目 录

[第一章 认识C++的对象 3](#_Toc338237663)

[1.1 初识C++的函数和对象 3](#_Toc338237664)

[1.2 认识C++语言面向过程编程的特点 4](#_Toc338237665)

[1.3 程序的编辑、编译和运行的基本概念 5](#_Toc338237666)

[第二章 从结构到类的演变 6](#_Toc338237667)

[2.1 结构的演化 6](#_Toc338237668)

[2.2 从结构演变一个简单的类 6](#_Toc338237669)

[2.3 面向过程与面向对象 6](#_Toc338237670)

[2.4 C++面向对象程序设计的特点 6](#_Toc338237671)

[2.5 使用类和对象 7](#_Toc338237672)

[2.6 string对象数组与泛型算法 7](#_Toc338237673)

[第3章 函数和函数模板 7](#_Toc338237674)

[3.1 函数的参数及其传递方式 7](#_Toc338237675)

[3.2 深入讨论函数返回值 8](#_Toc338237676)

[3.3 内联函数 9](#_Toc338237677)

[3.4 函数重载和默认参数 9](#_Toc338237678)

[3.5 函数模板 9](#_Toc338237679)

[第4章 类和对象 10](#_Toc338237680)

[4.1 类及其实例化 10](#_Toc338237681)

[4.2 构造函数 11](#_Toc338237682)

[4.3 析构函数 12](#_Toc338237683)

[4.4 调用复制构造函数的综合实例 13](#_Toc338237684)

[4.5 成员函数重载及默认参数 13](#_Toc338237685)

[4.6 this指针 13](#_Toc338237686)

[4.7 一个类的对象作为另一个类的成员 13](#_Toc338237687)

[4.8 类和对象的性质 13](#_Toc338237688)

[4.9 面向对象的标记图 14](#_Toc338237689)

[4.10 面向对象编程的文件规范 15](#_Toc338237690)

[第五章 特殊函数和成员 16](#_Toc338237691)

[5.1 对象成员的初始化 16](#_Toc338237692)

[5.2 静态成员 17](#_Toc338237693)

[5.3 友元函数 17](#_Toc338237694)

[5.4 const对象 18](#_Toc338237695)

[5.5 数组和类 19](#_Toc338237696)

[5.6 指向类成员函数的指针 19](#_Toc338237697)

[5.7 求解一元二次方程 20](#_Toc338237698)

[第六章 继承和派生 20](#_Toc338237699)

[6.1 继承和派生的基本概念 20](#_Toc338237700)

[6.2 单一继承 20](#_Toc338237701)

[6.3 多重继承 22](#_Toc338237702)

[6.4 二义性及其支配规则 22](#_Toc338237703)

[第七章 类模板与向量 22](#_Toc338237704)

[7.1 类模板 22](#_Toc338237705)

[7.2 向量与泛型算法 23](#_Toc338237706)

[7.3 出圈游戏 25](#_Toc338237707)

[第八章 多态性和虚函数 25](#_Toc338237708)

[8.1 多态性 25](#_Toc338237709)

[8.2 虚函数 25](#_Toc338237710)

[8.3 多重继承与虚函数 27](#_Toc338237711)

[8.4 类成员函数的指针与多态性 27](#_Toc338237712)

[第9章 运算符重载及流类库 27](#_Toc338237713)

[9.1 运算符重载 27](#_Toc338237714)

[9.2 流类库 28](#_Toc338237715)

[9.3 文件流 31](#_Toc338237716)

[第10章 面向对象设计实例 32](#_Toc338237717)

[10.1 过程抽象和数据抽象 32](#_Toc338237718)

[10.2 发现对象并建立对象层 33](#_Toc338237719)

[10.3 定义数据成员和成员函数 33](#_Toc338237720)

[10.4 如何发现基类和派生类结构 34](#_Toc338237721)

[10.5 接口继承与实现继承 34](#_Toc338237722)

[10.6 设计实例 35](#_Toc338237723)

# 第一章 认识C++的对象

## 1.1 初识C++的函数和对象

一般称现实世界中客观存在的事物为对象。

1.混合型语言

C++程序以.cpp作为文件扩展名，并且必须有一个且只能有一个名为mian（不是C++的关键字）的主函数。真正的面向对象的语言没有主函数，C++保留了这个面向过程的主函数，所以称之为混合型语言。

2.灵活的注释方式

3.使用输出和输入对象

C++将数据从一个对象流向另一个对象的流动的抽象称为“流”。从流中获取数据的操作称为提取操作。向流中添加数据的操作称为插入操作。

cin用来处理标准输入，即键盘输入。cout用来处理标准输出，即屏幕输出。

4.使用命名空间

所谓命名空间（namespace）是一种将程序库名称封装起来的方法，它提高了程序的性能和可靠性。

C++新标准就是将标准类库中的头文件与一般的头文件（需要使用扩展名“.h”）区分开来。当然，也可以自己定义符合标准库的头文件，使用这种头文件时，也需要同时使用命名空间语句。

如果仍然使用C库中的头文件，则需要使用扩展名“.h”形式，例如<math.h>和<stdio.h>。如果使用C++提供的头文件，则不需要使用扩展名“.h”，例如，<string>。注意C++库中替代C库中的头文件的正确名称，例如可以使用<cmath>替代<math.h>。

5.对象的定义及初始化

定义对象包括为它命名并赋予它数据类型。

6.函数原型及其返回值

函数都需要有类型说明。int main() 指出main是整数类型，返回值由return后面的表达式决定，且表达式的值必须与声明函数的类型一致。

C++使用变量的基本规则是：必须先声明，后使用，对函数调用也是如此。

7.const修饰符和预处理程序

C语言一般使用"#define"定义常量，在C++中，建议使用const代替宏定义。const是放在语句定义之前的，因此可以进行类型判别。

用关键字const修饰的标识符是一类特殊的常量，称为符号常量，或const变量。使用const允许编译器对其进行类型检查并可改善程序的可读性。

C++语言可以使用宏定义。无参数的宏作为常量，而参数的宏则可以提供比函数更高的效率。但预处理只是进行简单的文本代替而不进行语法检查，所以会存在一些问题。

因为被const修饰的变量的值在程序中不能被改变，所以在声明符号常量是，必须对符号常量进行初始化，除非这个变量使用extern修饰的外部变量。

C++语言预处理程序不是C++编译程序的一部分，它负责分析处理几种特殊的语句，这些语句被称为预处理语句。顾名思义，预处理程序对这几种特殊语句的分析处理是在编译程序的其他部分之前进行的。为了与一般的C++程序语句相区别，所有预处理语句都以位于行首的符号“#”开始。

预处理语句有3种，分别是宏定义、文件包含和条件编译。

预处理程序把所有出现的、被定义的名字全部替换成对应的“字符序列”。#define中的名字与C++中的标识符有相同的形式，为了区别，往往用大写字母来表示（标识符用小写字母），这也适合const语句。

文件引用使用双引号还是尖括号，其含义并不一样。采用尖括号引用系统提供的包含文件，C++编译系统将首先在C++系统设定的目录中寻找包含文件，如果没有找到，就到指定的目录中去寻找。采用双引号引用自己定义的包含文件（一般都放在自己定义的指定目录中），这将通知C++编译器在用户当前的目录下或指定的目录下寻找包含文件。指定的目录不必在同一个逻辑盘中。

8.程序运行结果

9.程序书写格式

## 1.2 认识C++语言面向过程编程的特点

C++语言的标准模板库（Standard Templete Library,STL）提供了与要操作的元素类型无关的算法，不仅使许多复杂问题迎刃而解，而且也将许多面向对象的程序设计问题转化成基于对象的面向过程编程。

1.2.1 使用函数重载

C++允许为一个函数定义几个版本，从而使一个函数名具有多种功能，这称为函数重载。

1.2.2 新的基本数据类型及其注意事项

void是无类型标识符，只能声明函数的返回值类型，不能声明变量。C++语言还比C语言多了bool（布尔）型。C++标准只限定int和short至少要有16位，而long至少32位，short不得长于int，int不得长于long。

地址运算符“&”用来取对象存储的首地址。

C++语言中的整数常量有4种类型：十进制常量、长整型常量、八进制常量和十六进制常量，并用前缀和后缀进行分类标识。

1.2.3 动态分配内存

在使用指针时，如果不使用对象地址初始化指针，可以自己给它分配地址。对于值存储一个基本类型数据的指针，申请的方式如下：

new 类型名[size] //申请可以存储size个该数据类型的对象

不再使用时，简单地使用“delete指针名”即可释放已经申请的存储空间。

1.2.4 引用

别名的地址就是原来对象的地址，选定命名时使用“引用”运算符“&”，再选用数据类型与之配合。引用的声明方式如下：

数据类型& 别名=对象名；

所谓“引用”就是将一个新标识符和一块已经存在的存储区域相关联。因此，使用引用时没有分配新的存储区域，它本身并不是新的数据类型。

引用通常用于函数的参数表中或者作为函数的返回值。对引用实质性的理解应抓住如下两点：

（1）引用实际上就是变量的别名，使用引用就如同直接使用变量一样。引用与变量名在使用的形式上是完全一样的，引用只是作为一种标识对象的手段，不能直接声明对数组的引用，也不能声明引用的引用。

（2）引用的作用与指针有相似之处，它会对内存地址上存在的变量进行修改，但它不占用新的地址，从而节省开销。

1.2.5 对指针使用const限定符

可以用const限定符强制改变访问权限。

1.左值和右值

左值是指某个对象的表达式。

2.指向常量的指针

指向常量的指针是在非常量指针声明前面使用const，例如：

const int \*p;

它告诉编译器，“\*p”是常量，不能将“\*p”作为左值进行操作，即限定了“\*p=”的操作，所以称为指向常量的指针。

3.常量指针

把const限定符放在\*号的右边，是使指针本身称为一个const指针。

int x=5;

int \* const p=&x；

不能改变p的指向，但可以通过间接引用运算符“\*”改变其值，例如语句“\*p=56;”将上面的x的值改变为56。

4.指向常量的常量指针

也可以声明指针和指向的对象都不能改动的“指向常量的常量指针”，这时必须要初始化指针。例如：

int x=2;

const int \* const p=&x;

告诉编译时，\*p和p都是常量，都不能作为左值。

1.2.6 泛型算法应用于普通数组

所谓泛型算法，就是提供的操作与元素的类型无关。

1.2.7 数据的简单输入输出格式

C++提供了两种格式控制方式：一种是使用ios \_base类提供的接口；另一种是使用一种称为操控符的特殊函数，它的特点是可直接包含在输出和输入的表达式中，因此更为方便，不带形式参数的操控符定义在头文件<iostream>中，带形式参数的操控符定义在头文件<iomanip>中。使用它们时，一是要正确包含它们，二是只有与符号“<<”或“>>”连接时才起作用，三是无参数的操控符函数不能带有“()”号。

## 1.3 程序的编辑、编译和运行的基本概念

用C++语言写成的程序称为源程序，源程序必须经过C++编译程序翻译成机器语言才能执行。要得到一个用C++语言设计的、名为myapp.exe的可执行文件，其过程可分为如下几步：

（1）先使用编辑器编辑一个C++程序mycpp.cpp，又称其为C++的源程序。

（2）然后使用C++编译器对这个C++程序进行编译，产生文件mycpp.obj

（3）再使用连接程序（又称Link），将mycpp.obj变成mycpp.exe

所谓集成环境，就是将C++语言编辑、编译、连接和运行程序都集成到一个综合环境中。

# 第二章 从结构到类的演变

## 2.1 结构的演化

类是从结构演变而来，开始称为“带类的C”。这种演变就是从让结构含有函数开始的。

2.1.1 结构发生质的演变

1.函数与数据共享

2.封装性

2.1.2 使用构造函数初始化结构对象

## 2.2 从结构演变一个简单的类

## 2.3 面向过程与面向对象

所谓“面向过程”，就是不必了解计算机的内部逻辑，而把精力主要集中在对如何求解问题的算法和过程的描述上，通过编写程序把解决问题的步骤告诉计算机。

所谓函数，就是模块的基本单位，是对处理问题的一种抽象。

结构化程序设计使用的是功能抽象，面向对象程序设计不仅能进行功能抽象，而且能进行数据抽象。“对象”实际上是功能抽象和数据抽象的统一。

面向对象的程序设计方法不是以函数过程和数据结构为中心，而是以对象代表求解问题的中心环节。他最求的是现实问题空间与软件系统解空间的近似和直接模拟。

软件开发是对给定问题求解的过程。从认识论的角度看，可以归为两项主要的活动：认识与描述。

软件开发者将被开发的整个业务范围称作“问题域”（problem domain），“认识”就是在所要处理的问题域范围内，通过人的思维，对该问题域客观存在的事物以及对所要解决的问题产生正确的认识和理解，包括弄清事物的属性，行为及其彼此之间的关系并找出解决问题的方法。

“描述”是指用一种语言把人们对问题域中事物的认识、对问题及其解决方法的认识描述出来。最终的描述必须使用一种能够被机器读得懂的语言，即编程语言。

## 2.4 C++面向对象程序设计的特点

和传统的程序设计方法相比，面向对象的程序设计具有抽象、封装、继承和多态性等关键要素。

2.4.1 对象

C++可使用对象名、属性和操作三要素来描述对象。

2.4.2 抽象和类

抽象是一种从一般的观点看待事物的方法，即集中于事物的本质特征，而不是具体细节或具体实现。

类的概念来自于人们认识自然、认识社会的过程。在这一过程中，人们主要使用由特殊到一般的归纳法和由一般到特殊的演绎法。在归纳的过程中，是从一个个具体的事物中把共同的特征抽取出来，形成一个一般的概念，这就是“归类”；在演绎的过程中，把同类事物，根据不同的特征分成不同的小类，这就是“分类”。对于一个具体的类，它有许多具体的个体，这些个体叫做“对象”。

类的作用是定义对象。

所谓“一个类的所有对象具有相同的属性”，是指属性的个数、名称、数据类型相同，各个对象的属性值则可以互不相同，并且随着程序的执行而变化。

2.4.3 封装

将类封装起来，也是为了保护类的安全。所谓安全，就是限制使用类的属性和操作。

对象内部数据结构这种不可访问性称为信息（数据）隐藏。

封装就是把对象的属性和操作结合成一个独立的系统单位，并尽可能隐蔽对象的内部细节。

在类中，封装是通过存取权限实现的。

2.4.4 继承

继承是一个类可以获得另一个类的特性的机制，继承支持层次概念。

通过继承，低层的类只须定义特定于它的特征，而共享高层的类中的特征。

2.4.5 多态性

不同的对象可以调用相同名称的函数，但可导致完全不同的行为的现象称为多态性。

## 2.5 使用类和对象

2.5.1 使用string对象

所谓方法，书就是供这类对象使用的成员函数。对象使用自己的成员函数的方法是通过“.”运算符，格式如下：

对象名.成员函数

2.5.2 使用string类的典型成员函数实例

2.5.3 使用complex对象

C++标准程序库提供complex类定义复数对象。

2.5.4 使用对象小结

注意：类是抽象出一类物质的共同特征，模板则是归纳出不同类型事物的共同操作。

## 2.6 string对象数组与泛型算法

# 第3章 函数和函数模板

## 3.1 函数的参数及其传递方式

C语言函数参数的传递方式只有传值一种，传值又分为传变量值和传变量地址值两种情况。比较复杂的是结构变量，结构变量的值是指结构域中所有的变量值。在C++中，像int、float、double、char、bool等简单数据类型的变量，也是对象。类对象一般都包括数据成员和成员函数，如果在C++沿用C语言的说法，则对象的值就是对象所有数据成员的值，约定参数传递中传递“对象值”是指对象的数据成员值，传递“对象地址值”是指对象的首地址值。C++函数还可以使用传递对象的“引用”方式，即它的函数参数有两种传递方式：传值和传引用。传引用其实就是传对象的地址，所有也称传地址方式。

参数传递中不要混淆传地址值和传地址的区别。传地址值传递的是值，不是地址；传地址传的是地址不是地址值。传递对象地址值是使用对象指针做为参数；传递地址时使用对象引用作为参数。

3.1.1 对象作为函数参数

将对象作为函数参数，是将实参对象的值传递给形参对象，这种传递是单向的。形参拥有实参的备份，当在函数中改变形参的值时，改变的是这个备份中的值，不会影响原来实参的值。

3.1.2 对象指针作为函数参数

将指向对象的指针作为函数参数，形参是对象指针（指针可以指向对象的地址），实参是对象的地址值。

注意：不要非要在主程序里产生指针，然后再用指针作为参数。函数原型参数的类型是指针，可以指针让它指向对象地址，即：

string \*s1=&str1;

是完全正确的，使用&str1标识的是取对象str1的地址值，所以&str1直接作为参数即可。

3.1.3 引用作为函数参数

可以使用“引用”作为函数的参数（引用形参）。这时函数并没有对形参对象初始化，即没有指定形参对象是哪个对象的别名。在函数调用是，实参对象名传给形参对象名，形参对象名就成为实参对象名的别名。实参对象和形参对象代表同一个对象，所以改变形参对象的值就是改变实参对象的值。

实际上，在虚实结合时是把实参对象的地址传给形参对象，使形参对象的地址取实参对象的地址，从而使形参对象和实参对象共享同一个单元。这就是地址传递方式。

通过使用引用参数，一个函数可以修改另外一个函数内的变量。因为引用对象不是一个独立的对象，不单独占用内存单元，而对象指针要另外开辟内存单元（其内容是地址），所以传引用比传指针更好。

注意：虽然系统向形参传递的是实参的地址而不是实参的值，但实参必须使用对象名。

3.14 默认参数

默认参数就是不要求程序员设定该参数，而由编译器在需要时给该参数赋默认值。当程序员需要传递特殊值时，必须显式地指明。默认参数是在函数原型中说明的，默认参数可以多于1个，但必须放在参数序列的后部。

如果一个默认参数需要指明一个特定值，则在其之前所有的参数都必须赋值。

3.1.5 使用const保护数据

用const修饰传递参数，意思是通知函数，它只能使用参数而无权修改它。这主要是为了提高系统的自身安全。C++中普遍采用这种方法。

## 3.2 深入讨论函数返回值

C++函数的返回值类型可以是除数组以外的任何类型。非void类型的函数必须向调用者返回一个值。数组只能返回地址。当函数返回值是指针或引用对象时，需要特别注意：函数返回所指的对象必须继续存在，因此不能将函数内部的局部对象作为函数的返回值。

3.2.1 返回引用的函数

函数可以返回一个引用，将函数说明为返回一个引用的主要目的是为了将该函数用在赋值运算符的左边。函数原型的表示方法如下：

数据类型 &函数名（参数列表）;

3.2.2 返回指针的函数

函数的返回值可以是存储某种类型数据的内存地址，称这种函数为指针函数。它们的一般定义形式如下：

类型标识符 \*函数名（参数列表）；

在C++中，除了内存分配失败之外，new不会返回空指针，并且没有任何对象的地址为零。指针所指向的对象的生存期不应低于该指针的生存期。

3.2.3 返回对象的函数

3.2.4 函数返回值作为函数的参数

如果用函数返回值作为另一个函数的参数，这个返回值必须与参数类型一致。

## 3.3 内联函数

使用关键字inline说明的函数称内联函数。在C++中，除具有循环语句、switch语句的函数不能说明为内联函数外，其他函数都可以说明为内联函数。使用内联函数能加快程序执行速度，但如果函数体语句多，则会增加程序代码的大小。使用小的内联函数在代码速度和大小上可以取得折衷，其他情况下取决于程序员是追求代码速度，还是追求代码的规模。

由于编译器必须知道内联函数的函数体，才能进行内联替换，因此，内联函数必须在程序中第一次调用此函数的语句出现之前定义。

## 3.4 函数重载和默认参数

函数重载可以使一个函数名具有多种功能，即具有“多种形态”，这种特性称为多态性。

C++的多态性又被直观地称为“一个名字，多个函数”。源代码只指明函数调用，而不说明具体调用哪个函数。编译器的这种连接方式称为动态联编或迟后联编。在动态联编中，直到程序运行才能确定调用哪个函数（动态联编需要虚函数的支持）。如果编译器在编译时，能根据源代码调用固定的函数标识符，并用物理地址代替它们，这就称为静态联编或先期联编。静态联编是在程序被编译时进行的。

使用默认参数，就不能对少于参数个数的函数进行重载。另外，仅有函数返回值不同也是区分不了重载函数的。当使用默认参数设计类的构造函数时，要特别注意这一问题。

## 3.5 函数模板

1.引入函数模版

由于函数在设计时没有使用实际的类型，而是使用虚拟的类型参数，故其灵活性得到加强。当用实际的类型来实例化这种函数时，就好像按照模版来制造新的函数一样，所以称这种函数为函数模板。将函数模版与某个具体数据类型连用，就产生了模板函数，又称这个过程为函数模板实例化，这种形式就是类型参数化。

2.函数模板的参数

对于一个默认调用，能从函数参数推断出模板参数的能力是其中最关键的一环。要想省去显式调用的麻烦，条件是由这个调用的函数参数表能够惟一地标识出模板参数的一个集合。

3.使用显式规则和关键字typename

C++专门定义一个仅仅用在模板中的关键字typename，它的用途之一是代替template参数列表中的关键字class。

# 第4章 类和对象

## 4.1 类及其实例化

对象就是一类物体的实例，将一组对象的共同特征抽象出来，从而形成“类”的概念。

4.1.1 定义类

像C语言构造结构一样，类也是一种用户自己构造的数据类型并遵循C++的规定。

类要先声明后使用，不管声明内容是否相同，声明同一个名字的两个类是错误的，类是具有惟一标识符的实体；在类中声明的任何成员不能使用extern、auto和register关键字进行修饰；类中声明的变量属于该类，在某些情况下，变量也可以被该类的不同实例所共享。

类和其他数据类型不同的是，组成这种类型的不仅可以有数据，而且可以有对数据进行操作的函数，他们分别叫做类的数据成员和类的成员函数，而且不能在类声明中对数据成员使用表达式进行初始化。

1.类声明

类声明以关键字class开始，其后跟类名。类所声明的内容用花括号括起来，右花括号后的分号作为类关键字声明语句的结束标志。这一对花括号之间的内容称为类体；

访问权限用于控制对象的某个成员在程序中的可访问性，如果没有使用关键字，则所有成员默认声明为private权限。

2.定义成员函数

类中声明的成员函数用来对数据成员进行操作，还必须在程序中实现这些成员函数。

定义成员函数的一般形式如下：

返回类型 类名：：成员函数名（参数列表）

{

成员函数的函数体//内部实现

}

其中“：：”是作用域运算符，“类名”是成员函数所属类的名字，“：：”用于表名其后的成员函数是属于这个特定的类。换言之，“类名：：成员函数名”的意思就是对属于“类名”的成员函数进行定义，而“返回类型”则是这个成员函数返回值的类型。

也可以使用关键字inline将成员函数定义为内联函数。

如果在声明类的同时，在类体内给出成员函数的定义，则默认为内联函数。

3.数据成员的赋值

不能在类体内给数据成员赋值。在类体外就更不允许了。

数据成员的具体值是用来描述对象的属性的。只有产生了一个具体的对象，这些数据值才有意义。如果在产生对象时就使对象的数据成员具有指定值，则称为对象的初始化。

4.1.2 使用类的对象

对象和引用都使用运算符“.”访问对象的成员，指针则使用“- >”运算符。

暂不涉及还没有介绍的保护成员，可以归纳出如下规律：

（1）类的成员函数可以直接使用自己类的私有成员（数据成员和成员函数）

（2）类外面的函数不能直接访问类的私有成员（数据成员和成员函数）

（3）类外面的函数只能通过类的对象使用该类的公有成员函数。

在程序运行时，通过为对象分配内存来创建对象。在创建对象时，使用类作为样板，故称对象为类的实例。

定义类对象指针的语法如下：

类名\* 对象指针名；

对象指针名=对象的地址；

也可以直接进行初始化。

类名\* 对象指针名=对象的地址；

类对象的指针可以通过“->”运算符访问对象的成员，即：

对象指针名->对象成员名

4.1.3 数据封装

面向对象的程序设计是通过为数据和代码建立分块的内存区域，以便提供对程序进行模块化的一种程序设计方法，这些模块可以被用做样板，在需要时在建立其副本。根据这个定义，对象是计算机内存中的一块区域，通过将内存分块，每个模块（即对象）在功能上保持相对独立。

对象被视为能做出动作的实体，对象使用这些动作完成相互之间的作用。换句话说，对象就像在宿主计算机上拥有数据和代码并能相互通信的具有特定功能的一台较小的计算机。

## 4.2 构造函数

C++有称为构造函数的特殊成员函数，它可自动进行对象的初始化。

初始化和赋值是不同的操作，当C++语言预先定义的初始化和赋值操作不能满足程序的要求时，程序员可以定义自己的初始化和赋值操作。

4.2.1 默认构造函数

当没有为一个类定义任何构造函数的情况下，C++编译器总要自动建立一个不带参数的构造函数。

4.2.2 定义构造函数

1.构造函数的定义和使用方法

构造函数的名字应与类名同名。并在定义构造函数时不能指定返回类型，即使void类型也不可以。

当声明一个外部对象时，外部对象只是引用在其他地方声明的对象，程序并不为外部对象说明调用构造函数。如果是全局对象，在main函数执行之前要调用它们的构造函数。

2.自动调用构造函数

程序员不能在程序中显式地调用构造函数，构造函数是自动调用的。例如构造一个Point类的对象a，不能写成“Point a.Point(x,y) ；”，只能写成“Point a(x,y) ；”。编译系统会自动调用Point(x,y)产生对象a并使用x和y将其正确地初始化。

可以设计多个构造函数，编译系统根据对象产生的方法调用相应的构造函数，构造函数是在产生对象的同时初始化对象的。

4.2.3 构造函数和预算符new

运算符new用于建立生存期可控的对象，new返回这个对象的指针。由于类名被视为一个类型名，因此，使用new建立动态对象的语法和建立动态变量的语法类似，其不同点是new和构造函数一起使用。

使用new建立的动态对象只能用delete删除，以便释放所占空间。应养成及时释放不再使用的内存空间的习惯。

4.2.4 构造函数的默认参数

如果程序定义自己的有参数构造函数，又想使用无参数形式的构造函数，解决的方法时间相应的构造函数全部使用默认的参数设计。

4.2.5 复制构造函数

引用在类中一个很重要的用途是用在复制构造函数中。一是一类特殊而且重要的函数，通常用于使用已有的对象来建立一个新对象。

在通常情况下，编译器建立一个默认复制构造函数，默认复制构造函数采用拷贝方式使用已有的对象来建立新对象，所以又直译为拷贝构造函数。程序员可以自己定义复制构造函数，对类A而言，复制构造函数的原型如下：

A::A(A&)

从这个函数原型来看，首先它是一个构造函数，因为这毕竟是在创造一个新对象。其次，他的参数有些特别，是引用类自己的对象，即用一个已有的对象来建立新对象。使用引用是从程序的执行效率角度考虑的。为了不改变原有对象，更普通的形式是像下面这样使用const限定：

A::A(const A &)

像调用构造函数一样，如果自定义了复制构造函数，编译器只调用程序员为它设计的赋值构造函数。

在C++中，在一个类中定义的成员函数可以访问该类任何对象的私有成员。

## 4.3 析构函数

在对象消失时，应使用析构函数释放由构造函数分配的内存。构造函数、赋值构造函数和析构函数是构造型成员函数的基本成员，应深刻理解他们的作用并熟练掌握其设计方法。

4.3.1 定义析构函数

因为调用析构函数也是由编译器来完成的，所以编译器必须总能知道应调用哪个函数。最容易、也最符合逻辑的方法是指定这个函数的名称与类名一样。为了与析构函数区分，在析构函数的前面加上一个“~”号（仍然称析构函数与类同名）。在定义析构函数时，不能指定任何返回类型，即使指定void类型返回类型也不行。析构函数也不能指定参数，但是可以显示地说明参数为void，即形如A：：~A（void）。从函数重载的角度分析，一个类也只能定义一个析构函数且不能指明参数，以便编译系统自动调用。

析构函数在对象的生存期结束时被自动调用。当对象的生存期结束时，程序为这个对象调用析构函数，然后回收这个对象占用的内存。全局对象和静态对象的析构函数在程序运行结束之前调用。

类的对象数组的每个元素调用一次析构函数。全局对象数组的析构函数在程序结束之前被调用。

4.3.2 析构函数与运算符delete

运算符delete与析构函数一起工作。当使用运算符delete删除一个动态对象时，他首先为这个动态对象调用析构函数，然后再释放这个动态对象占用的内存，这和使用new建立动态对象的过程正好相反。

当使用delete释放动态对象数组时，必须告诉这个动态对象数组有几个元素对象，C++使用“[ ]”来实现。即语句

delete[ ] ptr ; //注意不要写错为delete ptr[]

当程序先后创建几个对象时，系统按后建先析构的原则析构对象。当使用delete调用析构函数时，则按delete的顺序析构。

4.3.3 默认析构函数

如果在定义类时没有定义析构函数，C++编译器也为它产生一个函数体为空的默认析构函数。

## 4.4 调用复制构造函数的综合实例

## 4.5 成员函数重载及默认参数

## 4.6 this指针

使用this指针，保证了每个对象可以拥有自己的数据成员，但处理这些数据成员的代码可以被所有的对象共享。

C++规定，当一个成员函数被调用是，系统自动向它传递一个隐含的参数，该参数是一个指向调用该函数的对象的指针，从而使成员函数知道该对哪个对象进行操作。在程序中，可以使用关键字this来引用该指针。this指针是C++实现封装的一种机制，它将对象和该对象调用的成员函数连接在一起，在外部看来，每一个对象都拥有自己的成员函数。

除非有特殊需要，一般情况下都省略符号“this ->”，而让系统进行默认设置。

## 4.7 一个类的对象作为另一个类的成员

## 4.8 类和对象的性质

4.8.1 对象的性质

（1）同一个类的对象之间可以相互赋值。

（2）可使用对象数组。

（3）可使用指向对象的指针，使用取地址运算符&将一个对象的地址置于该指针中。

注意，指向对象的指针的算术运算规则与C语言的一样，但指向对象的指针不能取数据成员的地址，也不能去成员函数的地址。

（4）对象可以用作函数参数。

（5）对象作为函数参数时，可以使用对象、对象引用和对象指针。

（6）一个对象可以做为另一个类的成员。

4.8.2 类的性质

1.使用类的权限

为了简单具体，讨论数据成员为私有，成员函数为公有的情况。

（1）类本身的成员函数可以使用类的所有成员（私有和公有成员）。

（2）类的对象只能访问公有成员函数。

（3）其他函数不能使用类的私有成员，也不能使用公有成员函数，它们只能通过类的对象使用类的公有成员函数。

（4）虽然一个可以包含另外一个类的对象，但这个类也只能通过被包含类的对象使用那个类的成员函数，通过成员函数使用数据成员。

2.不完全的类声明

类不是内存中的物理实体，只有当使用类产生对象时，才进行内存分配，这种对象建立的过程称为实例化。

应当注意的是：类必须在其成员使用之前先进行声明。

class MembersOnly; //不完全的类声明

MenbersOnly \*club; //定义一个全局变量类指针

第一条语句称为不完全类声明，它用于在类没有完全定义之前就引用该类的情况。

不完全声明的类不能实例化，否则会出现编译错误；不完全声明仅用于类和结构，企图存取没有完全声明的类成员，也会引起编译错误。

3.空类

尽管类的目的是封装代码和数据，它也可以不包括任何声明。

class Empty{};

4.类作用域

声明类时所使用的一对花括号形成所谓的类的作用域。在类作用域中声明的标识符只在类中可见。

如果该成员函数的实现是在类定义之外给出的，则类作用域也包含类中成员函数的作用域。

类中的一个成员名可以使用类名和作用域运算符来显式地指定，这称为成员名限定。

## 4.9 面向对象的标记图

4.9.1 类和对象的UML标记图

4.9.2 对象的结构与连接

只有定义和描述了对象之间的关系，各个对象才能构成一个整体的、有机的系统模型，这就是对象的结构和连结关系。对象结构是指对象之间的分类（继承）关系和组成（聚合）关系，统称为关联关系。对象之间的静态关系是通过对象属性之间的连接反映的，称为实例连接。对象行为之间的动态关系是通过对象行为（信息）之间的依赖关系表现的，称之为消息连接，实例连接和消息连接统称为连接。

1.分类关系及其表示

C++中的分类结构是继承（基类/派生类）结构，UML使用一个空三角形表示继承关系，三角形指向基类。

2.对象组成关系及其表示

组成关系说明的结构是整体与部分关系。C++中最简单的是包含关系。C++语言中的“聚合”隐含了两种实现方式，第一种方式是独立地定义，可以属于多个整体对象，并具有不同生存期。这种所属关系是可以动态变化的，称之为聚集。使用空心菱形表示它们之间的关系。第二种方式是用一个类的对象作为一种广义的数据类型来定义整体对象的一个属性，构成一个嵌套对象。在这种情况下，这个类的对象只能隶属于惟一的整体对象并与它同生同灭，称这种情况为“组合”，它们之间的关联关系比第一种强，具有管理组成部分的责任，使用实心菱形表示。

3.实例连接及其表示

实例连接反映对象之间的静态关系，例如车和驾驶员的关系，这种双边关系在实现中可以通过对象（实例）的属性表达出来。实例连接有一对一、一对多、多对多3种连接方式。

4.消息连接及其表示

消息连接描述对象之间的动态关系。即若一个对象在执行自己的操作时，需要通过消息请求另一个对象为它完成某种服务，则说第一个对象与第二个对象之间存在着消息连接。消息连接是有方向的，使用一条带箭头的实线表示，从消息的发送者指向消息的接收者。

4.9.3 使用实例

4.9.4 对象、类和消息

对象的属性是指描述对象的数据成员。数据成员可以是系统或程序员定义的数据类型。对象属性的集合称为对象的状态。

对象的行为是定义在对象属性上的一组操作的集合。操作（函数成员）是响应消息而完成的算法，表示对象内部实现的细节。对象的操作集合体现了对象的行为能力。

对象的属性和行为是对象定义的组成要素，分别代表了对象的静态和动态特征。由以上分析的例子可见，无论对象是简单的或是负责的，一般具有以下特征：

（1）有一个状态，由与其相关的属性集合所表征。

（2）有惟一标识名，可以区别于其他对象。

（3）有一组操作方法，每个操作决定对象的一种行为。

（4）对象的状态只能被自己的行为所改变。

（5）对象的操作包括自身操作（施加于自身）和施加于其他对象的操作。

（6）对象之间以消息传递的方式进行通信。

（7）一个对象的成员仍可以是一个对象。

## 4.10 面向对象编程的文件规范

一般要求将类的声明放在头文件中，非常简单的成员函数可以在声明中定义（默认内联函数形式），实现放在.cpp文件中。在.cpp文件中，将头文件包含进去。主程序单独使用一个文件，这就是多文件编程规范。

4.10.1 编译指令

C++的源程序可包含各种编译指令，以指示编译器对源代码进行编译之前先对其进行预处理。所有的编译指令都以#开始，每条编译指令单独占用一行，同一行不能有其他编译指令和C++语句（注释例外）。编译指令不是C++的一部分，但扩展了C++编程环境的使用范围，从而改善程序的组织和管理。

1.嵌入指令

嵌入指令#include指示编译器将一个源文件嵌入到带有#include指令的源文件中该指令所在的位置处。尖括号或双引号中的文件名可包含路径信息。例如：

#include<\user\prog.h>

注意：由于编译指令不是C++的一部分，因此，在这里表示反斜杠时只使用一个反斜杠。

2.宏定义

#define指令定义一个标识符及串，在源程序中每次遇到该标识符时，编译器均用定义的串代替之。该标识符称为宏名，而将替换过程称之为宏替换。#define指令用以进行宏定义，其一般形式如下：

#define 宏名 替换正文

“宏名”必须是一个有效的C++标识符，“替换正文”可为任意字符组成的字符序列。“宏名”和“替换正文”之间至少有一个空格。注意，宏定义由新行结束，而不以分号结束。如果给出了分号，则它也被视作为替换正文的一部分。当替换正文要书写在多行上时，除最后一行外，每行的行尾要加上一个反斜线，表示宏定义继续到下一行。

因宏定义有许多不安全因素，对需要使用无参数宏的场合，应该尽量使用const代替宏定义。

在程序的一个地方定义的宏名，如果不想使其影响到程序的其他地方，可以在不再使用时用#undef删除。

3.条件编译指令

条件编译指令是#if、#else、#elif和#endif，它们构成类似于C++的if选择结构，其中#endif表示一条指令结束。

编译指令#if用于控制编译器对源程序的某部分有选择地进行编译。该部分从#if开始，到#endif结束。如果#if后的常量表达式为真，则编译这部分，否则就不编译该部分，这时，这部分代码相当于被从源文件中删除。

编译指令#else在#if测试失效的情况下建立另外一种选择。可以在#else分支中使用编译指令#error输出出错信息。#error使用的形式如下：

#error 出错信息

“出错信息”是一个字符序列。当遇到#error指令时，编译器显示其后面的“出错信息”，并中止对程序的编译。

编译指令可嵌套，嵌套规则和编译器对其的处理方式与C++的if预计嵌套情况类似。

4.define操作符

关键字defined不是指令，而是一个预处理操作符，用于判断一个标识符是否已经被#define定义。如果标识符identifier已被#define定义，则defined（identifier）为真，否则为假。

条件编译指令#ifdef和#ifndef用于测试其后的标识符是否被#define定义，如果已经被定义，则#ifdef测试为真，#ifndef测试为假；如果没有被定义，则#ifdef测试为假，#ifndef测试为真。

4.10.2 在头文件中使用条件编译

# 第五章 特殊函数和成员

## 5.1 对象成员的初始化

可以在一个类中说明具有某个类的类型的数据成员，这些成员成为对象成员。在类A中说明对象成员的一般形式如下：

class A

{

类名1 成员名1;

类名2 成员名2;

……

类名n 成员名n;

};

说明对象成员是在类名之后给出对象成员的名字。为初始化对象成员，A类的构造函数要调用这些对象成员所在类的构造函数，A类的构造函数的定义形式如下：

A::A(参数表0)：成员1（参数表1），成员2（参数表2），成员n（参数表n）

{//其他操作}

冒号“：”后由逗号隔开的项目组成员初始化列表，其中的参数表给出了为调用相应成员所在类的构造函数时应提供的参数。参数列表中的参数都来自“参数表0”，可以使用任意复杂的表达式，其中可以有函数调用。如果初始化列表某项的参数表为空，则列表中相应的项可以省略。

对象成员构造函数的顺序取决于这些对象成员在类中说明的顺序，与他们在成员初始化列表中给出的顺序无关。析构函数的调用顺序与构造函数正好相反。

## 5.2 静态成员

简单成员函数是指声明中不含const、volatile、static关键字的函数。如果类的数据成员或成员函数使用关键字static进行修饰，这样的成员称为静态数据成员或静态成员函数，统称为静态成员。

静态数据成员只能说明一次，如果在类中仅对静态数据成员进行声明，则必须在文件作用域的某个地方进行定义。在进行初始化之前，必须进行成员名限定。

注意：由于static不是函数中的一部分，所以在类声明之外定义静态成员函数时，不使用static。在类中定义的静态成员函数是内联的。

类中的任何成员函数都可以访问静态成员。因为静态成员函数没有this指针，所以静态成员函数只能通过对象名（或指向对象的指针）访问该对象的非静态成员。

静态成员函数与一般函数有如下不同：

（1）可以不指向某个具体的对象，只与类名连用。

（2）在没有建立对象之前，静态成员就已经存在。

（3）静态成员是类的成员，不是对象的成员。

（4）静态成员为该类的所有对象共享，它们被存储于一个公用的内存中。

（5）没有this指针，所以除非显式地把指针传给它们，否则不能存取类的数据成员。

（6）静态成员函数不能说明为虚函数。

（7）静态成员函数不能直接访问非静态函数。

静态对象具有如下性质：

（1）构造函数在代码执行过程中，第一次遇到它的时候变量定义时被调用，但直到整个程序结束之前仅调用一次。

（2）析构函数在整个程序退出之前被调用，同样也只调用一次。

## 5.3 友元函数

有时两个概念上相近的类要求其中一个类可以无限制地存取另一个类的成员。

友元函数解决了这类难题。友元函数可以存取私有成员、公有成员和保护成员。其实，友元函数可以是一个类或函数，尚未定义的类也可以作为友元引用。

1.类本身的友元函数

为本类声明一个友元函数，这时，虽然在类中说明它，但它并不是类的成员函数，所以可以在类外面像普通函数那样定义这个函数。

虽然友元函数是在类中说明的，但函数的定义一般在类之外，作用域的开始点在它的说明点，结束点和类的作用域相同。

友元说明可以出现于类的私有或公有部分。因为友元说明也必须出现于类中，所以应将友元看作类的接口的一部分。使用它的主要目的是提高程序效率。友元函数可以直接访问类对象的私有程序，因而省去调用类成员函数的开销。它的另一个优点是：类的设计者不必在考虑号该类的各种可能使用情况之后再设计这个类，而可以根据需要，通过使用友元增加类的接口。但使用友元的主要问题是：它允许友元函数访问对象的私有成员，这破坏了封装和数据隐藏，导致程序的可维护性变差，因此在使用友元时必须权衡得失。

注意：友元函数可以在类中声明时定义。如果在类外定义，不能再使用friend关键字。

2.将成员函数用作友元

一个类的成员函数（包括构造函数和析构函数）可以通过使用friend说明为另一个类的友元。

3.将一个类说明为另一个类的友元

可以将一个类说明为另一个类的友元。这时，整个类的成员函数均具有友元函数的性能。声明友元关系简化为“friend class 类名；”

需要注意的是，友元关系是不传递的，即当说明类A是类B的友元，类B又是类C的友元时，类A却不是类C的友元。这种友元关系也不具有交换性，即当说明类A是类B的友元时，类B不一定是类A的友元。

当一个类要和另一个类协同工作时，使一个类称为另一个类的友元是很有用的。

友元声明与访问控制无关。友元声明在私有区域进行或在公有区域进行是没有太大区别的。虽然友元函数的声明可以置于任何部分，但通常置于能强调其功能的地方以使其直观。对友元函数声明的惟一限制是该函数必须出现在类声明内的某一部分。

## 5.4 const对象

可以在类中使用const关键字定义数据成员和成员函数或修饰一个对象。一个const对象只能访问const成员函数，否则将产生编译错误。

1.常量成员

常量成员包括常量数据成员、静态常数据成员和常引用。静态常数据成员仍保留静态成员特征，需要在类外初始化。常数据成员和常引用只能通过初始化列表来获得初值。

2.常引用作为函数参数

使用引用作为参数，传送的是地址。但有时仅希望将参数的值提供给函数使用，并不允许函数改变对象的值，这时可以使用常引用作为参数。

3.常对象

在对象名前使用const声明常量对象，但声明时必须同时进行初始化，而且不能被更新。定义的语法如下：

类名 const 对象名（参数表）；

4.常成员函数

可以声明一个成员函数为const函数。一个const对象可以调用const函数，但不能调用非const成员函数。const放在函数声明之前意味着返回值是常量，但这不符合语法。必须将关键字const放在参数列表之后，才能说明该函数是一个const成员函数。声明常成员函数的格式如下：

类型标识符 函数名（参数列表） const;

为了保证不仅声明const成员函数，而且确实也定义为const函数，程序员在定义函数时必须重申const声明。也就是说，const已经成为这种成员函数识别符的一部分，编译器和连接程序都要检查const。定义格式如下：

类型标识符 类名::函数名（参数列表）const{//函数体}

const位于函数参数表之后，函数体之前。也可以在类中用内联函数定义const函数。

类型标识符 函数名（参数列表）const{//函数体}

在定义成员函数时，函数体之前加上const可以防止覆盖函数改变数据成员的值。

在常成员函数里，不能更新对象的数据成员，也不能调用该类中没有用const修饰的成员函数。如果将一个对象说明为常对象，则通过该对象只能调用它的const成员函数，不能调用其他成员函数。

注意：用const声明static成员函数没有什么作用。在C++中声明构造函数和析构函数时使用const关键字均是非法的，但有些C++的编译程序并不给出出错信息。volatile关键字的使用方法与const类似，但因其很少用，此处不介绍。

## 5.5 数组和类

编译器调用适当的构造函数建立数组的每一个分量。如果找不到合适的构造函数，则产生错误信息。

## 5.6 指向类成员函数的指针

对象是一个完整的实体。为了支持这一封装，C++包含了指向类成员的指针。可以用普通指针访问内存中给定类型的任何对象，指向类成员的指针则用来访问某个特定类的对象中给定类型的任何成员。

C++既包含指向类数据成员函数的指针，又包含指向成员函数的指针。他提供一种特殊的指针类型，指针指向类的成员，而不是指向该类的一个对象中该成员的一个实例，这种指针称为指向类成员的指针。不过，指向类数据成员的指针要求数据成员是公有的，用途不大，所以不予介绍，仅简单介绍指向类成员函数的指针。

类并不是对象，但有时可将其视为对象来使用。可以声明并使用指向对象成员函数的指针。指向对象的指针是比较传统的指针，假设类A的成员函数为“void fa(void)；”，如要建立一个指针pafn，它可以指向任何无参数和无返回值的类A的成员函数：

void（A::\* pafn）(void)；

虽然“::”是作用域分辨符，但在这种情况下，“A::”最好读成“A的成员函数”，这时，从内往外看，此声明可读作：pafn是一个指针，指向类A的成员函数，此成员函数既无参数，也无返回值。下面的例子说明了pafn如何被赋值并用以调用函数fa：

pafn=A::fa; //指向类A的成员函数fa的指针pafn

a x; //类A的对象x

A \*px=&x; //指向类A对象x的指针px

(x.\*pafn)(); //调用类A的对象x的成员函数fa

(px->pafn)(); //调用类A的对象x的指针px指向的成员函数fa

指向类A中参数类型列表为list，返回类型为type的成员函数的指针声明形式如下：

type（A::\* pointer）（list）；

如果类A的成员函数fun的原型与pointer所指向的函数的原型一样，则语句

pointer=A::fun;

将该函数的地址（这个地址不是真实地址，而是在A类中所有对象的便宜）置给了pointer。在使用指向类成员函数的指针访问对象的某个成员函数时，必须指定一个对象。用对象名或引用调用pointer所指向的函数时，使用运算符“.\*”，使用指向对象的指针调用pointer所指向的成员函数时，使用运算符“->\*”。

## 5.7 求解一元二次方程

# 第六章 继承和派生

## 6.1 继承和派生的基本概念

这种通过特殊化已有的类来建立新类的过程，叫做“类的派生”，原来的类叫做“基类”，新建立的类则叫做“派生类”。另一方面，从类的成员角度看，派生类自动地将基类的所有成员作为自己的成员，这叫做“继承”。基类和派生类又可以分别叫做“父类”和“子类”，有时也称为“一般类”和“特殊类”。

从一个或多个以前定义的类（基类）产生新类的过程称为派生，这个新类称为派生类。派生的新类同时也可以增加或者重新定义数据和操作，这就产生了类的层次性。

类的继承是指派生类继承基类的数据成员和成员函数。继承常用来表示类属关系，不能将继承理解为构成关系。

当从现有的类中派生出新类时，派生类可以有以下几种变化：

（1）增加新的成员（数据成员或成员函数）

（2）重新定义已有的成员函数

（3）改变基类成员的访问权限

C++派生类使用两种基本的面向对象技术：第一种称为性质约束，即对基类的性质加以限制；第二种称为性质扩展，即增加派生类的性质。

C++中有两种继承：单一继承和多重继承。对于单一继承，派生类只能有一个基类；对于多重继承，派生类可以有多个基类。

## 6.2 单一继承

6.2.1 单一继承的一般形式

在C++，声明单一继承的一般形式如下：

class 派生类名:访问控制 基类名{

private:

成员声明列表

protected:

成员声明列表

public:

成员声明列表

};

这里和一般的类声明一样，用关键字class声明一个新的类。冒号后面的部分指示这个新类是哪个基类的派生类。所谓“访问控制”是指如何控制基类成员在派生类中的访问属性。它是3个关键字public、private、protected中的一个。一对大括号“{}”是用来声明派生类自己的成员的。

6.2.2 派生类的构造函数和析构函数

定义派生类的构造函数的一般形式如下：

派生类名::派生类名（参数表0） : 基类名（参数表）

{...//函数体}

冒号后“基类名（参数表）”称为成员初始化列表，参数表给出所调用的基类构造函数所需要的实参。实参的值可以来自“参数表0”，或由表达式给出。

构造函数（包括析构函数）是不被继承的，所以一个派生类只能调用它的直接基类的构造函数。当定义派生类的一个对象时，首先调用基类的构造函数，对基类成员进行初始化，然后执行派生类的构造函数，如果某个基类仍是一个派生类，则这个过程递归执行。该对象消失时，析构函数的执行顺序和执行构造函数的顺序正好相反。

6.2.3 类的保护成员

C++语言规定，使用公有方式产生的派生类成员函数可以直接访问基类中定义的或从另一个基类继承来的公有成员，但不能访问基类的私有成员。

在类的声明中，关键字protected之后声明的是类的保护成员。保护成员具有私有成员和公有成员的双重角色；对派生类的成员函数而言，它是公有成员，可以被访问；而对其他函数而言则仍是私有成员，不能被访问。

6.2.4 访问权限和赋值兼容规则

1.公有派生和赋值兼容规则

在公有派生的情况下，基类成员的访问权限在派生类中保持不变。这就意味着：

（1）基类的公有成员在派生类中依然是公有的。

（2）基类的保护成员在派生类中依然是保护的。

（3）基类的不可访问的和私有的成员在派生类中也仍然时不可访问的。

当希望类的某些成员能够被派生类所访问，而又不能被其他的外界函数访问的时候，就应当把它们定义为保护的。

所谓赋值兼容规则是指在公有派生情况下，一个派生类的对象可以作为基类的对象来使用的情况。

注意：静态成员可以被继承，这时基类对象和派生类的对象共享该静态成员。

2.“isa”和“has-a”的区别

类与类之间的关系有两大类：一是继承和派生问题，二是一个类使用另一个类的问题。后者的简单用途是把另一个类的对象作为自己的数据成员或者成员函数的参数。

对于继承，首先要掌握公有继承的赋值兼容规则，理解公有继承“就是一个（isa）”的含义。

分层也可以叫做包含、嵌入或者聚合。公有继承的意思是“isa”。与此相反，分层的意思是指“has-a(有一个)”。

3.公有继承存取权限表

注意：静态成员可以被继承，这时，基类对象和派生类的对象共享该静态成员。

4.私有派生

通过私有派生，基类的私有和不可访问成员在派生类中是不可访问的，而公有和保护成员这时就成了派生类的私有成员，派生类的对象不能访问继承的基类成员，必须定义公有的成员函数作为接口。更重要的是，虽然派生类的成员函数可通过定义自定义的函数访问基类的成员，但将派生类作为基类在继续派生时，这时即使使用公有派生，原基类公有成员在新的派生类中也将是不可访问的。

5.保护派生

派生也可以使用protected。这种派生使原来的权限都降一级使用，即private变为不可访问；protected变为private；public变为protected。因为限制了数据成员和成员函数的访问权限，所以用的比较少。它与private继承的主要区别在于下一级的派生中。

## 6.3 多重继承

一个类从多个基类派生的一般形式如下：

class 类名1：访问控制 类名2，访问控制 类名3,...,访问控制 类名n

{...//定义派生类自己的成员};

## 6.4 二义性及其支配规则

对基类成员的访问必须是无二义性的，如使用一个表达式的含义能解释为可访问多个基类中的成员，则这种对基类成员的访问就是不确定的，称这种访问具有二义性。

6.4.1 作用域分辨符和成员名限定

从类中派生其他类可能导致几个类使用同一个成员函数名或数据成员名。程序必须确切地告诉编译器使用哪个版本的数据成员或成员函数。

如果基类中的名字在派生类中再次声明，则派生类中的名字隐藏了基类中的相应名字。C++可以迫使编译器“看到”当前作用域的外层部分，存取那些被隐藏的名字，这是由作用域分辨运算符“::”实现的（简称作用域运算符）。这一过程叫做作用域分辨。作用域分辨操作的一般形式如下：

类名::标识符

“类名”可以是任意基类或派生类名，“类标识符”是该类中声明的任一成员名。

6.4.2 派生类支配基类的同名函数

基类的成员和派生类新增的成员都具有类作用域，基类在外层，派生类在内层。如果这时派生类定义了一个和基类成员函数同名的新成员函数（因为参数不同属于重载，所以这里是指具有相同参数表的成员函数），派生类的新成员函数就覆盖了外层的同名成员函数。在这种情况下，直接使用成员函数名只能访问派生类的成员函数，只有使用作用域分辨，才能访问基类的同名成员函数。

派生类D中的名字N支配基类B中同名的名字N，称为名字支配规则。

# 第七章 类模板与向量

## 7.1 类模板

如果将类看作包含某些数据类型的框架，然后将这些数据类型从类中分离出来形成一个通用的数据类型T，为这个数据类型T设计一个操作集，并且允许原来那些数据类型的类都能使用这个操作集，这将避免因为类的数据类型不同而产生的重复性设计，其实，类型T并不是类，而是对类的描述，常称之为类模板。在编译时，由编译器将类模板与某种特定数据类型联系起来，就产生一个特定的类（模板类）。利用类模板能大大简化程序设计。

7.1.1 类模板基础知识

1.类模板的成分及语法

关键字class在这里的含义是“任意内部类型或用户定义类型”，但T也可能是结构或类。

对于函数模版及类模板来说，模板层次结构的大部分内容都是一样的，然而在模板声明之后，对类而言便显示出了根本性的差异。为了窗机类模板，在类模板参数表之后应该有类声明。在类中可以像使用其他类型（如int或double）那样使用模板参数。

类模板声明的一般方法如下：

template <类模板参数> class 类名{//类体}；

2.类模板的对象

类模板也称为参数化类。初始化类模板时，只要传给它指定的数据类型，编译器就用指定类型替代模板参数产生相应的模板类。

用类模板定义对象的一般格式如下：

类名<模板实例化参数类型> 对象名（构造函数实参列表）；

类名<模板实例化参数类型> 对象名；//默认或者无参数构造函数

使用模板类时，当给模板实例化参数类型一个指定的数据类型时，编译器自动用这个指定数据类型替代模板参数。

在类体外面定义成员函数时，必须用template重写类模板声明。一般格式如下：

template <模板参数>

返回类型 类名<模板类型参数>::成员函数名(函数参数列表){//函数体}

<模板类型参数> 是指template的“< >”内使用class（或typename）声明的类型参数，构造函数和析构函数没有返回类型。

模板实例化参数类型包括数据类型和值。编译器不能从构造函数参数列表推断出模板实例化参数类型，所以必须显式地给出对象的参数类型。

7.1.2 类模板的派生与继承

类模板也可以继承，继承的方法与普通的类一样。声明模板类继承之前，必须重新声明类模板。模板类的基类和派生类都可以是模板（或非模板）类。类模板的派生与继承很复杂，本书仅简单介绍模板类继承非模板类和从模板类派生一个类模板两种情况。

可以用一个非模板类为一组模板提供一种共同的实现方法。

## 7.2 向量与泛型算法

在数组生存期内，数组的大小是不会改变的。向量是一维数组的类版本，它与数组相似，其中的元素项总是连续存储的，但它和数组不同的是：向量中存储元素的多少可以在运行中根据需要动态地增长或缩小。向量是类模板，具有成员函数。

7.2.1 定义向量列表

向量（vector）类模板定义在头文件vector中，它提供4种构造函数，用来定义由各元素组成的列表。用length表示长度，数据类型用type表示，对象名为name。

向量定义的赋值运算符“=”，允许同类型的向量列表相互赋值，而不管它们的长度如何。向量可以改变赋值目标的大小，使它的元素数目与赋值源的元素数目相同。

向量的第一个元素也是从0开始。

不能使用列表初始化向量，但可以先初始化一个数组，然后把数组的内容复制给向量。

7.2.2 泛型指针

“与操作对象的数据类型相互独立”的算法称为泛型算法。也就是说，泛型算法提供了许多可用于向量的操作行为，而这些算法和想要操作的元素类型无关。这是借助一对泛型指针来实现的。向量具有指示第一元素的标记begin和指示结束的标记end，也就是标识要进行操作的元素空间。如果begin不等于end，算法便会首先作用于begin所指元素，并将begin前进一个位置，然后作用于当前begin所指元素，如此继续进行，直到begin等于end为止。因为end是最后一个元素的下一个位置，所以元素存在的范围是半开区间[begin，end)。

在向量中，泛型指针是在底层指针的行为之上提供一层抽象化机制，取代程序原来的“指针直接操作方式”。假设用T表示向量的参数化数据类型，iterator在STL里面是一种通用指针，它在向量中的作用相当于T\*。用iterator声明向量的正向泛型指针的一般形式如下：

vector<type> :: iterator泛型指针名；

对向量的访问可以是双向的，图中的rbegin和rend是提供给逆向泛型指针的开始和结束标志。逆向泛型指针的加操作是使它向rend方向移动，减操作向rbegin移动。声明逆向泛型指针使用reverse\_iterator。声明的方法如下：

vector<数据类型> :: reverse\_iterator 指针名；

7.2.3 向量的数据类型

向量除了可以使用基本数据类型之外，还可以使用构造类型，只要符合构成法则即可。

7.2.4 向量最基本的操作方法

1.访问向量容量信息的方法

（1）size()：返回当前向量中已经存放的对象的个数

（2）max\_size()：返回向量可以容纳最多对象的个数，一般是操作系统的寻址空间所需容纳的对象的个数。这个参数不是用户指定的，它取决于硬件结构。

（3）capacity()：返回无需再次分配内存就能容纳的对象个数。它的初始值为程序员最初申请的元素个数。当存放空间已满，又增加一个元素时，它在原来的基础上自动翻倍扩充空间，以便存放更多的元素。通俗地讲，也就是已申请的空间。这三者的关系如下。

（4）empty()：当前向量为空时，返回true。

2.访问向量中对象的方法

（1）front()：返回向量中的第一个对象。

（2）back()：返回向量中的最后一个对象。

（3）operator[](size\_type,n)：返回向量中的第n+1个对象（下标为n的向量元素）。

3.在向量中插入对象的方法

（1）push\_back(const T&)：向向量尾部插入一个对象。

（2）insert(iterator it,const T&)：向it所指向的向量位置前插入一个对象。

（3）insert(iterator it,size\_type n,const T&X)：向it所指向量位置前插入n个值为X的对象。

4.在向量中删除对象的方法

（1）pop\_back(const T&)：删除向量中最后一个对象。

（2）erase（iterator it）：删除it所指向的容器对象。

（3）clear()：删除向量中的所有对象，empty()返回true。

## 7.3 出圈游戏

# 第八章 多态性和虚函数

## 8.1 多态性

静态联编所支持的多态性称为编译时的多态性。当调用重载函数时，编译器可以根据调用时使用的实参在编译时就确定下来应调用哪个函数。动态联编所支持的多态性称为运行时的多态性，这由虚函数来支持。虚函数类似于重载函数，但与重载函数的实现策略不同，即对虚函数的调用使用动态联编。

8.1.1 静态联编中的赋值兼容性及名字支配规律

对象的内存地址空间中只包含数据成员，并不存储有关成员函数的信息。这些成员函数的地址翻译过程与其对象的内存地址无关。

声明的基类指针只能指向基类，派生类指针只能指向派生。它们的原始类型决定它们只能调用各种的同名函数area。

8.1.2 动态联编的多态性

当编译系统编译含有虚函数的类时，将为它建立一个虚函数表，表中的每一个元素都指向一个虚函数的地址。此外，编译器也为类增加一个数据成员，这个数据成员是一个指向该虚函数表的指针，通常称为vptr。

虚函数的地址翻译取决于对象的内存地址。编译器为含有虚函数类的对象首先建立一个入口地址，这个地址用来存放指向虚函数表的指针vptr，然后按照类中虚函数的声明次序，一一填入函数指针。当调用虚函数时，先通过vptr找到虚函数表，然后再找出虚函数的真正地址。

派生类能继承基类的虚函数表，而且只要是和基类同名（参数也相同）的成员函数，无论是否使用virtual声明，它们都自动称为虚函数。如果派生类没有改写继承基类的虚函数，则函数指针调用基类的虚函数。如果派生类改写了基类的虚函数，编译器将重新为派生类的虚函数建立地址，函数指针会调用改写过的虚函数。

虚函数的调用规则是：根据当前对象，优先调用对象本身的成员函数。这和名字支配规律类似，不过虚函数是动态联编的，是在执行期“间接”调用实际上欲联编的函数。

## 8.2 虚函数

一旦基类定义了虚函数，该基类的派生类中的同名函数也自动称为虚函数。

8.2.1 虚函数的定义

虚函数只能是类中的一个成员函数，但不能是静态成员，关键字virtual用于类中该函数的声明中。

当在派生类中定义了一个同名的成员函数时，只要该成员函数的参数个数和相应类型以及它的返回类型与基类中同名的虚函数完全一样，则无论是否为该成员使用virtual，它都将成为一个虚函数。

8.2.2 虚函数实现动态性的条件

关键字virtual指示C++编译器对调用虚函数进行动态联编。这种多态性是程序运行到需要的语句处才动态确定的，所以称为运行时的多态性。不过，使用虚函数并不一定产生多态性，也不一定使用动态联编。例如，在调用中对虚函数使用成员名限定，可以强制C++对该函数的调用使用静态联编。

产生运行时的多态性有如下3个前提：

（1）类之间的继承关系满足赋值兼容性规则。

（2）改写了同名函数。

（3）根据赋值兼容性规则使用指针（或引用）。

由于动态联编是在运行时进行的，相对于静态联编，它的运行效率比较低，但它可以使程序员对程序进行高度抽象，设计出可扩充性好的程序。

8.2.3 构造函数和析构函数调用虚函数

在构造函数和析构函数中调用虚函数采用静态联编，即他们所调用的虚函数是自己的类或基类中定义的函数，但不是任何在派生类中重定义的虚函数。

目前推荐的C++标准不支持虚构造函数。由于析构函数不允许有参数，因此一个类只能有一个虚析构函数。虚析构函数使用virtual说明。只要基类的析构函数被说明为虚函数，则派生类的析构函数，无论是否使用virtual进行说明，都自动地成为虚函数。

delete运算符和析构函数一起工作（new和构造函数一起工作），当使用delete删除一个对象时，delete隐含着对析构函数的一次调用，如果析构函数为虚函数，则这个调用采用动态联编。一般来说，如果一个类中定义了虚函数，析构函数也应说明为虚函数，尤其是在析构函数要完成一些有意义的任务时，例如，释放内存。

如果基类的析构函数为虚函数，则在派生类为定义析构函数时，编译器所生成的析构函数也为虚函数。

8.2.4 纯虚函数与抽象类

在许多情况下，不能再基类中为虚函数给出一个有意义的定义，这时可以将它说明为纯虚函数，将其定义留给派生类去做。说明纯虚函数的一般形式如下：

class 类名{

virtual 函数类型 函数名(参数列表)=0;

};

一个类可以说明多个纯虚函数，包含有纯虚函数的类称为抽象类。一个抽象类只能作为基类来派生新类，不能说明抽象类的对象。但可以说明指向抽象类对象的指针（或引用）。

从一个抽象类派生的类必须提供纯虚函数的实现代码，或在该派生类中仍将它说明为纯虚函数，否则编译器将给出错误信息。这说明了纯虚函数的派生类仍是抽象类。如果派生类给了某类所有纯虚函数的实现，则该派生类不再是抽象类。

如果通过同一个基类派生一系列的类，则将这些类总称为类族。抽象类的这一特点保证了进度类族的每个类都具有（提供）纯虚函数所要求的行为，进而保证了围绕这个类族所建立起来的软件能正常运行，避免了这个类族的用户由于偶然失误而影响系统正常运行。

抽象类至少含有一个虚函数，而且至少有一个虚函数是纯虚函数，以便将它与空的虚函数区分开来。下面是两种不同的表示方法：

virtual void area（）=0;

virtual void area(){}

在成员函数内可以调用纯虚函数。因为没有为纯虚函数定义代码，所以在构造函数或虚构函数内调用一个纯虚函数将导致程序运行错误。

## 8.3 多重继承与虚函数

## 8.4 类成员函数的指针与多态性

在派生类中，当一个指向基类成员函数的指针指向一个虚函数，并且通过指向对象的基类指针（或引用）访问这个虚函数时，仍发生多态性。

# 第9章 运算符重载及流类库

## 9.1 运算符重载

9.1.1 重载对象的赋值运算符

编译器在默认情况下为每个类生成一个默认的赋值操作，用于同类的两个对象之间相互赋值。默认的含义是逐个为成员赋值，即将一个对象的成员的值赋给另一个对象相应的成员，这种赋值方式对于有些类可能是不正确的。

C++的关键字“operator”和运算符一起使用就表示一个运算符函数。例如“operator +”表示重载“+”运算符。

9.1.2 运算符重载的实质

C++是由函数组成的，在C++内部，任何运算都是通过函数来实现的。因为任何运算都是通过函数来实现的，所以运算符重载其实就是函数重载，要重载某个运算符，只要重载相应的函数就可以了。与以往稍有不同的是，需要使用新的关键字“operator”，它和C++的一个运算符连用，构成一个运算符函数名，例如“operator+”.通过这种构成方法就可以像重载普通函数那样重载运算符函数operator+()。由于C++已经为各种基本数据类型定义了该运算函数，所以只需要为自己定义的类型重载operator+()就可以了。

一般地，为用户定义的类型重载运算符都要求能够访问这个类型的私有成员，所以只有两条路可走：要么将运算符重载为这个类型的成员函数，要么将运算符重载为这个类型的友元。

C++的运算符大部分都可以重载，不能重载的只有. 、:: 、\* 和 ?: 。前面三个是因为在C++中都有特定的含义，不准重载以避免不必要的麻烦；“?:”则是因为不值得重载。另外，“sizeof”和“#”不是运算符，因而不能重载，而=、（）、[ ] 、->这4个运算符只能用类运算符来重载。

9.1.3 <<、>>和++运算符重载实例

其实，插入符“<<”和提取符“>>”的重载也与其他运算符重载一样，但操作符的左边是流对象的别名而不是被操作的对象，运算符跟在流对象的后面，它们要直接访问类的私有数据，而且流是标准类库，用户只能继承不能修改，更不能是流库的成员，所以它们必须作为类的友元重载。

插入符函数的一般形式如下：

ostream &operator<<(ostream & output，类名 &对象名)

{

return output;

}

output是类ostream对象的引用，它是cout的别名，即ostream&output=cout。调用参数时，output引用cout（即cout的别名）。显然，插入符函数的第2个参数使用引用方式比直接使用对象名的可读性要好一些。

提取符函数的一般形式如下：

istram &operator>>(istream & input，类名&对象名)

{

return input;

}

input是类istream对象的引用。它是cin的别名，即istream&input=cin。调用参数时，input引用cin（即cin的别名）。

另外，提取符函数需要返回新的对象值，所以应该使用引用，即“类名&对象名”，不能使用“类名 对象名”。插入符函数不改变对象的值，所以两种方法都可以。

显然，运算符“<<”重载函数有两个参数，第1个是ostream类的一个引用，第2个是自定义类型的一个对象。这个重载方式是友元重载。这个函数的返回类型是一个ostream类型的引用，在函数中实际返回的是该函数的第一个参数，这样做是为了使得“<<”能够连续使用。

有些C++编译器不区分前缀或后缀运算符，这时只能通过对运算符函数进行重载来反映其为前缀或后缀运算符。

注意不能自己定义新的运算符，只能是把C++原有的运算符用到自己设计的类上面去。同时，经过重载，运算符并不改变原有的优先级，也不改变它所需的操作数目。当不涉及到定义的类对象时，它仍然执行系统预定义的运算，只有用到自己定义的对象上，才执行新定义的操作。

应该根据需要进行运算符重载。不排除在某些特殊情况下会有一些特殊的需要，但大多数情况下不会将运算符“+”重载为两个复数相减的运算（尽管有能力这么做）。一般总是要求运算符重载合乎习惯。

9.1.4 类运算符和友元运算符的区别

如果运算符所需的操作数（尤其是第一个操作数）希望进行隐式类型转换，则运算符应通过友元来重载。另一方面，如果一个运算符的操作需要修改类对象的状态，则应当使用类运算符，这样更符合数据封装的要求。但参数是引用还是对象，则要根据运算符在使用中可能出现的情况来决定。

如果对象作为重载运算符函数的参数，则可以使用构造函数将常量转换成该类型的对象。如果使用引用作为参数，因为这些常量不能作为对象名使用，所以编译系统就要报错。

9.1.5 下标运算符“[ ]”的重载

运算符“[ ]”只能用类运算符来重载。

## 9.2 流类库

C++的流类库由几个进行I/O操作的基础类和几个支持特定种类的源和目标的I/O操作的类组成。

9.2.1 流类库的基础类

在C++中，输入输出时同流来完成的。C++的输出操作将一个对象的状态转换成一个字符序列，输出到某个地方。输入操作也是从某个地方接收到一个字符序列，然后将其转换成一个对象的状态所要求的格式。这看起来很像数据在流动，于是把接收输出数据的地方叫做目标，把输入数据来自的地方叫做源。而输入和输出操作可以看成字符序列在源、目标以及对象之间的流动。C++将与输入和输出有关的操作定义为一个类体系，放在一个系统库里，以备用户调用。这个执行输入和输出操作的类体系就叫做流类，提供这个流类实现的系统库就叫做流类库。C++的流类库由几个进行I/O操作的基础类和几个支持特定种类源和目标的I/O操作类组成。

在C++中，如果在多条继承路径上有一个汇合处，则称这个汇合处的基类为公共基类（ios符合条件）。因为可以通过不同的访问路径访问这个基类，从而使公共的基类会产生多个实例，这样会引起二义性。如果想使这个公共的基类只产生一个实例，则可以将这个基类说明为虚基类。ios类就是isrream类和ostream类的虚基类，用来提供对流进行格式化I/O操作和错误处理的成员函数。用关键字virtual可将公共基类说明为虚基类，虚基类的定义很难处理，这就是为什么最初的C++语言没有能支持多重继承的原因。

从ios类公有派生的istream和ostream两个类分别提供对流进行提取操作和插入操作的成员函数，而iostream类通过组合istream类和ostream类来支持对一个流进行双向（也就是输入和输出）操作，它并没有提供新的成员函数。

C++流类库预定义了4个流，它们是cin、cout、cerr、clog。事实上，可以将cin视为类istream的一个对象，而将cout视为类ostream的对象。

流是一个抽象概念，当实际进行I/O操作时，必须将流和一种具体的物理设备（比如键盘）联接起来。C++的流类库预定义的4个流所联接起来的具体设备为：

cin 与标准输入设备相联接

cout 与标准输出设备相联接

cerr 与标准错误输出设备相联接（非缓冲方式）

clog 与标准错误输出设备相联接（缓冲方式）

9.2.2 默认输入输出的格式控制

关于数值数据，默认方式能够自动识别浮点数并用最短的格式输出，还可以将定点数分成整数和小数部分。

特别要注意字符的读入规则。对单字符来讲，它将舍去空格，直到读到字符为止。对于单字符对象a，b和c，“cin>>a>>b>>c；”能将连续的3个字符分别正确地赋给相应对象。

对字符串来讲，它从读到第一个字符开始，到空格符结束。对于字符数组，使用数组名来整体读入。但对于字符指针，尽管为它动态分配了地址，也只能采取逐个赋值的方法，它不仅不以空格结束，反而舍弃空格（读到字符才计数）。因为字符串没有结束位，所以将字符串作为整体输出时，有效字符串后面将出现乱码。不过，可以手工增加表示字符串的结束符“\0”来消除乱码。

当用键盘同时给一个单字符对象和一个字符串对象赋值时，不要先给字符串赋值。如果先给它赋值，应该强行使用结束符。

Bool(布尔型)在VC6.0中把输入的0识别为false，其他的值均识别为1。输出时，只有0和1两个值。如果默认输入输出格式不能满足自己的要求，就必须重载它们。

9.2.3 使用ios\_base类

1.ios\_base类简介

ios\_base类派生ios类，ios类又是istream类和ostream类的虚基类。

表9.1 常量名及含义

|  |  |
| --- | --- |
| 常量名 | 含义 |
| skipws | 跳过输入中的空白 |
| left | 输出数据按输出域左边对齐输出 |
| right | 输出数据按输出域右边对齐输出 |
| intermal | 在指定任何引导标志或基之后填充字符 |
| dec | 转换基数为十进制形式 |
| oct | 转换基数为八进制形式 |
| hex | 转换基数为十六进制形式 |
| showbase | 输出带有一个表示制式的字符 |
| showpoint | 浮点输出时必须带有一个小数点和尾部的0 |
| uppercase | 十六进制数值输出使用大写A~F，科学计数显示使用大写字母E |
| showpos | 在正数前添加一个“+”号 |
| scientific | 使用科学计数法表示浮点数 |
| fixed | 使用定点形式表示浮点数 |
| untibuf | 每次插入之后，ostream::osfx刷新该流的缓冲区。默认缓冲单元为cerr |
| boolalpha | 把逻辑值输出为true和false（否则输出为1和0） |
| adjustfield | 对齐方式域（与left、right、internal配合使用，例如ios\_base::adjustfield） |
| basefield | 数字方式域（与dex、oct、hex配合使用，例如ios\_base::basefield） |
| floatfield | 浮点方式域（与fix、scientific配合使用，例如ios\_base::floatfield） |

表9.2 处理标志的成员函数及其作用

|  |  |
| --- | --- |
| 成员函数 | 作 用 |
| long flags（long） | 允许程序员设置标志字的值，并返回以前所设置的标志字 |
| long flags() | 仅返回当前的标志字 |
| long setf(long,long) | 用于设置标志字的某一位，第2个参数指定所要操作的位，第1个参数指定为该参数所设置的值 |
| long setf(long) | 用来设置参数指定的标志位 |
| long unsetf(long) | 清除参数指定的标志位 |
| int width(int) | 返回以前设置显示数据的域宽 |
| int width() | 只返回当前域宽（默认宽度为0） |
| char fill(char) | 设置填充字符，设置的宽度小时，空余的位置用填充字符来填充，默认条件下是空格。这个函数返回以前设置的填充字符 |
| char fill() | 获得当前的填充字符 |
| int precision(int) | 返回以前设置的精度（小数点后的小数位数） |
| int precision | 返回当前设置的精度 |

2.直接使用格式控制

表9.1中的名字可以直接用在系统提供的输入输出流中，并且有些是成对的。加no前缀标识取消原操作。

用width(int)设置宽度的效果只对一次输入或输出有效，在完成一次输入或输出之后，宽度设置自动恢复为0（表示按实际数据宽度输入输出）。在设置输入时，实际输入的字符串最大长度为n-1(宽度n计入结束符)。setw(int)只对紧跟其后的输出有效。

3.使用成员函数

在使用成员函数进行格式控制的时候，setf用来设置，unsetf用来恢复默认设置。它们被流对象调用，使用表9.1的常量设置。

## 9.3 文件流

在C++里，文件操作是通过流来完成的。C++总共有输入文件流、输出文件流和输入输出文件流3种，并已将它们标准化。

要打开一个输入文件流，需要定义一个ifstream类型的对象；要打开一个输出文件流，需要定义一个ofstream类型的对象；如果要打开输入输出文件流，则要定义一个fstream类型的对象。这3种类型都定义在头文件<fstream>里。

9.3.1 使用文件流

流类具有支持文件的能力，在建立和使用文件时，就像使用cin和cout一样方便。

可以总结出对文件进行操作的方法如下：

（1）打开一个相应的文件流.

（2）把这个流和相应的文件关联起来。

因为ifstream、ofstream和fstream这3个类都具有自动打开文件的构造函数，而这个构造函数就具有open()的功能。因此，事实上可以用一条语句：

ofstream myStream("myText.txt");

来完成上述两步。如果指定文件路径，路径中的“\”号必须使用转义字符表示。

（3）操作文件流。

综上所述，流是I/O流类的中心概念。流是一种抽象，它负责在数据的生产者和消费者之间建立联系并管理数据的流动，程序将流对象看做是文件对象的化身。

一个输出流对象是信息流动的目标，ofstream是最重要的输出流。一个输入流对象是数据流动的源头，ifstream是最重要的输入流。一个iostream对象可以是数据流动的源或目的，fstream就是从它派生的。

其实，文件流的对象是字节流，而且文本文件和二进制文件均是字节流。将文件与流关联起来，就是使用“>>”和“<<”对文件进行读写。但不能使用空格，这是由“>>”和“<<”原来的性质所决定的。

9.3.2 几个典型流成员函数

1.输出流的open函数

open函数的原型如下：

void open(char const \*,int filemode,int =filebuf::openprot);

它有3个参数，第一个是要打开的文件名，第2个是文件的打开方式，第3个是文件的保护方式，一般都使用默认值。第2个参数可以取如下所示的值：

ios\_base::in 打开文件进行读操作，这种方式可避免删除现存文件的内容

ios\_base::out 打开文件进行写操作，这是默认模式

ios\_base::ate 打开一个已有的输入或输出文件并查找到文件尾

ios\_base::app 打开文件以便在文件尾部添加数据

ios\_base::binary 指定文件以二进制方式打开，默认为文本方式

ios\_base::trunc 如文件存在，将其长度截断为零并清除原有内容

除ios\_base::app方式之外，在其他几种方式下，文件刚刚打开是，指示当前读写位置的文件指针都定位于文件的开始位置，而ios\_base::app使文件当前的写指针定位于文件尾。这几种方式也可以通过“或”运算符“|”同时使用。

2.输入流类的open函数

使用默认构造函数建立对象，然后调用open成员函数打开文件。

输入流的打开模式如下所示：

ios\_base::in 打开文件用于输入（默认）

ios\_base::binary 指定文件以二进制方式打开，默认为文本方式

3.close成员函数

close成员函数用来关闭与一个文件流相关联的磁盘文件。如果没有关闭该文件，因为文件流是对象，所以在文件流对象的生存期结束时，将自动关闭该文件。不过，应养成及时关闭不再使用的文件的习惯，以避免误操作或者干扰而产生的错误。

4.错误处理函数

在对一个流对象进行I/O操作时，可能会产生错误。当错误发生时，可以使用文件流的错误处理成员函数进行错误类型判别。这些成员函数的作用见表9.3。

可以使用这些函数检查当前流的状态，例如：

if(cin.good()) cin>>data;

函数clear更多地是用于在已知流发生错误的情况下清除流的错误状态，也可以用于设置流的错误状态。除非发生致命错误（hardfail），否则可以使用函数clear清除流的错误状态。

“！”运算符已经过了重载，它与fail函数执行相同的功能，因此表达式if(!cout)等价于if(cout.fail())，if(cout)等价于if(!cout.fail())。

表9.3 成员函数及其功能

|  |  |
| --- | --- |
| 函 数 | 功 能 |
| bad() | 如果进行非法操作，返回true，否则返回false |
| clear() | 设置内部错误状态，如果用缺省参量调用则清除所有错误位 |
| eof() | 如果提取操作已经到达文件尾，则返回true，否则返回false |
| good() | 如果没有错误条件和没有设置文件结束标志，返回true，否则返回false |
| fail() | 与good相反，操作失败返回false，否则返回true |
| is\_open() | 判定流对象是否成功地与文件关联，若是，返回true，否则返回false |

9.3.3 文件存取综合实例

# 第10章 面向对象设计实例

## 10.1 过程抽象和数据抽象

抽象是形成概念的必要手段，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。

对于分析而言，抽象原则具有两方面的意义：

（1）尽管问题域中的事物很复杂，但分析员并不需要了解和描述其全部特征，只需要分析研究与系统目标有关的事物及其本质特征。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，抽取其共同特征可以得到一批事物的抽象概念。

过程抽象是指任何一个完成确定功能的操作序列，其使用者都可以把它看做一个单一的实体，尽管实际上它可能是由一系列更低级的操作完成的。运用过程抽象，软件开发者可以把一个复杂的功能分解为一些子功能，过程抽象对于在对象范围内组织对象的成员函数是很有用的。

数据抽象是指根据施加于数据之上的操作来定义数据类型，并限定数据的值只能由这些操作来修改和观察。

## 10.2 发现对象并建立对象层

软件开发者将被开发的整个业务范围称作“问题域”，可以按如下步骤考虑发现对象并建立对象层。

1.将问题域和系统责任作为出发点

2.正确运用抽象原则

在OOA中正确地运用抽象原则，首先要舍弃那些与系统责任无关的事物，只注意与系统责任有关的事物。其次，对于与系统责任有关的事物，也不是把他们的任何特征都在相应的对象中表达出来，而是要舍弃那些与系统责任无关的特征。判断事物是否与系统责任有关的关键问题，一是该事物是否为系统提供了一些有用的信息（是否需要系统为它保存和管理某些信息）；二是它是否向系统提供了某些服务（是否需要系统描述它的某些行为）。

3.寻找候选对象的基本方法

寻找候选对象的基本方法的主要策略是从问题域、系统边界和系统责任三方面找出可能有的候选对象。

4.审查和筛选对象

5.异常情况的检查和调整

一般认为下述情况都算异常情况，则需要进行调整。

（1）类的数据成员或成员函数不适合该类的全部对象。

（2）不同类的数据成员或成员函数相同或相似。

（3）对同一事物的重复描述。

## 10.3 定义数据成员和成员函数

为了发现对象的数据成员，首先应考虑借鉴以往的OOA结果，看看相同或相似的问题域是否有已开发的OOA模型，尽可能复用其中同类对象数据成员的定义。然后重点研究当前问题域和系统责任，针对本系统应该设置的每一类对象，按照问题域的实际情况，以系统责任为目标进行正确地抽象，从而找出每一类对象应有的数据成员。

1.寻找数据成员的一般方法

2.审查与筛选数据成员

对于初步发现的数据成员，要进行审查和筛选。即对每个数据成员提出以下问题：

（1）这个数据成员是否体现了以系统责任为目标的抽象。

（2）这个数据成员是否描述了这个对象本身的特征。

（3）该属性是否符合人们日常的思维习惯。

（4）这个数据成员是不是可以通过继承得到。

（5）是否可以从其他数据成员直接导出的数据成员。

3.定义成员函数

使用中要特别注意区分成员函数、非成员函数和友元函数三者。

## 10.4 如何发现基类和派生类结构

1.学习当前领域的分类学知识

分析员应该学习一些与当前问题域有关的分类学知识，，因为问题域现行的分类方法（如果有），往往比较正确地反映了事物的特征、类别以及各种概念的一般性与特殊性。

2.按照常识考虑事物的分类

从不同的角度考虑问题域中事物的分类，可以形成一些建立基类与派生类结构的初步设想，从而启发自己发现一些确实需要的基类与派生类结构。

3.构建基类与派生类

按照基类与派生类结构的两种定义，可引导我们从两种不同的思路去发现基类与派生类结构。一种思路是把每一个类看做是一个对象集合，分析这些集合之间的包含关系。另一种思路是看一个类是不是具有另一个类的全部特征。这包括两种情况：一是建立这些类时已经计划让某个类继承另一个类的全部成员，此时应建立基类与派生类结构来落实这种继承；另一种情况是起初只是孤立地建立每一个类，现在发现一个类中定义的成员（数据成员和成员函数）全部在另一个类中重新出现了，此时应该考虑建立基类与派生类结构，把后者作为前者的派生类，以简化定义。

4.考察类的成员

对系统中的每个类，从以下两个方面考察它们的成员；一是看一个类的成员是否适合这个类的全部对象。如果某些数据成员和成员函数只能适合该类的一部分对象，说明应该从这个类中划分出一些派生类，建立基类与派生类关系。

另一方面检查是否有两个（或更多的）类含有一些共同的数据成员和成员函数。如果有，则考虑若把这些共同的数据成员和成员函数提取出来后能否构成一个在概念上是包含原来那些类的基类，组成一个基类与派生类结构。

还要对发现的结构进行审查、调整和简化，处理异常情况，才能建立合适的结构。

## 10.5 接口继承与实现继承

现在假设设计一个可以供其他类继承的基类，派生类使用公有继承方式。公有继承实际上是由两个不同部分组成的，即函数接口的继承和函数实现的继承。可概括成如下两点：

（1）继承的总是成员函数的接口。对于基类是正确的任何事情，对于它的派生类必须也是正确的。

（2）声明纯虚函数的目的是使派生类仅仅继承函数接口，而纯虚函数的实现则由派生类去完成。

（3）声明虚函数的目的是使派生类既能继承基类对此虚函数的实现，又能继承虚函数提供的接口。

（4）声明实函数的目的是使派生类既能继承基类对此实函数的实现，又能继承实函数提供的接口。

1.纯虚函数

纯虚函数最显著的两个特征是：

（1）它们必须由继承它的非抽象类重新说明

（2）它们在抽象类中没有定义

2.虚函数

虚函数的情况不同于纯虚函数，派生类继承虚函数的接口，虚函数一般只提供一个可供派生类选择的实现。之所以声明虚函数，目的在于使派生类既继承函数接口，也继承虚函数的实现。

3.实函数

一个实成员函数指定它自己在派生类中保持不变。因此，声明实函数的目的是使派生类既继承这个函数的实现，也继承其函数接口。

4.避免重新定义继承的实函数

虽然在派生类中可以重定义基类的同名实函数，但是从使用安全角度来看，为了提高程序质量，在实际应用中应避免这样做。

## 10.6 设计实例