cin>>name;
        cout<<"Please input the student's age:";
        cin>>age;
    }
    void Disp()
    {   cout<<"Information of Student:"<<endl;
        cout<<"Student's number:"<<num<<endl;
        cout<<"Student's name:"<<name<<endl;
        cout<<"Student's age:"<<age<<endl;
        cout<<endl;
    }
};
int main()
{
    Student stu1, stu2;                    //用类 Student 定义对象 stu1 和 stu2
    stu1.Init();                           //通过对象名调用成员函数
    stu1.Disp();
    Student *pstu=&stu2;                   //定义指向对象的指针
    pstu->Init();                          //通过对象指针调用成员函数
    pstu->Disp();
    return 0;
}
```

程序运行结果同例 3-1。本例中,数据成员仍然为私有,但成员函数声明为公有,所以,可以在类外通过对象来调用公有成员,通过公有成员实现对类内私有数据的输入和输出,实现对象中私有成员的隐蔽。通常为了保护数据的安全性,将数据声明为私有,而设置公有成员函数对其进行访问,所有公有成员函数是类与外界的接口。

综上所述,类定义的一般形式如下:

```
class 类名
{
    public:
        公有数据成员和成员函数
    private:
        私有数据成员和成员函数
    protected:
        保护数据成员和成员函数
};
```

类的定义由类头部和类体两部分组成。类头部由关键字 class 开头,然后是类名,其命名

规则与一般标识符的命名规则一致，有时可能有附加的命名规则，例如，美国微软公司的 MFC 类库中的所有类均是以大写字母 C 开头的。类体中包含了所有成员的声明，并放在一对花括号中。

类中包含的数据和函数统称为成员，数据称为数据成员，函数称为成员函数，它们都有自己的访问权限。对类内成员的访问控制是通过类的访问权限实现的。

类内成员的访问权限分为以下 3 种。

(1) private：声明私有成员。私有成员只能被类内的成员函数访问，类外的任何对象对它的访问都是不允许的。私有成员是对象中被隐蔽的部分，通常是描述该类对象属性的数据成员，这些数据成员用户无法访问，只有通过成员函数或某些特殊说明的函数才可访问，它体现了对象的封装性。当声明中省略 private 时，系统默认该成员为私有成员。

(2) protected：声明保护成员，一般情况下与私有成员的含义相同，它们的区别表现在类的继承中对新类的影响不同。保护成员的具体用法将在第 5 章中介绍。

(3) public：声明公有成员。公有成员可以被类外的对象访问，它提供了外部程序与类的接口功能。

**注意：**

(1) 类的定义也是一个语句，所以最后的分号不能丢掉；否则，会产生难以理解的编译错误。

(2) 说明类成员访问权限的关键字 private、protected 和 public 可以按任意顺序出现任意多次，但一个成员只能有一种访问权限。为使程序更加清晰，应将私有成员和公有成员归类放在一起。

(3) 类的成员默认为私有，结构的成员默认为公有。

(4) 数据成员可以是任何数据类型，但不能用自动(auto)、寄存器(register)和外部(extern)来说明。

(5) 成员函数可以在类内定义，也可在类内声明原型而在类外定义。

### 3. 类的成员函数

类的成员函数与普通函数的形式基本一样，也包括函数名、返回值类型和函数参数，但它是属于一个类的成员，是专门对类内数据进行操作的。成员函数只能通过所属的对象或类来调用，而不能直接调用。成员函数通常有一部分是公有的，一部分是私有的。公有的成员函数可在类外被访问，也称为类的接口。我们可以为各个数据成员和成员函数指定合适的访问权限。成员函数又称为方法，成员函数是 C++ 中的术语，方法是面向对象方法中的术语，它们是同一个实体。

成员函数有以下两种定义方式。

(1) 一种是在类内定义。例如前面的例子，所有成员函数在类体内定义。此时编译系统将函数作为内联函数进行处理，即将这些函数隐含地声明为内联成员函数。与普通内联函数的处理方法相同，内联成员函数也是在编译时将调用语句替换为函数代码，从而减少函数调用的开销。

(2) 一种是在类外定义，即在类体内给出函数原型的声明，而在类体外完成对函数的定义。其一般形式是：

```
返回类型  类名::函数名(参数表)
{
```

```
    //函数体
}
```

【例 3-4】 类外定义成员函数，类内先声明公有成员，再声明私有成员。

```
//D:\QT_example\3\Example3_4.cpp
#include <iostream>
using namespace std;
class Student
{
public:
    void Init();                        //成员函数声明
    void Disp();                        //成员函数声明
private:
    int num;
    char name[10];
    int age;
};
void Student::Init()                    //成员函数定义
{   cout<<"Please input the student's number:";
    cin>>num;
    cout<<"Please input the student's name:";
    cin>>name;
    cout<<"Please input the student's age:";
    cin>>age;
}
void Student::Disp()                    //成员函数定义
{   cout<<"Information of Student:"<<endl;
    cout<<"Student's number:"<<num<<endl;
    cout<<"Student's name:"<<name<<endl;
    cout<<"Student's age:"<<age<<endl;
    cout<<endl;
}
int main()
{
    Student stu1, stu2;                 //用类 Student 定义对象 stu1 和 stu2
    stu1.Init();                        //通过对象名调用成员函数
    stu1.Disp();
    Student *pstu=&stu2;                //定义指向对象的指针
    pstu->Init();                       //通过对象指针调用成员函数
    pstu->Disp();
    return 0;
}
```

本例中，函数 Init()和 Disp()都在类外定义，但它们都属于类 Student 的成员函数，所以在类外定义时函数名前要加上类名 Student::进行限定。

**说明：**

(1) 类外定义的函数名前必须加上前缀“类名::”，其中，“::”称为作用域运算符。

(2) 在类内声明成员函数的函数原型时，参数表中的参数可以只说明参数的数据类型，可省略参数名。但在类外定义成员函数时，参数表中的参数不但要说明参数的数据类型，而且要指定参数名。

(3) 定义成员函数时，其返回值类型必须与函数原型说明中的返回类型一致。

(4) 可以将类外定义的成员函数指定为内联函数，即在声明或定义时前面加上关键字 inline 声明。

例如，将例 3-4 中的成员函数指定为内联函数。

```
class Student
{
public:
    inline void Init();                    //内联成员函数声明
    inline void Disp();                    //内联成员函数声明
private:
    int num;
    char name[10];
    int age;
};
void Student::Init()                       //内联成员函数定义
{   cout<<"Please input the student's number:";
    cin>>num;
    cout<<"Please input the student's name:";
    cin>>name;
    cout<<"Please input the student's age:";
    cin>>age;
}
void Student::Disp()                       //内联成员函数定义
{   cout<<"Information of Student:"<<endl;
    cout<<"Student's number:"<<num<<endl;
    cout<<"Student's name:"<<name<<endl;
    cout<<"Student's age:"<<age<<endl;
    cout<<endl;
}
```

特别说明：将简单的成员函数声明为内联函数可以提高程序的运行效率，但必须将类的声明和内联函数的定义都放在同一个文件（或同一个头文件）中，否则编译时无法进行置换。

### 3.2.2 对象的定义与使用

类是对具有相同数据成员（相同的内部存储结构）和相同操作的一组对象的抽象描述，是一种数据类型，类中不存储具体的数据值，定义类时系统不会分配存储空间。只有定义了类的一个具体实例——对象，系统才给对象分配存储空间，对象才能存储具体的数据值。程序中须在先定义了类之后才能定义具体的对象。

#### 1. 对象的定义

定义对象如同前面定义一般数据类型的变量一样，定义的一般格式如下：

```
类名 对象名列表;
```

例如：

```
Student stu1, stu2;                    //定义了类 Student 的两个对象 stu1 和 stu2
```

也可以定义指向对象的指针或引用，还可以定义对象数组，例如：

```
Student stu, *ps;                      //定义 Student 类的对象 stu 和指针 ps
ps=&stu;                               //指针 ps 指向对象 stu
Student &stu1=stu;                     //定义对象 stu 的引用 stu1
```

也可以在定义类的同时直接定义对象，其方法是，在类声明的右花括号的后面直接写出该类的对象名表，例如：

```
class Student
{…
}stu1, stu2;
```

如果类的定义位于函数外部，则用这种方法定义的对象是全局对象，在它的生命周期内，任何函数都可以使用它。

### 2. 对象成员的访问

对象的公有成员是提供给外部的接口，无论是数据成员还是成员函数，只要是公有的就可以通过对象进行访问。

对象成员的访问形式与访问结构成员的形式相同，可以使用圆点运算符“.”和指向运算符“->”，如例 3-3 和例 3-4 所示。一般访问的形式总结如下。

（1）通过对象名和“.”运算符的访问形式。

数据成员：

```
对象名.数据成员名
```

成员函数：

```
对象名.成员函数名(实参表)
```

（2）通过指针与“*”和“->”运算符的访问形式。

数据成员：

```
指针名->数据成员名
```

或

```
(*指针名).数据成员名
```

成员函数：

```
指针名->成员函数名(实参表)
```

或

```
(*指针名).成员函数名(实参表)
```

【例 3-5】 日期类与对象的定义和成员访问。

```
//D:\QT_example\3\Example3_5.cpp
#include <iostream>
using namespace std;
class Date
{
public:
    void Set(int y, int m, int d);              //带参成员函数声明
    int GetYear()
    { return year; }
    int IsLeapYear();
    void Print();
private:
    int year, month, day;
};
void Date:: Set (int y,int m, int d )
{ year =y; month =m; day =d; }
int Date::IsLeapYear()
{ return ( year%4 ==0 && year%100!=0 )||( year%400==0); }
void Date::Print()
{ cout<<year<<"."<<month<<"."<<day<<endl; }
int main()
{
```

```
    Date today, tomorrow;                    //定义两个对象 today 和 tomorrow
    today.Set (2017,1, 21);                  //设置 today 日期为 2017-1-21
    cout<<"today: ";
    today.Print();
    cout<<today.GetYear();                   //输出 today 的年
    if( today.IsLeapYear() )                 //判断 today 是否为闰年
        cout<<" is a leap year!"<<endl;
    else
        cout<<" is not a leap year!"<<endl;
    tomorrow.Set (2017,1,22);                //设置日期为 2017-1-22
    Date *pd=&tomorrow;
    cout<<"tomorrow: ";
    pd->Print();                             //通过指针访问对象成员
    return 0;
}
```

程序运行结果为：

```
today: 2017.1.21
2017 is not a leap year!
tomorrow: 2017.1.22
```

本例中，定义了两个日期对象 today 和 tomorrow，并通过调用公有接口 Set()完成数据成员的赋值，例如，“today.Set(2017, 1, 21);”的参数传递过程如图 3-1 所示。不同的对象拥有不同的数据成员值，各个对象通过公有接口访问自己的私有成员。

### 3. 对象的存储与 this 指针

在例 3-5 中，Date 类的两个对象 today 和 tomorrow 具有不同的数据成员值，但它们调用的是同一个 Set()函数实现赋值。也就是说，各个对象的数据成员占用不同的存储空间，存储各自的数据成员值，但它们的成员函数代码却是相同的，如果为每个对象的成员函数都分配一份存储空间显然是一种浪费。为此，C++ 编译系统在为对象分配存储空间时，只分配其数据成员所占用的存储空间，而不包括函数代码部分。从物理角度看，成员函数代码是存储在对象空间之外的，而且只在内存中保存一份，如图 3-2 所示。但从逻辑角度看，不同对象调用的相同成员函数，传递的参数和执行的结果却是不同的。同一段函数代码是如何识别所操作的不同对象的数据呢？这是因为，系统自动为成员函数添加了一个名称为 this 的指针参数。

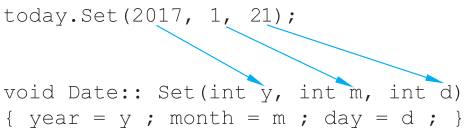


图 3-1 对象 today 的成员函数的参数传递

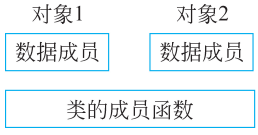


图 3-2 类的成员函数的存储

例如，Date 类的 Set() 函数具有一个隐含的参数“Date * const this”，通常 this 指针是成员函数的第一个参数，相当于：

```
void Date::Set( Date * const this,int y, int m, int d )
{ this->year =y; this->month =m; this->day =d; }
```

语句“today. Set(2017,1,21);”可理解为：

```
today.Set(&today,2017, 1, 21);          //系统自动为隐含的 this 指针传递 today 的地址
```

语句“tomorrow.Set(2017,1,22);”可理解为：

```
tomorrow.Set(&tomorrow,2017, 1, 22);    //系统自动为 this 指针传递 tomorrow 的地址
```

但是，实际程序中是不可显式定义 this 参数以及实参 &today 和 &tomorrow 的，它们是由系统自动添加的并且是隐藏的。

通过参数传递，使得 this 指针指向当前调用的对象，在成员函数中通过 this 指针可实现对当前调用对象的成员的访问，因此，函数体内的语句：

```
year=y;
```

等价于：

```
this->year=y;                    //此处的 this 指针可显式使用
```

**注意：**

(1) 类的所有非静态成员函数都自动拥有一个隐含 this 指针参数，形式为：

```
class_type * const this          // class_type 表示类类型
```

(2) this 是一个常指针，当一个对象调用其成员函数时，this 被赋值为当前调用对象的地址，在本次函数执行期间不能被修改，以确保 this 指向当前调用对象。

(3) this 指针作为一个隐含参数，是由系统自动设置的，它不能被显式声明，但可以在程序中显式使用。

**【例 3-6】** 输出 this 指针的值。

```
//D:\QT_example\3\Example3_6.cpp
#include <iostream>
using namespace std;
class Sample
{
public:
    void Set(char c)
    { ch =c; }
    void Disp()
    {cout<<",this ="<<this<<endl; }
private:
    char ch;
};
int main()
{
    Sample a, b;
    a.Set('a');
    b.Set('b');
    cout<<"&a="<<&a;
    a.Disp();
    cout<<"&b="<<&b;
    b.Disp();
    return 0;
}
```

程序运行结果为：

```
&a=0x28ff3f,this=0x28ff3f
&b=0x28ff3e,this=0x28ff3e
```

程序中的 Disp()函数直接输出 this 的值，与当前调用对象的地址是相同的。this 指针显式使用主要出现在运算符重载和自引用等场合。

### 3.2.3 类的作用域

一个标识符在程序中有效的作用区域称为该标识符的作用域。类的作用域是指在类的声

明中一对花括号所形成的区域(包括类外定义的成员函数),一个类的所有成员都在该类的作用域内。在类的作用域内,类的任何成员都可以引用类中的其他成员;但在类的作用域以外,对类的成员的引用则要受到一定的限制,有时甚至是不允许的。这充分体现了类的封装性。

【例 3-7】 类作用域示例。

```
//D:\QT_example\3\Example3_7.cpp
#include <iostream>
using namespace std;
class Sample
{
public:
    void Set(char c)
    { ch=c; }
    void Disp()
    { cout<<"ch="<<ch<<", this="<<this<<endl; }
private:
    char ch;
};
void Func()                    //全局函数
{   ch='c';                    //错误:'ch' was not declared in this scope
    Disp();                    //错误:'Disp' was not declared in this scope
}
int main()
{
    Sample a, b;
    a.ch='a';                  //错误:'char Sample::ch' is private
    b.ch='b';                  //错误:'char Sample::ch' is private
    a.Disp();
    b.Disp();
    return 0;
}
```

本例中,全局函数 Func()中试图直接访问类的成员 ch 和 Disp(),出现编译错误,因为此处的 Func()函数是类外的一般函数,不在类的作用域内,不能直接访问类的成员,而在类外也没有单独声明这两个标识符,所以,系统提示是未声明的标识符。正确的做法是在成员名前加上对象名限定,即 a.Disp(),也就是通过对象访问这些成员。但需要注意,在类外,通过对象只能访问公有成员,而不能访问私有成员,如 a.ch 就会出现编译错误。一般地,在类外通过公有成员来操作私有成员,例如为实现给对象 a.ch 赋值,就需通过调用 Set()接口来实现,正确示例如例 3-8 所示。

【例 3-8】 类作用域示例。

```
//D:\QT_example\3\Example3_8.cpp
#include <iostream>
using namespace std;
class Sample
{
public:
    void Set(char c)
    { ch=c; }
    void Disp()
    { cout<<"ch="<<ch<<", this="<<this<<endl; }
private:
    char ch;
};
```

```
void Func(Sample s)     //全局函数
{   s.Set('c');         //通过对象 s 引用成员
    s.Disp();
}
int main()
{
    Sample a, b,c;
    a.Set('a');         //对象 a 通过调用公有成员函数 Set()为私有数据成员 ch 赋值
    b.Set('b');
    a.Disp();
    b.Disp();
    Func(c);
    return 0;
}
```

### 3.2.4 类的封装性和信息隐藏——公有接口与私有实现的分离

把数据和操作数据的函数封装在一起构成了类。在声明类时，一般把一部分成员函数声明为公有的，而把数据成员声明为私有的，外界只能通过公有成员函数访问数据，起到信息隐藏的作用。在前面的程序中，是将类的声明和成员函数的定义直接写在程序的开头，如果一个类要被多个程序使用，就要重复定义该类。

实际上，在面向对象的程序开发中，一般做法是将类的声明（其中包括成员函数的声明）放在一个头文件中，而将函数定义（实现）部分放在另一个文件中，即将接口与实现分离。使用类时，只要把相关的头文件包含进来即可，由于在头文件中包含了类的声明，所以在程序中就可以用该类来定义对象，由于在类体中包含了对成员函数的声明，在程序中就可以调用这些对象的公用成员函数。

例 3-4 可用 3 个文件组成的工程实现。

（1）头文件 student.h：类的声明部分。

```
//D:\QT_example\3\ Example3_9\ student.h
#ifndef STUDENT_H               //避免多次包含
#define STUDENT_H
class Student
{ public:
      void Init();              //成员函数声明
      void Disp();              //成员函数声明
  private:
      int num;
      char name[10];
      int age;
};
#endif
```

（2）源文件 student.cpp：成员函数定义，类的实现部分。

```
//D:\QT_example\3\ Example3_9\ student.cpp
#include <iostream>
using namespace std;
#include "student.h"                    //将类声明头文件包含进来
void Student::Init()                    //成员函数定义
{   cout<<"Please input the student's number:";
    cin>>num;
    cout<<"Please input the student's name:";
```

```
    cin>>name;
    cout<<"Please input the student's age:";
    cin>>age;
}
void Student::Disp()                          //成员函数定义
{   cout<<"Information of Student:"<<endl;
    cout<<"Student's number:"<<num<<endl;
    cout<<"Student's name:"<<name<<endl;
    cout<<"Student's age:"<<age<<endl;
    cout<<endl;
}
```

(3) 源文件 main.cpp：包括主函数的源文件。

```
//D:\QT_example\3\ Example3_9\main.cpp
#include <iostream>
using namespace std;
#include "student.h"                          //将类声明头文件包含进来
int main()
{
    Student stu1, stu2;                       //用类 Student 定义对象 stu1 和 stu2
    stu1.Init();                              //通过对象名调用成员函数
    stu1.Disp();
    Student *pstu=&stu2;                      //定义指向对象的指针
    pstu->Init();                             //通过对象指针调用成员函数
    pstu->Disp();
    return 0;
}
```

本程序需要新建一个工程名为 Example3_9 的工程，并加入上面的三个文件，其中，对源文件 student.cpp 和 main.cpp 进行编译，分别得到目标文件 student.o 和 main.o；然后连接得到可执行文件 Example3_9.exe，运行结果与例 3-4 相同。

📖**注意**：由于将头文件 student.h 放在用户当前目录中，因此包含时用"student.h"而不用＜student.h＞，否则编译时会找不到此文件。

在实际应用中，通常是将若干常用的功能相近的类声明集中在一起，形成类库。类库有两种：一种是 C++ 编译系统提供的标准类库；一种是用户根据需要开发的用户类库，提供给自己和自己授权的人使用，这称为自定义类库。

类库由两部分组成：一是类声明的头文件；二是已编译过的实现部分的目标文件。开发商把用户所需的各种类的声明按类放在不同的头文件中，同时对包含成员函数定义的源文件进行编译，得到成员函数定义的目标代码。软件商向用户提供这些头文件和类的实现的目标代码(不提供函数定义的源代码)。用户在使用类库中的类时，只需将有关头文件包含到自己的程序中，编译后就会自动与库中的目标代码相连接，最后生成可执行文件。这和在程序中使用 C++ 系统提供的标准函数的方法是一样的。

由于类库的出现，用户可以像使用零件一样方便地使用在实践中积累的通用的或专用的类，这就大大减少了程序设计的工作量，有效地提高了工作效率。

## 3.3 构造函数与析构函数

构造函数和析构函数都是类内特殊的成员函数。类是一种自定义的数据类型，可以很简单也可以很复杂，当声明一个类的对象时，编译系统需要为对象分配内存空间，进行必要的初

始化，这个工作是由构造函数完成的。与构造函数对应的是析构函数，当对象被撤销时，就要回收内存空间，并做一些善后工作，这个任务是由析构函数完成的。而且，构造函数与析构函数都是由系统自动调用的。

### 3.3.1 构造函数

类是一种抽象数据类型，它不被分配内存空间，不能存放具体数据。所以，不能在类声明中给数据成员赋初始值。在定义类的对象时，应考虑为其数据初始化，使该对象的数据具有确定的初始值，前面的例子中是通过调用公用接口（如例 3-4 中的 Init()函数）为数据赋值，如果忘记调用该函数，对象的数据就是不确定的值，会引起运行错误。C++ 中提供了构造函数来完成对象的初始化工作。构造函数是属于某个类的成员函数，不同的类有不同的构造函数。构造函数可由用户自己定义，也可由系统自动生成。

#### 1. 构造函数的定义与作用

构造函数被声明为类的公有成员，其作用是为类的对象分配内存空间，并进行初始化。构造函数不同于一般的成员函数，它具有以下特性。

（1）构造函数的名字与类名相同。构造函数不能由用户任意命名，创建对象时系统根据函数名进行调用。

（2）构造函数的参数可以是任何数据类型，但它没有返回值，不能为它定义返回类型，包括 void 类型在内。

（3）构造函数不能被显式地调用，在定义对象时，编译系统会自动地调用构造函数完成对象内存空间的分配和初始化工作。

**【例 3-9】** 为日期 Date 类定义构造函数完成初始化。

```
//D:\QT_example\3\ Example3_9.cpp
#include <iostream>
using namespace std;
class Date
{
public:
    Date()                                  //构造函数
    { year =2017; month =7;day =10; }
    void Set(int y, int m, int d);          //带参成员函数声明
    void Print();
private:
    int year,month,day;
};
void Date:: Set(int y, int m, int d)
{ year =y; month =m; day =d; }
void Date::Print()
{ cout<<year<<"."<<month<<"."<<day<<endl; }
int main()
{
    Date today, tomorrow;                   //定义两个对象 today 和 tomorrow
    today.Print();
    tomorrow.Print();
    tomorrow.Set (2017,7, 11);              //设置日期为 2017-7-11
    tomorrow. Print();
    return 0;
}
```

程序运行结果为：

```
2017.7.10
2017.7.10
2017.7.11
```

本例中，定义对象 today 和 tomorrow 时，系统自动调用构造函数 Date()，为其数据成员 year、month、day 分别赋初值为 2017、7、10，两个对象的数据初始值相同。最后一个对象 tomorrow 通过调用函数 Set()将日期修改为 2017.7.11。

**说明：**

(1) 构造函数与普通成员函数一样，可以定义在类体内，也可以定义在类体外。如例 3-9 中 Date 类的定义。

```
class Date
{
public:
    Date();                              //构造函数的声明
    void Set(int y, int m, int d);       //带参成员函数的声明
    ⋮
    void Print();
private:
    int year,month,day;
};
Date::Date()                             //构造函数的定义
{ year =2017; month =7;day =10; }
```

(2) 在实际应用中，一般都要为类定义构造函数，如果没有定义，编译系统就自动生成一个默认的构造函数，这个默认构造函数不带任何参数，只负责创建对象，不负责初始化。如果程序员定义了构造函数则系统不再生成。系统自动生成的构造函数的形式为：

```
类名::构造函数名(){ }
```

(3) 构造函数是类的成员函数，具有一般成员函数的所有性质，可访问类的所有成员，可以是内联函数，可带有参数表，可带有默认的形参值，还可以重载。

(4) 当创建新的对象时，该对象所属的类的构造函数自动被调用。

例 3-9 中 Date 类定义的是不带参数构造函数，定义对象的一般形式为：

```
类名 对象名;
```

在一个对象生存期中构造函数只被调用一次。

**注意：** 例 3-9 中不带参数构造函数对对象的初始化是固定的，即每个对象的初始值相同，如希望在建立对象时赋予不同的初始值，则需要定义带参数的构造函数。

### 2. 带参数构造函数与成员初始化列表

构造函数可以带有参数，通过参数为数据成员赋初值。所以，对于带参数的构造函数，应考虑为构造函数传递实参。当建立对象时给出实参值，在自动调用构造函数时将实参传递给构造函数。此时，定义对象的一般形式为：

```
类名 对象名(实参表);
```

**【例 3-10】** 使用带参数构造函数实现例 3-9。

```
//D:\QT_example\3\ Example3_10.cpp
#include <iostream>
using namespace std;
```