HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY PRESS

C++程序设计实践教程

C++ CHENG XU SHE JI SHI JIAN JIAO CHENG

马光志 编著

华中科技大学出版社

C++程序设计
实践教程

马光志　编著

华中科技大学出版社
（华中理工大学出版社）

图书在版编目(CIP)数据

C++程序设计实践教程/马光志 编著
武汉:华中科技大学出版社, 2001年2月
ISBN 7-5609-2366-6

Ⅰ.C…
Ⅱ.马…
Ⅲ.C语言-程序设计
Ⅳ.TP312

C++程序设计实践教程 马光志 编著

责任编辑:曾 光 封面设计:刘 卉
责任校对:张 欣 责任监印:熊庆玉

出版发行:华中科技大学出版社 武昌喻家山 邮编:430074 电话:(027)87545012

经 销:新华书店湖北发行所

录 排:华中科技大学惠友科技文印中心
印 刷:武汉市科普教育印刷厂

开本:787×1092 1/16 印张:16.25 字数:360 000
版次:2001年2月第1版 印次:2001年2月第1次印刷 印数:1—3 000
ISBN 7-5609-2366-6/TP·421 定价:19.80元

内 容 简 介

本书全面系统地介绍了C++语言的基本概念，并为这些概念精心设计和挑选了实例。书中内容包括：类、对象、封装、继承、重载、多态、引用、内联、友元、模板、异常、断言、虚函数、抽象类、静态成员、成员指针、名字空间、流及类库等等。为了使本书介绍的对象建模技术更具实战性，本书按照面向对象的系统分析与设计步骤，完整地介绍了一个对象建模实例，并用C++语言进行了面向对象的程序设计。

本书内容新颖、通俗易懂、注重理论与实践相结合，既可作为高等院校计算机及其相关专业的教材，又可作为C++初学者和高级程序设计人员的参考书。

前　言

C++是由 Bjarne Stroustrup 于 1986 年在 AT&T 贝尔实验室开发的。开发这一语言的目的在于通过数据封装减小程序变量的副作用，从而降低程序的复杂性并提高程序的可靠性。C++是 C 语言的直接扩展，C++的多继承机制能更好地描述对象的属性和行为，因此，C++特别有助于开发大型软件系统。到目前为止，C++的语法和语义还在发展变化，已基本具备面向对象语言的全部特点。

许多教材往往用大量篇幅介绍 C 或 C++的开发环境，而没有全面深刻地介绍类和面向对象的程序设计。许多人在学完 C++后，觉得 C 和 C++没有多大区别，体会不到面向对象的程序设计的优点，这与教材忽视概念的应用背景介绍有很大关系。

本书全面深刻地介绍 C++的类，不重复介绍 C 语言的有关概念，所有章节均以类为中心，由浅入深地逐步展开，尽量避免前后参照和相互关联，力图使初学者理解而不是死记概念。在详细介绍有关概念的基础上，本书通过精心设计的例题，具体说明有关概念的使用方法。绝大部分例题能在 Borland C++ Ver 3.1 上运行，少数例题必须在 5.0 甚至更高版本上才能运行。

第 1 章介绍 C++的特点，阐述了面向对象的有关概念；第 2 章介绍类型和函数，该章是以后许多章节的基础；第 3 章介绍类的基本概念以及 new 和 delete 的用法；第 4 章介绍作用域、名字空间、成员指针等高级概念；第 5 章介绍静态成员及友元函数；第 6 章介绍单继承类；第 7 章介绍虚函数及抽象类；第 8 章介绍多继承类与虚基类；第 9 章介绍运算符重载；第 10 章介绍函数模板及类模板；第 11 章介绍异常处理与断言；第 12 章介绍 C++的流及类库；第 13 章介绍对象模型的有关概念，建立并实现了赌场对象模型。建议学时数为 60~70 学时，第 12 章和第 13 章可根据实际情况取舍。

在本书编写的过程中，卢正鼎教授给予了大力的支持，并提出了宝贵意见，笔者的导师袁蒲佳及其他老师也提出了有益建议。在此，笔者向他们表示深深的感谢。对本书所存在的错误和不足之处，诚恳地希望广大读者批评指正。

马 光 志

2000 年 6 月于华中理工大学

目 录

第 1 章 引 论 (1)

1.1 程序设计语言 (1)
1.2 程序编译技术 (2)
1.3 面向对象的语言及程序设计 (4)
1.4 面向对象的基本概念 (6)
1.5 C++语言的特点 (10)
1.6 C++的程序结构 (11)
练习题 (15)

第 2 章 C++的变量、类型及函数 (16)

2.1 声明及定义 (16)
2.2 类型定义 (18)
2.3 引用类型 (24)
2.4 函数参数 (30)
2.5 函数内联 (34)
练习题 (36)

第 3 章 C++的类 (38)

3.1 类的声明及定义 (38)
3.2 访问权限 (42)
3.3 内联及位段 (45)
3.4 new 和 delete (49)
3.5 隐含参数 this (53)
3.6 对象初始化 (54)
3.7 类的存储空间 (57)
练习题 (58)

第 4 章 作用域及成员指针 (61)

4.1 作用域 (61)
4.2 名字空间 (65)
4.3 成员指针 (70)
4.4 const、volatile 和 mutable (73)
4.5 引用对象 (78)
练习题 (83)

第5章 静态成员与友元 (86)
5.1 静态数据成员 (86)
5.2 静态函数成员 (90)
5.3 静态成员指针 (92)
5.4 成员友元 (94)
5.5 普通友元 (97)
练习题 (102)
第6章 单继承类 (105)
6.1 单继承类 (105)
6.2 派生控制 (107)
6.3 成员访问 (110)
6.4 构造与析构 (112)
6.5 父类和子类 (114)
6.6 派生类的存储空间 (118)
练习题 (119)
第7章 虚函数 (123)
7.1 虚函数 (123)
7.2 虚析构函数 (126)
7.3 抽象类 (128)
7.4 友元、绑定 (133)
7.5 类的存储空间 (135)
练习题 (138)
第8章 多继承类 (140)
8.1 多继承类 (140)
8.2 虚基类 (142)
8.3 派生类成员 (145)
8.4 构造与析构 (147)
8.5 类的存储空间 (150)
练习题 (155)
第9章 运算符重载 (158)
9.1 概述 (158)
9.2 运算符函数参数 (161)
9.3 赋值与调用 (165)
9.4 强制类型转换 (170)
9.5 重载 new 和 delete (173)

9.6 表运算实例 (175)
练习题 (177)
第 10 章 模 板 (183)
10.1 函数模板 (183)
10.2 模板函数 (185)
10.3 类模板 (187)
10.4 模板类及覆盖 (193)
10.5 内存回收实例 (198)
练习题 (200)
第 11 章 异常处理与断言 (201)
11.1 异常处理 (201)
11.2 catch 顺序 (204)
11.3 异常接口 (206)
11.4 异常类型 (208)
11.5 异常对象的析构 (210)
11.6 断言 (215)
练习题 (216)
第 12 章 C++流及类库 (217)
12.1 流类概述 (217)
12.2 输出流 (218)
12.3 输入流 (222)
12.4 文件流 (223)
12.5 串流处理 (225)
第 13 章 对象分析与设计 (227)
13.1 概 述 (227)
13.2 对象模型 (228)
13.3 对象分析技术 (231)
13.4 对象设计与实现 (232)
13.5 一个实例 (234)
13.6 对象实现 (240)
附录 A C++运算符表 (247)
附录 B ASCII 字符码表 (249)
参考文献 (251)

引 论

1.1 程序设计语言

程序是一个十分广泛的概念。当宣布开会时，便启动了会议程序；当打开电源时，便启动了计算机程序。会议程序可用汉语描述，也可用其他语言描述。计算机程序也能用不同的语言(例如，机器语言或更为通用的高级程序设计语言等)描述。

机器语言是一种计算机自身可以识别的语言，机器语言程序是由机器指令和数据形成的二进制文档。鉴于机器语言程序难于理解、编程繁琐、不易维护，于是，便出现了一种简单的用符号表示的汇编语言。汇编语言和机器语言同为低级语言，但汇编语言比机器语言稍高级。汇编语言程序必须经过汇编程序的翻译，才能转换为计算机可以理解和执行的机器语言。同机器语言相比，汇编语言更易理解和用于编程，但它仍然是一种面向机器的程序设计语言。

高级语言是一种类似于自然语言的、形式化的程序设计语言。严格地讲，高级语言不局限于任何一种计算机，用高级语言编写的程序可以在不同机器之间相互移植。高级语言容易理解、易于编程，因此，很容易普及。目前，比较常用的高级语言有 C、PASCAL、FORTRAN、COBOL、ADA、BASIC 等。

高级语言程序必须经过编译程序的翻译，才能转换为某种等价的低级语言程序。编译程序比汇编程序复杂得多，编译程序必须经过若干编译过程，才能将高级语言程序编译成低级语言程序。在编译程序的过程中，通常要生成中间代码。

高级语言是面向过程的语言，即描述的是问题求解的过程、算法或方法。问题求解的常用手段是功能分解，功能分解的结果用高级语言结构化地实现，结构化程序设计对缓解软件危机起了一定作用，但随着软件产业的发展这种作用越来越有限。

面向对象的程序设计语言为解决软件危机提供了更为先进的手段。在一定程度上，“程序=数据结构+算法”表明了高级语言描述数据和算法的能力，但缺乏一种将数据和方法封装在一起的机制，面向对象语言的最显著的特征就是具有这种封装机制。目前，比较普及的面向对象的语言有 SMALLTALK、JAVA、EIFFEL 以及 C++等等。

除上述通用型的程序设计语言之外，还有一些为特定应用专门设计的专用程序设计语言．例如，为数据库管理系统设计的数据查询语言 SQL，为人工智能应用系统设计的逻辑推理语言 LISP、PROLOG，以及为定义程序设计语言设计的元语言 BNF 等。

元语言是一种更高级、更抽象的语言，可以简洁、完整、精确、可靠地描述程序设计语言的语言规范。要系统地掌握一门程序设计语言，就必须对有关元语言有所了解，否则便难于掌握程序设计语言，更谈不上灵活熟练地应用程序设计语言。C 和 C++的语言规范通常用 BNF 描述，因此，只有熟悉 BNF 才能真正掌握 C 和 C++语言。

值得注意的是，随着计算机软硬件技术的发展，程序设计语言的某些概念也可能发生变化。过去认为，凡是计算机自身识别的语言就是机器语言。这种观点已不适应现代计算机的发展状况，例如，已经出现了能直接识别和执行 JAVA 语言的网络计算机。

1.2 程序编译技术

程序设计语言的翻译方式通常有两种，即解释方式和编译方式。在解释方式下，由解释器读入高级语言程序，并逐条解释和执行程序的每条语句；在编译方式下，由编译程序读入高级语言程序，并编译成某种等价的目标语言程序，然后由操作系统装入和执行目标语言程序。

就程序最终的执行速度而言，用编译方式实现要比用解释方式实现快。程序设计语言可以用多种方式实现。例如，BASIC 语言既可用解释方式实现，也可用编译方式实现，C 和 C++通常只以编译方式实现，以下重点介绍编译技术。

1.2.1 编译技术

高级语言程序通常要经过预处理、词法分析、语法分析、代码生成和模块连接等环节，才能被编译成可被计算机执行的目标语言程序。每个编译环节的输出通常作为下一编译环节的输入，该输出通常以文件的形式存放在磁盘上。编译结束时，通常只保留目标文件和目标语言程序，中间文件在编译结束后会被自动删除。

对于 C 和 C++编译程序而言，预处理过程通常要完成由#define 定义的宏替换、由#include 定义的文件包含以及由#if 等定义的条件编译，最终生成不包含#define、#include 及其他编译命令的文本文件。在预处理过程中，注解、多余的空格或空行均会被删除。

词法分析的任务是分析识别出各种有意义的词法记号，它以预处理程序输出的文本文件作为输入。词法记号是高级语言程序中所使用的各种单词，包括保留字、标识符、操作符、分隔符以及各种常量。识别的单词连同其他相关信息一起存放在符号表中，供语法分析程序检查语言的语法结构时使用。

语法分析程序根据程序设计语言的形式规范检查程序语法的正确性。程序设计语言的形式规范严格地描述了变量声明、函数及各种语句的语法结构。语法分析程序分析源程序的程序语义，其输出将作为代码生成程序的输入。

代码生成程序根据已经识别的程序语义生成和源程序等价的某种形式的中间代码。这种代码可以是一种需要再次解释的代码，也可以是一种带有连接信息的特定计算机的机器指令代码。

模块连接程序将中间代码同标准库或非标准库连接起来，生成可被计算机执行的目标

语言程序。标准库包括输入/输出函数、数学运算函数及绘图处理函数等，这些函数本身不是C或C++的一部分；非标准库包括项目开发人员或其他公司开发的与项目密切相关的函数。C++除了提供同C兼容的各种标准库外，还提供了大量的用于面向对象开发的各种类库。

同C程序一样，C++程序可由多个模块文件构成。程序的模块文件可以独立或一起编译，生成的目标文件可通过库管理程序加入到非标准库中。库管理程序通常随编译程序一起提供。使用库管理程序可以降低软件开发和维护的难度，提高软件项目的开发质量和开发效率。

1.2.2 模块连接

模块连接的任务是将编译生成的目标文件同标准和非标准库连接起来，生成具有代码段、数据段、堆栈段的目标语言程序。模块连接分为静态和动态连接两种形式，两者的区别在于：静态连接由编译程序完成，动态连接由操作系统完成。因此，静态连接又称为编译时连接，动态连接又称为运行时连接。

静态连接将被调函数的函数体拷贝到目标语言程序中。动态连接则只在目标语言程序中保存被调函数的描述信息。当目标语言程序开始运行时，静态连接程序的函数体随目标语言程序一起装入内存，因而执行时能很快找到函数的入口并执行；动态连接程序的函数体并不随目标语言程序装入内存，操作系统在接到调用命令后再从外存储器装入该函数,并且保证同一个函数不会在内存中出现多个副本。由此可知，静态连接的程序运行速度较快，动态连接的程序运行速度较慢。

现代操作系统一般为多任务操作系统，并且几乎都支持动态连接和动态装入。在多任务环境下，运行静态连接的程序会造成某些函数在内存中出现多个副本，而运行动态连接的程序则可保证同一函数只有一个共享副本。因此，运行动态连接的程序会为操作系统节省大量内存。

为了适应操作系统的多任务环境，编译程序大多提供了连接方式选择开关，程序员可以决定使用静态连接还是动态连接。动态连接的程序需要动态连接库才能运行，除非运行环境存在相应的动态连接库，否则，必须安装相应的动态连接库才能运行动态连接程序。

C和C++的编译程序采用了不同的换名策略。所谓换名就是将高级语言的变量名、函数名等转换为低级语言或中间语言的变量名、函数名。当一个C++程序同时由C和C++模块文件构成时，即使C和C++两种模块文件以相同的方式说明变量和函数，由于编译后双方变量和函数的替代名称不一样，也会造成一种模块无法访问另一种模块的变量和函数。关于C++如何访问C的变量和函数可参考ANSI C++标准的有关规定。

1.2.3 函数绑定

绑定(binding)就是函数的入口地址同函数调用相联系的过程。函数调用指令由操作码和地址码两部分构成，地址码就是被调用函数的入口地址。绑定就是要计算被调用函数的入口地址，并将该地址存放到函数调用指令的地址码部分。

绑定分为静态绑定和动态绑定两种形式，两者的区别在于：静态绑定在程序执行前完成，由编译程序或操作系统装入程序计算函数的入口地址；动态绑定则在程序执行过程中完成，由程序自身计算函数的入口地址。

传统的过程型程序设计语言仅支持静态绑定。对于代码不可浮动的程序，函数的入口地址已在编译时全部计算完毕，操作系统装入程序直接装入可执行程序文件即可；对于代码可以浮动的程序，操作系统装入程序需要重新计算函数的入口地址。因此，静态绑定既可以由编译程序静态连接完成，也可以由操作系统动态连接完成。

对象型的程序设计语言 C++既支持静态绑定，又支持动态绑定。C++的动态绑定依赖于运行时指针所指对象的实际类型，根据对象类型程序自动计算并连接多态函数的入口地址。由此可见，动态绑定需要计算和连接多态函数入口的执行代码。这段代码不是由程序员编写的，而是由编译程序根据可能的对象类型自动生成的。因此，动态绑定是由编译程序自动生成的、静态或动态连接的、程序自身激活并执行的过程。

1.3　面向对象的语言及程序设计

对象的类型简称为类，类是一种能封装对象“属性”及其“操作”的类型。程序设计语言 ADA、MODULA-2 虽然也提供了封装功能，但它们不能封装“属性”或数据成员，这些语言被称为基于对象的程序设计语言。比较普及的面向对象的程序设计语言有 SMALLTALK、EIFFEL、ACTOR、CLOS、OBJECT-ORIENTED PASCAL、OBJECTIVE-C、C++以及 JAVA 等。

1.3.1　面向对象语言的发展

人们认识世界的过程是抽象程度不断提高的过程。程序设计语言经历了过程抽象、语法抽象、数据抽象和进程抽象等发展过程，在面向过程的 ALGOL、ADA、MODULA-2 等语言的基础上，逐步演变形成了面向对象的程序设计语言。

20 世纪 60 年代末，美国国防部投入了巨大的人力和物力，研制开发了 ADA 语言。ADA 语言并非面向对象的程序设计语言，但它所具有的模块化、信息隐蔽、数据抽象和并发执行等特点，对于面向对象的方法和技术起到了积极的推动作用。人们普遍认为，ADA 是一种基于对象的程序设计语言。

20 世纪 70 年代初，面向数据的抽象和面向过程的抽象结合，产生了具有面向对象雏形的 SIMULA。SIMULA 允许用户创建面向对象的系统，但系统的实现使用的却是面向过程和数据的 ALGOL 语言。

1976 年出现了 SMALLTALK-72，其列表和控制结构有了面向对象的概念，但尚未形成类的概念；1978 年出现的 SMALLTALK-76 有了类的概念；而 1981 年出现的 SMALLTALK-80 则进一步发展和完善了面向对象的概念。SMALLTALK-80 是由美国 Xerox Learning Research Group 研制的，至今仍然是一种广泛应用的面向对象的程序设计语言。

20 世纪 80 年代初就出现了 C++的早期版本“带类的 C”，即 OBJECTIVE C。直到 1983

年，C++的名字才正式确定下来。1986 年，Bjarne Stroustrup 在美国 AT&T 的贝尔实验室开发了 C++。C++的出现要归功于 C 的成功，因此，C 作为 C++的子集在 C++予以保留。此外，C++还吸收了其他语言的一些特点。例如，C 的前身 BCPL 的注释符号“//”再次引入到 C++中，SIMULA 的派生类和虚函数、ALGOL 的运算符重载和变量自由声明等也被引入到 C++中。

20 世纪 90 年代初，计算机网络的普及和应用导致了一种新的面向对象的语言 JAVA 的出现。JAVA 是在 C++的基础上，去掉了 C++的某些语法成分形成的，最初用于家用电器的交互式软件设计。1995 年，随着全球互联网的风靡，JAVA 才成为网络信息浏览的热门开发语言。

1.3.2 面向对象的程序设计

面向对象的程序设计要经历系统分析、系统设计、对象设计与对象实现等四个阶段。由于面向对象的软件工程支撑整个软件生命周期，因此，系统分析、系统设计、对象设计与对象实现等阶段之间的界限不是十分明显。在整个软件生命周期中，可以通过面向对象的方法不断地扩充和完善早期建立的对象模型，这种早期模型和晚期模型的一致性，使面向对象的技术能更好地支持快速原型法。

系统分析阶段的主要任务是理解应用系统的定义及建立应用系统的对象模型。这一阶段需要分析和发现对象，定义对象的属性和内部状态变换，定义外部事件和对象之间的联系及约束，最终建立应用系统的对象模型、动态模型及功能模型。

系统设计阶段的主要任务是确立对象模型的实现方法。这一阶段需要将系统分解为子系统，确定应用系统固有的并发性，分配处理器的任务，选择数据存储的管理手段，处理全局资源的访问，确立应用系统的边界等等。

对象设计阶段的主要任务是将系统分析阶段建立的三种模型转换成类。这一阶段需要将应用系统的对象模型转换为类的属性，将动态模型转换为类的方法，将功能模型转换为主控模块或子控模块。在转换对象模型时，主要要考虑对象关联的实现方法，例如，要增加新的属性或者增添新的类；在转换动态模型时，最好选用事件驱动的运行环境，因顺序执行的运行环境难于实现并发处理；在转换功能模型时，可以用顺序执行或事件驱动的方式实现流程控制。

对象实现阶段的主要任务是用面向对象的程序设计语言实现对象设计阶段定义的类。这一阶段涉及选择合适的面向对象的程序设计语言及开发环境，定义类的属性及其方法，编写主控模块及子控模块，针对具体语言进行性能调整等。当面向对象的语言不提供多继承机制时，要考虑多继承类转换为单继承类的策略。

1.4　面向对象的基本概念

1.4.1　对象、类及类型

对象即现实世界中的各种具体的或抽象的“事物”，整个世界就是由各种简单或复杂的对象构成。形形色色的事物相互有别，使得我们能够对事物进行分类并以某种形式进行描述；对象的分类及其描述即形成了对象的类型。

现实世界中的事物用“特征”和“行为”描述，与之对应的对象则用“属性”和“操作”来表述；在面向对象的程序设计语言中，“特征”和“属性”用“数据成员”表示，而“行为”和“操作”则用“函数成员”表示。

例如，“体重”是一个简单对象，其“数据成员”通常定义为简单变量 weight，其“函数成员”则是+、-等运算以及 sin(double)等数学运算函数。

复杂对象由若干简单对象构成，人们所说的对象一般指复杂对象。例如，“人”是由“姓名”、“体重”等简单对象构成的复杂对象，其“数据成员”即为“姓名”和“体重”等，其“函数成员”则是“吃”、“穿”、“住”、“行”等涉及人类生活的函数。

简单对象的类型用简单类型表示,复杂对象的类型用结构或联合体表示。在面向对象的程序设计语言 C++中，表示简单对象的类型有 char、int、long、double 等，表示复杂对象的类型则是用 struct、union 和 class 等定义的结构或联合体。

假如将人的对象类型取名为人类 HUMANKIND，则人类就具有“姓名”、“体重”等属性，并且具有“吃”、“穿”、“住”、“行”等操作。在 C++中，人类 HUMANKIND 可以用如下结构体描述：

```
struct   HUMANKIND{
    char    *name;
    double weight;
public:
    void    eat( );
    void    wear( );
    void    reside( );
    void    travel( );
};
```

一个对象也称为类的一个实例，即该类的一个变量或常量。例如，人类 HUMANKIND 的一个实例即某个人，相应于该人的变量 person 定义如下：

```
struct    HUMANKIND    person;
```

对象 person 是类型 HUMANKIND 的一个实例，对象 person 在定义的同时一定会被初始化。例如，person 的数据成员 name 被初始化为空指针 0，person 的数据成员 weight 被初始化为 0。

对象的初始状态由对象的所有数据成员的初始值构成。例如，person 的初始状态为{ 0，0}；给对象 person 取名“赵薇”后，person 的状态变为{ “赵薇”，0 }；给对象 person 称体重 50 公斤后，person 的状态变为{ “赵薇”，50 }。

1.4.2 封装与交互

现实世界中的事物在展现其外在“特征”和“行为”的同时，也给人们留下了无数的有待解答的问题：这些事物还有哪些“特征”和“行为”没有表现出来？已经表现出来的“行为”究竟是怎样产生的？这些事物就像有一层“躯壳”(或称“封装”)，令人无法打开“躯壳”并了解其中的奥妙。

“封装”对于面向对象的程序设计而言，具有十分重要的意义：

● 封装”为对象定义了一个边界，对象的私有“属性”和“操作”被隔离起来，在建立系统的对象模型时，只需考虑对象的公共“属性”和“操作”，大大降低了系统模型的复杂程度。

●“封装”为对象定义了一个接口，对象的公共“属性”和“操作”通过该接口对外开放，其他对象借此接口访问对象的公共“属性”和“操作”。在接口不变的情况下，重新定义对象不会影响其他对象的访问，给对象设计带来了较大的灵活性。

●“封装”屏蔽了对象“操作”的细节，从而能够保证核心算法不被泄露，有助于保护知识产权。

C++通过 private 定义对象私有的“属性”和“操作”，通过 protected 定义受保护的“属性”和“操作”,通过 public 定义公共的“属性”和“操作”。为了屏蔽对象“操作”的实现细节，C++允许在一个程序模块里定义对象的类型，而在另一个模块里定义函数成员的函数体。

对象交互可以在不同的对象之间进行，也可以在同一对象的内部进行。一次交互是指一个对象向另一个对象发出请求，要求另一个对象完成某种“操作”。对象之间发出的请求称为“消息”，消息是一个对象要求另一个对象执行某个操作的规格说明。对象的交互有以下两种形式。

● 直接交互：一对象直接调用另一对象的公共“操作”，这种方式在没有消息管理机制的单任务操作系统环境下用得最多。

● 间接交互：一对象发送消息到消息队列中，由操作系统识别消息应该由哪个对象接受，然后调用相应对象的相关“操作”，这种方式在多任务操作系统环境下用得最多。

发出请求的对象称为消息发送者，接受请求的对象称为消息的接受者。同一个对象可以接受不同规格的消息，并据此调用不同的“操作”作出不同响应。相同规格的消息可以传给不同的对象，接受消息的对象作出的响应可以不同。消息的发送者在发送消息时并不一定需要知道接受者是否存在，而消息的接受者可以响应也可以不响应消息。

消息的内容一般包括接受者的对象标识、请求接受者执行的操作以及执行该操作所需的参数等。执行一个操作可以不要参数，也可以要一个或多个参数。

对象标识是指一个对象能区别于其他对象的唯一性标志。常见的对象标识有对象名称或由计算机自动产生的一个整数等。对象标识几乎都是由计算机自动产生的，不需要专门

为对象定义对象标识。在C++中，对象的地址被自动用来当做对象标识。

1.4.2 重载与多态

重载是多态的一种特例，重载又称为编译时多态，多态则特指运行时多态。所谓编译时多态是指：在若干同名的重载函数中，编译程序能根据重载函数的参数差异，确定应用程序到底要调用哪个重载函数。所谓运行时多态是指：在程序运行的过程中，执行程序根据调用虚函数的对象类型将虚函数映射为相应类型的函数成员，从而使不同类型的对象在调用虚函数时表现出不同的对象行为。除非特别声明，以后出现的多态均指运行时多态。简单对象的“操作”仅允许重载而不允许多态，复杂对象的操作既可以重载也可以多态。

对象的不同“操作”应取不同的操作名，必要时也可以取相同的操作名。倘若几个“操作”共用同一个名字，则这些“操作”的参数个数或类型必然不同，这种“现象”称为操作的重载。

对于简单对象，例如，对于int类型的对象3和5，可以进行加法和取正两种操作：

(1) 3 + 5

(2) +3

两种操作虽然都取名为“+”，但两种操作的参数（操作数）个数不同。由于两种操作涉及的对象都是简单对象，因此，可以断定上述两种操作对“+”进行了重载，重载使得两种操作的名称相同但完成的功能不同。

在C++中，参数为简单类型的函数只能进行重载。例如，程序可以定义int add(int, int)和double add(int, double)两个重载函数，然后通过语句add(5, 6.5)调用上述重载函数，由于实参5和6.5的类型分别是int和double类型，因此，编译程序可以断定程序调用的是double add(int, double)。

对于两个不同类型的复杂对象的函数成员，当它们的参数个数和类型完全一致时，编译程序必须根据对象的类型确定要调用的函数成员。例如，对于类型为点POINT的对象p和类型为圆CIRCLE的对象c，它们都有参数个数及类型完全相同的“无参”函数成员draw( )，draw( )用于绘制对象点p或圆c所代表的图形。如果用p.draw( )调用函数成员draw( )，很显然调用是通过对象p发出的，则函数draw( )绘制的图形一定是一个点；如果用c.draw( )调用函数成员draw( )，很显然调用是通过对象c发出的，则函数draw( )绘制的图形一定是一个圆。

函数成员draw( )初看起来是一个无参函数，实际上它有一个隐含的参数this，它是一个指向p或c的指针。对于对象p的函数成员draw( )，隐含参数this的类型为POINT * const；对于对象c的函数成员draw( )，隐含参数this的类型为CIRCLE * const。编译程序就是根据隐含参数的类型来确定要调用哪一个重载函数成员的。

然而，当POINT和CIRCLE具有父子继承关系时，指向父类POINT的对象的指针可以指向子类CIRCLE的对象。例如，如下定义：

```
POINT   p, *q;
CIRCLE  c;
```

对父类指针q而言，赋值q=&p和q=&c都是允许的。于是，对编译程序而言，无法在某

个时刻确定q所指的对象是p还是q，进而就会产生这样一个问题：当使用q->draw( )调用函数成员draw（ ）时，调用的是对象p的draw( )还是对象c的draw( )?

指针q实际指向的对象只能在运行时确定，调用q->draw( )应该根据q指向的对象来绘制点或者圆，这就是draw( )所应该表现出来的运行时多态。C++通过将draw( )声明为虚函数成员来实现运行时多态。

1.4.3 继承与抽象

每个小孩都从父母身上继承了一些特性和行为，以面向对象的观点来看，父亲和母亲称为基类对象，小孩则称为派生类对象。派生类对象可以有两个或多个基类对象，这种继承称为多继承；派生类对象也可以只有一个基类对象，这种继承称为单继承。

多继承是现实世界中普遍存在的现象。随着“克隆”技术的出现，单继承现象也会越来越普遍。JAVA是支持单继承的面向对象的程序设计语言，而C++是支持多继承的面向对象的程序设计语言，很显然，C++描述对象的能力比JAVA强。

多重继承是指从祖先基类经过若干代派生最终产生的派生类。例如，多重继承使得小孩既像其父亲、母亲，还像其爷爷、奶奶、外公和外婆。JAVA有一个始祖基类，它是任何类及派生类的基类；在C++中，不存在这样的始祖基类。

派生类对象可以继承基类对象的“属性”和“操作”，同时也可以改变、增加新的“属性”和“操作”。如果派生类对象继承的“操作”不变，就没有必要为派生类对象编写相应函数成员，因此，面向对象的程序设计减少了数据冗余，提高了软件单元的可重用性。

基类的“属性”和“操作”都是可被继承的内容。从继承的、修改的和增加的“属性”和“操作”等内容来看，继承可以分为取代继承、包含继承、受限继承和异化继承。

取代继承：派生类对象完整地继承了所有基类的所有“属性”和“操作”，并且没有修改、增加新的“属性”和“操作”。例如，“克隆”技术产生的后代同其亲本完全一样，这种继承就属于取代继承。取代继承适用于单继承，这种继承没有产生变种，几乎没有应用价值。

包含继承：派生类对象完整地继承了所有基类的所有“属性”和“操作”，并增加了新的“属性”和“操作”。例如，给“火车头”增加“车厢”，使之变成“客运火车”。包含继承适用于单继承和多继承，具有较大的应用价值。

受限继承：派生类对象部分地继承了基类的“属性”和“操作”，并且没有增加新的“属性”和“操作”。受限继承是包含继承的反例，适用于单继承和多继承。

异化继承：派生类对象继承了基类的“属性”和“操作”，并且对原有“属性”和“操作”进行了修改。严格地说，人类生育过程中的继承是一种受限的和异化的继承，后代染色体的数目通常不会改变，但染色体的分子结构通常会发生改变，从而导致后代的血型可能和双亲不同。如果异化的“属性”或“操作”超过了50%，则应考虑解除基类和派生类之间的继承关系。

包含继承是最普遍的一种继承形式，基类的“属性”和“操作”是各派生类的共同特性，因此，基类是对派生类的一种抽象。在多重继承的情形下，祖先基类或始祖基类是抽象级别最高的基类。