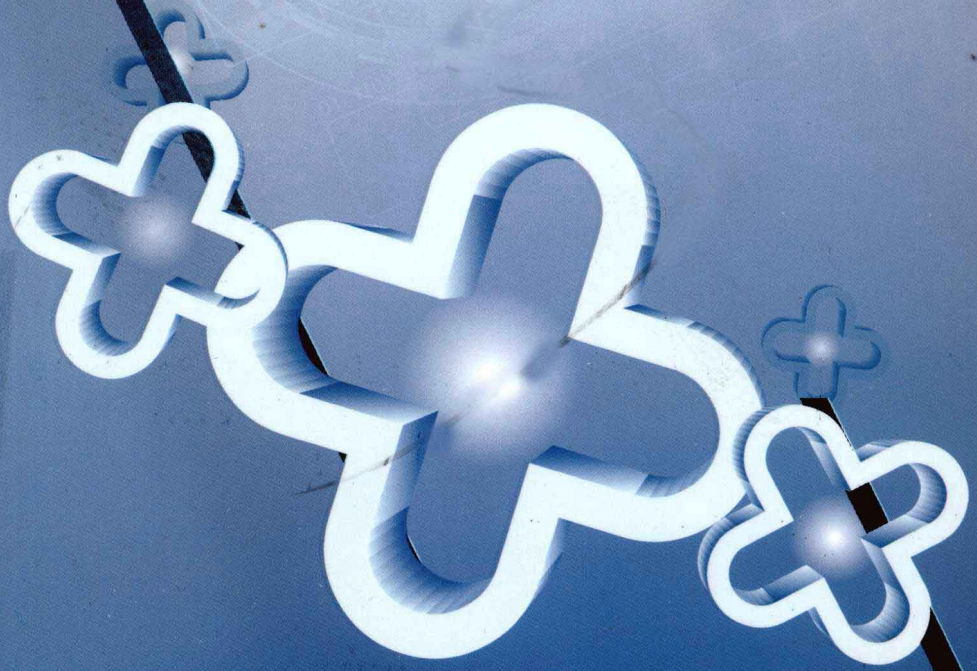
高等学校应用型“十二五”规划教材 • 计算机类

精品课程配套教材

C/C++语言程序设计

龚尚福　贾澎涛　主编



西安电子科技大学出版社
http://www.xduph.com

高等学校应用型“十二五”规划教材 计算机类

★ 精品课程配套教材

C/C++语言程序设计

龚尚福　贾澎涛　主编

西安电子科技大学出版社

前　言

随着科学技术的发展和人类社会的进步，计算机语言的应用日益渗透到国防、工业、农业、企事业和人们日常生活的各个领域，其中 C 语言是应用面最广的语言之一。C 语言具有简洁、紧凑、灵活、实用、高效、可移植性好等优点，也是高等院校讲授程序设计课程的首选语言。随着程序设计方法的改进，面向对象的程序设计已成为主流，C++语言也应运而生并很快流行起来。因此，学习并熟练掌握 C 和 C++就显得尤为重要。

本教材的理念是："以实例驱动教学"，不能削弱基础知识和基本语法；既需要强调基础，更需要加强实践应用。我们的目的是：让学生在掌握 C/C++语法知识的基础上更好地学习程序设计的思维方式，培养学生理论联系实际、触类旁通的能力。

本教材主要具有以下特点：

(1) 注重基础性、系统性和实用性。

编者结合长期教学实践，力求在程序设计语言教学上做到循序渐进、深入浅出地阐述程序设计的方法和过程。

(2) 以原理为主线，以案例为引导，以掌握、应用为目的。

根据本科生的培养要求，本教材侧重于对学生程序设计的分析、设计、开发、调试和应用能力等方面的培养。在介绍基本语法的基础上，以大量的应用实例加以引导和启发，并通过加强习题练习和实验的训练，使学生在掌握基本语法的基础上，具有一定的程序设计思维和实践能力。

(3) 重点突出，难点分散，由浅入深。

本教材遵循面向应用的教学目标，重点突出，难点分散。例如，在指针概念的讲解上，首先将指针看做是一种数据类型，在第 2 章引入。其余相关的指针概念和用法分散到各章节讲授，由浅入深、循序渐进，让学生逐步接受指针的概念和用法。对内容的选取、概念的引入，以及文字叙述、例题和习题的设计等都进行了精心的策划和安排。

(4) 以实例为主，驱动教学。

本教材引入了俄罗斯方块游戏案例编程，该案例贯穿了面向对象程序设计的整个过程。该案例应用了前面所学过的几乎全部的语法知识，是编者精心设计的一个案例，更具实践性和趣味性，寓教于乐。

(5) 全书风格良好，适用面广。

本教材每章开头都有问题的引入和本章的主要内容摘要，书中还提供了大量的例题，每章末尾都附有一定数量的习题。全书文字叙述简练，风格统一，图文并茂，且所给例题程序均在机器上调试通过。

全书由 14 章和两个附录组成，可以分为两大部分。其中，第一部分是 C 语言的基础知识和程序设计方法，第二部分是 C++ 语言面向对象的程序设计方法及应用。每一部分内容又分成不同的模块，如下所述：

第一部分：C 语言的基础知识(1～3 章)；C 语言的主要内容(4～6 章，第 8 章)；C 语言

的编译(第 7 章)。

第二部分：C++的非面向对象特性(第 9 章)；C++的主要特性(10～12 章)；模板(第 13 章)；异常处理(第 14 章)。

建议读者独立完成每章末尾所附的习题，以便复习及检查学习效果。

本教材适用面较宽，为了能适应各类专业的不同要求，C 和 C++之间既相互配合又自成体系，便于读者删减使用。

本教材由西安科技大学龚尚福、贾澎涛任主编，西安科技大学梁荣、史晓楠、龚星宇参编。其中，龚尚福编写了第 1 章，梁荣编写了第 3、4、6、7 章，史晓楠编写了第 2、5、13 章，贾澎涛编写了第 10～12 章，龚星宇编写了第 8、9、14 章，龚尚福、贾澎涛审读了全书内容。

在本教材的编写过程中参考了大量同类专著和教材，我们也尽可能地将这些文献列于书后。对于因疏漏而未能列出的参考文献，在此特向其作者表示歉意。同时也对所有参考文献的作者们表示诚挚的感谢。在本教材的编写过程中，还得到了西安科技大学领导和教务处有关同志的大力支持，在此也表示衷心的感谢。

由于时间仓促，加之编者水平有限，书中不妥或错误之处敬请读者批评、指正。

为了帮助读者学习本教材，还配套编写了《C/C++语言程序设计同步进阶经典 100 例与习题指导》(李军民主编，西安电子科技大学出版社出版)一书，其中提供了本教材中各章习题的答案与上机实习指导，供读者选用。

编　者

2011 年 12 月

目　录

第1章 概 述

教学目标

※ 了解计算机语言的基本概念。
※ 了解算法与流程的基本概念。
※ 掌握程序设计的特点及其一般方法。
※ 了解 C/C++ 语言的发展及其特点。
※ 掌握 Microsoft Visual C++ 6.0 集成环境。

1.1 计算机语言及其发展

计算机是一个有用的工具，它能做许多事情，例如进行矩阵计算、方程求解、辅助设计等。在通过计算机解决某一个问题之前，必须先把求解问题的步骤描述出来，这个步骤称为算法。例如，对一个一元二次方程，若求其实数解，算法应为：

(1) 计算方程的判别式；

(2) 如判别式小于零，则输出方程没有实根的信息；

(3) 否则，计算方程的实根，并输出计算结果。

但是，这个算法不能直接输入到计算机，因为用这种自然语言表达的算法，计算机并不理解。正像人类之间通过语言进行沟通一样，要计算机做事，就必须使用计算机能够理解的语言，称之为计算机语言。将这个算法用某种特定的计算机语言表达出来，并且输入到计算机里，通过计算机编译系统编译后运行，才能得到计算机处理的结果。把算法通过特定语言进行描述的过程称为计算机编程(或程序设计)。

自从有了计算机，也就有了计算机编程语言。计算机语言的发展经历了三个阶段：

(1) 机器语言阶段。最初的计算机编程语言是所谓的机器语言(也称为第一代语言)，即直接使用机器代码编程。机器语言即机器指令的集合。每种计算机都有自己的指令集合，计算机能直接执行用机器语言所编的程序。机器语言包括指令系统、数据类型、通道指令、中断字、屏蔽字、控制寄存器的信息等。机器语言是计算机能理解和执行的唯一语言。机种不同，其机器语言组合方式也不一样。因此，同一个题目到不同的计算机上计算时，必须编写不同机器语言的程序。机器语言是最低级的语言。

(2) 汇编语言阶段。由于机器语言指令是用多位二进制数表示的，用机器语言编程必然很繁琐，非常消耗精力和时间，难记忆，易出错，并且难以检查程序和调试程序，工作效率低。例如，字母 A 表示为 1010，数字 9 表示为 1001；机器语言的加法指令码有三种形式，

既要考虑进位、符号，也要考虑溢出等情况，要用加法指令，就必须分别记忆。为了提高编程效率，人们引入了助记符，例如，加法用助记符 ADD 表示，减法用助记符 SUB 表示等。这就出现了所谓汇编语言(也称为第二代语言)。汇编语言同机器语言相比，并没有本质的区别，只不过是将机器指令用助记符号代替，但这已是很大的进步，它提高了编程效率，改进了程序的可读性和可维护性。直到今天，仍然有人在用汇编语言编程。但是汇编语言在运行之前，还需要一个专门的翻译程序(称为汇编程序)将其翻译为机器语言，因此实现同样的功能，汇编语言编写的程序执行效率相对于机器语言来说降低了。

(3) 高级语言阶段。虽然汇编语言较机器语言已有很大的改进，但仍有两个主要缺点：一是涉及太多的细节；二是与具体的计算机结构相关。所以，汇编语言也是低级语言，被称为面向机器的语言。为了进一步提高编程效率，改进程序的可读性、可维护性，20 世纪 50 年代以来相继出现了许多种类的高级计算机编程语言(也称为第三代语言)，例如：Fortran、Basic、Pascal、Java、C 和 C++ 等，其中 C 和 C++ 是当今最流行的高级计算机程序设计语言。

高级语言比低级语言更加抽象、简洁，它具有以下特点：

① 一条高级语言的指令相当于几条机器语言的指令；

② 用高级语言编写的程序同自然英语语言非常接近，易于学习；

③ 用高级语言编写程序并不需要熟悉计算机的硬件知识。

同汇编语言类似，高级语言也需要专门的翻译程序(称为编译器或解释器)将它翻译成机器语言后才能运行。因此，实现同样的功能，用高级语言编写的程序执行效率相对于机器语言和汇编语言来说是最低的。

1.2 算法与流程

1.2.1 算法的概念

程序设计的灵魂是算法，而语言只是形式。可以说计算机语言只是一种工具，用来描述处理问题的方法和步骤。但是只要有正确的算法，可以利用任何一种语言编写程序，使计算机进行工作，得出正确的结果。

所谓“算法”，指为解决一个问题而采取的方法和步骤，或者说是解题步骤的精确描述。对于同一个问题，可以有不同的解题方法与步骤。例如求 $1+2+3+\cdots+100$，即 $\sum_{n=1}^{100} n = 1+2+3+\cdots+100$，就有不同的方法，有人先进行 1 + 2，再加 3，再加 4，一直加到 100，得到结果 5050。而有的人采取另外的方法：$\sum_{n=1}^{100} n = 100+(99+1)+(98+2)+\cdots+(51+49)+50 = 50+50\times 100 = 5050$。显然，对心算来说，后者比前者更容易得出正确的结果。

算法有优劣，一般而言，应当选择简单的、运算步骤少的，运算快、内存开销小的算

法(算法的时空效率)。算法应具备有穷性、确定性、有效性、有零个或多个输入(即：可以没有输入，也可以有输入)、有一个或多个输出(即算法必须得到结果)的特性。

1.2.2 算法的表示形式

为了表示一个算法，可以用不同的方法。常用的算法表示方法有自然语言、传统流程图、结构化流程图(N-S 流程图)、伪代码、计算机语言等。本书将重点讲述传统流程图和N-S 流程图。

1. 传统流程图

流程图即用一些约定的几何图形来描述算法的组合图。用某种图框表示某种操作，用箭头表示算法流程。图 1.1 所示的图例就是美国标准化协会 ANSI 规定的一些常用的流程图符号，已为世界各国程序工作者普遍采用。

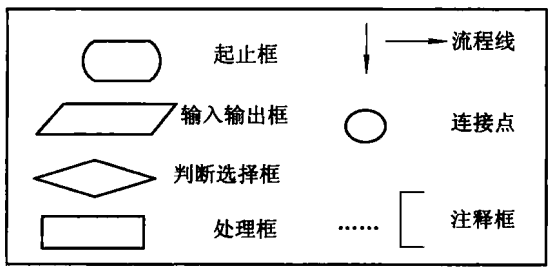


图 1.1 常用的流程图符号

- 起止框：表示算法的开始和结束。一般内部只写“开始”或“结束”。
- 输入输出框：表示算法请求输入需要的数据或算法将某些结果输出。一般内部常常填写“输入…”，“打印/显示…”。
- 判断选择框：对一个给定条件进行判断，根据给定的条件是否成立来决定如何执行其后的操作。它有一个入口，两个出口。
- 处理框：表示算法的某个处理步骤，一般内部常常填写赋值操作。
- 连接点：用于将画在不同地方的流程线连接起来。同一个编号的点是相互连接在一起的，实际上同一编号的点是同一个点，只是画不下才分开来画。使用连接点，还可以避免流程线的交叉或过长，使流程图更加清晰。
- 注释框：注释框不是流程图中必须的部分，不反映流程和操作，它只是对流程图中某些框的操作做必要的补充说明，以帮助阅读流程图的人更好地理解流程图的作用。

例 1.1 求 5!。

① 算法分析：实际上是在做 $1\times2\times3\times4\times5$ 的运算。

② 算法步骤可以分为：

步骤 1：设变量 p，被乘数，p=1；

步骤 2：设变量 i，代表乘数，i=2；

步骤 3：使 $p\times i$，乘积放在被乘数变量 p 中，可表示为：$p\times i\Rightarrow p$；

步骤 4：使 i 的值加 1，即 $i+1\Rightarrow i$；

步骤 5：如果 i 不大于 5，返回重新执行步骤 3 以及其后的步骤 4、步骤 5，否则，算法

结束。最后得到的 p 就是 5! 的值。

③ 绘制流程图，如图 1.2 所示。

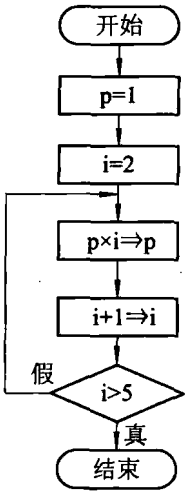


图 1.2 求 5!的流程图

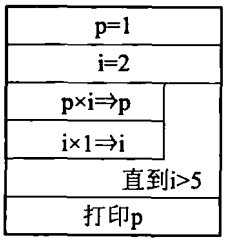


图 1.3 求 5!的 N-S 流程图

2. N-S 流程图

基本结构的顺序组合可以表示任何复杂的算法结构，于是基本结构之间的流程线就属于多余的了，于是美国学者 I.Nasii 和 B.shneiderman 于 1973 年提出了一种新的流程图形式：将全部算法写在一个矩形框内，完全去掉了带箭头的流程线。这种流程图称为 N-S 结构化流程图，也称盒图。

例 1.2 将求 5! 的算法用 N-S 图表示。

5!的 N-S 流程图如图 1.3 所示。

N-S 图比文字描述直观、形象，便于理解，比传统流程图紧凑易画，尤其是它废除了流程线，整个算法是由各个基本结构按顺序组成的。N-S 图的上下顺序就是执行时的顺序，写算法和看算法都是从上到下，十分方便。用 N-S 图表示的算法都是结构化算法，它由几种基本结构顺序组成，基本结构之间不存在跳转，流程的转移只存在于一个基本结构范围之内(如循环中流程的跳转)。N-S 图不能表示非结构化算法，而且当问题很复杂时，N-S 图可能很大。在后续章节程序算法描述中将采用传统流程图，如果读者感兴趣的话，可以将它们转化成为 N-S 流程图。

1.3 程序设计方法

为解决一个问题，用计算机语言编写计算机程序的过程，称为程序设计。程序设计需要有一定的方法来指导，有些问题算法比较简单，可以直观得到，如前面提到的一元二次方程求解的算法；对于有些较为复杂的问题，则需要对问题进行分解，如字符串的处理就要复杂一些，涉及到字符串的合并、拷贝、比较等，就不是一个简单算法能够表达的。对要解决的问题进行抽象和分解，对程序进行组织与设计，使得程序的可维护性、可读性、稳定性、效率等更好，是程序设计方法研究的问题。目前，有两种重要的程序设计方法：

结构化的程序设计和面向对象的程序设计，下面分别进行简单的介绍。

1.3.1 结构化程序设计方法

1. 结构化程序设计的基本概念

结构化程序设计(SP, Structured Programming)方法是由 E.Dijkstra 等人于 1972 年提出来的，它建立在 Bohm、Jacopini 证明的结构定理的基础上。结构定理指出：任何程序逻辑都可以用顺序、选择和循环三种基本结构来表示，如图 1.4 所示。在结构定理的基础上，Dijkstra 主张避免使用 goto 语句(goto 语句会破坏这三种结构形式)，而仅仅用上述三种基本结构反复嵌套来构造程序。在这一思想指导下，进行程序设计时，可以用所谓“自顶向下，逐步求精”的方式对问题进行分解。

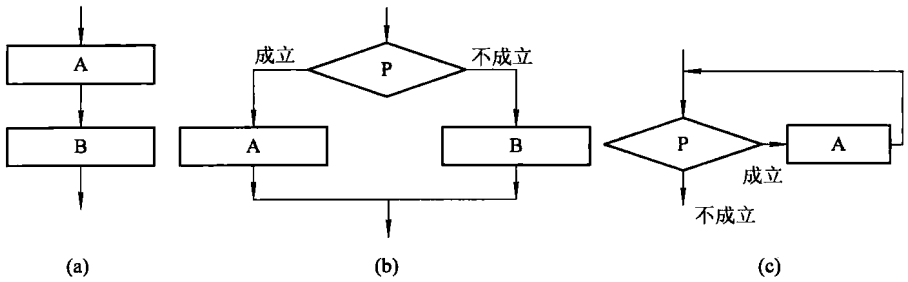


图 1.4 程序的顺序、选择和循环三种基本结构

(a) 顺序结构；(b) 选择结构；(c) 循环结构

由于用结构化方法设计的程序只存在三种基本结构，程序代码的空间顺序和程序执行的时间顺序基本一致，因此程序结构比较清晰。一个结构化程序应符合以下标准：

(1) 程序仅由顺序结构、选择结构和循环结构三种基本结构组成，基本结构可以嵌套。

(2) 每种基本结构都只有一个入口和一个出口，即一端进，一端出。这样的结构置于其他结构之间时，程序的执行顺序必然是从前一结构的出口到本结构的入口，经本结构内部的操作，到达本结构的唯一出口，体现出流水化特点。

(3) 程序中没有死循环(不能结束的循环)和死语句(程序中永远执行不到的语句)。

2. 结构化程序设计方法遵循的原则

结构化程序设计强调程序设计风格和程序结构的规范化，提倡清晰的流程结构。如果面临一个复杂的问题，是难以很快写出一个层次分明、结构清晰、算法正确的程序的。结构化程序设计方法的基本思路是：把一个复杂问题的求解过程分阶段(流水作业)进行，每个阶段处理的问题都控制在人们容易理解和处理的范围内。具体来说，可采取以下方法保证得到结构化的程序。

(1) 自顶向下，逐步求精。这种方法的特点是抓住整个问题的本质特性，采用自顶而下逐层分解的方法，对问题进行抽象，划分出不同的模块，形成不同的层次概念。把一个较大的复杂问题分解成若干相对独立而又简单的小问题，只要解决了这些小问题，整个问题也就解决了。实际上，其中每一个小问题又可进一步分解为若干更小的问题，一直重复下去，直到每一个小问题足够简单，便于编程为止。这种方法便于检查算法的正确性，在上

一层正确的情况下向下细分，如果每层都没有问题，整个算法就是正确的。由于每层细化时都相对比较简单，容易保证算法的正确性。检查时也是由上向下逐层进行，思路清晰，既严谨又方便。

(2) 模块化设计。模块化设计是把复杂的算法或程序，分解成若干相对独立、功能单一，甚至可供其它程序调用的模块。在引入结构化程序设计之后，这些模块不仅与通常所说的子算法、子程序或子过程有着相似的概念，是一种可供调用、相对独立的程序段，而且必须是由三种基本结构组成的。整个系统犹如积木一般，由各个模块组合而成。各模块之间相互独立，每个模块可以独立地进行分析、设计、编程、调试、修改和扩充，而不影响其他模块或整个程序的结构。

进行模块化设计时，注意在不同模块中提取功能相同的子模块，作为一个独立的子模块。这样可以缩短程序，提高模块的复用率。设计模块时要尽量减小模块间的耦合度(模块间的相互依赖性)，增大内聚度(模块内各成分的相互依赖性)。耦合度越小，模块相互间的独立性就越大；内聚度越大，模块内部各成分间的联系就越紧密，其功能也就越强。

模块化结构不仅使复杂的程序设计得以简化，开发周期得以缩短，节省费用，提高了软件的质量，还可有效地防止模块间错误的扩张，增强整个系统的稳定性与可靠性；同时，也使程序结构具备层次分明、条理清晰、便于组装、易于维护等特点。

(3) 程序结构化。所谓程序结构化，是指利用高级语言提供的相关语句实现三种基本结构，每个基本结构具有唯一的入口和出口，整个程序由三种基本结构组成，程序中不使用goto之类的语句。

3. 结构化程序设计过程

结构化程序设计的过程分为三个基本步骤：分析问题(Question)、设计算法(Algorithm)、编写程序(Program)，简称QAP方法。

第一步：分析问题。对问题进行定义与分析。① 确定要产生的数据(称为输出)，定义表示输出的变量。② 确定需要进行输入的数据(称为输入)，定义表示输入的变量。③ 研制一种算法，从有限步的输入中获取输出。这种算法定义为结构化的顺序操作，以便在有限步内解决问题。就数字问题而言，这种算法包括获取输出的计算；但对非数字问题来说，这种算法可能包括许多文本和图像处理操作。

第二步：设计算法。设计程序的轮廓(结构)，并画出程序的流程图。① 对一个简单的程序来说，通过列出程序顺序执行的动作，便可直接画出程序的流程。② 对于复杂的程序来说，使用自上而下的设计方法，把程序分割为一系列的模块，形成一张结构图。每一个模块完成一项任务，再对每一项任务进行逐步求精，描述这一任务中的全部细节，最终将结构图转变成流程图。

第三步：编写程序。采用一种计算机语言(如使用C语言)实现算法编程。① 编写程序，即将前面步骤中描述性的语言转换成C语句。② 编辑程序，即测试和调试程序。③ 获取结果，即获取程序运行结果。

结构化的程序设计虽然是广泛使用的一种程序设计方法，但也有一些缺点：

(1) 恰当的功能分解是结构化程序设计的前提。然而，对于用户需求来讲，变化最大的部分往往就是功能的改进、添加和删除。结构化程序要实现这种功能变化并不容易，有时甚至要重新设计整个程序的结构。

(2) 在结构化程序设计中，数据和对数据的操作(即函数)分离，函数依赖于数据类型的表示。数据的表示一旦发生变化，则与之相关的所有函数均要修改，这就使得程序维护量增大。

(3) 结构化的程序代码复用性较差。通常也就是调用一个函数或使用一个公共的用户定义的数据类型而已。由于数据结构和函数密切相关，使得函数并不具有一般特性。例如，一个求解方程实根的函数不能应用于求解复数的情形。

1.3.2 面向对象程序设计方法

面向对象程序设计是另一种重要的程序设计方法，它能够有效地改进结构化程序设计中存在的问题。面向对象的程序与结构化的程序不同，例如，由 C++ 语言编写的结构化的程序是由一个个的函数组成的，而由 C++ 语言编写的面向对象的程序是由一个个的对象组成的，对象之间通过消息可以相互作用。

在结构化程序设计中，解决某一个问题，就是要确定这个问题能够分解为哪些函数，数据能够分解为哪些基本的类型，如 int(整型)、double(双精度实型)等。也就是说，思考方式是面向机器结构的，而不是面向问题结构的，需要在问题结构和机器结构之间建立联系。而面向对象程序设计方法的思考方式是面向问题结构的，它认为现实世界是由一个个对象组成的，在解决某个问题时，先要确定这个问题是由哪些对象组成的。

客观世界中任何一个事物都可以看做一个对象。或者说，客观世界是由千千万万个对象组成的，它们之间通过一定的渠道相互联系。例如一所学校是一个对象，一个班级也是一个对象。实际生活中，人们往往在一个对象中进行活动，或者说对象是进行活动的基本单位。例如在一个班级对象中，学生上课、休息、开会和参加文娱活动等。作为对象，它应该至少具备两个因素：一是从事活动的主体，例如班级中的若干名学生；二是活动的内容，如上课、开会等。

从计算机的角度看，一个对象应该包括两个因素：一是数据，相当于班级中学生的属性；二是需要进行的操作(函数)，相当于学生进行的各种活动。对象就是一个包含数据以及与这些数据有关的操作集合。图 1.5 表示了一个对象是由数据和操作组成的集合。

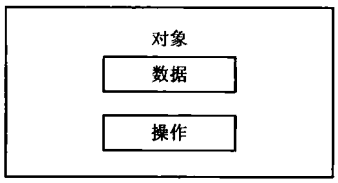


图 1.5 对象的组成

单个对象的用处并不大，程序往往通过对象之间的交互作用，获得更高级的功能和更复杂的行为。例如，一辆汽车停在路边，本身并不能动作，仅当另一个对象(一个司机)和它交互(开车)时才有用。对象之间的这种相互作用需通过消息进行通信(消息需要有足够的信息)。当对象 A 要执行对象 B 的方法时，对象 A 发送一个消息到对象 B，通知接收对象 B 要它做什么。

消息由三个部分组成：① 接受消息的对象；② 要执行的函数的名字；③ 函数需要的参数。例如，教师要求学生张三完成 5 的阶乘计算这一任务，则这个教师会说："张三，5 的阶乘是多少？"这句话就是教师向学生张三发出的消息。其中，教师是发送消息的对象，张三是接受消息的对象，教师调用张三计算阶乘的函数并发送了参数 5。

面向对象的程序设计有三个主要特性，它们是：封装、继承和多态，下面先对这几个特性作简单的介绍，具体内容将在后续章节中详细叙述。

1. 封装

在现实世界中，常常有许多相同类型的对象。例如，李四的汽车只是世界上许多汽车中的一个。如果把汽车看做一个大类，那么李四的汽车只是汽车对象类中的一个实例。汽车对象都有相同的数据和对数据的操作行为，但是每一辆汽车的数据又是独立的。根据这个事实，制造商制造汽车时，用相同的蓝图制造许多汽车。在面向对象的程序设计中，称这个蓝图为类。也就是说，类是定义某种对象共同的数据和操作的蓝图或原型。在 C++ 中，封装是通过类来实现的。数据成员和成员函数可以是公有的或私有的。公有的数据成员和成员函数能够被其他的类访问。如果一个成员函数是私有的，它仅能被该类的其他成员函数访问，而私有的数据成员仅能被该类的成员函数访问。因而，它们被封装在类的作用域内。封装是一个有用的机制，具体表现为：① 可以保护未经许可的访问；② 可使信息局部化。

2. 继承

继承在现实生活中很容易理解，例如，我们每个人都从我们的父母身上继承了一些特性，如血型、肤色、毛发的颜色等等。以面向对象程序设计的观点看，继承所表达的是对象类之间相关的关系，这种关系使得某类对象可以继承另外一类对象的特征和能力。再以哺乳动物猫为例，波斯猫和安哥拉猫都是猫的一种，它们都继承了猫科动物的所有特性，但又有自己的特征。用面向对象的术语来说，它们都是猫类的子类(或派生类)，而猫类是它们的父类(基类或超类)。它们的关系如图 1.6 所示。

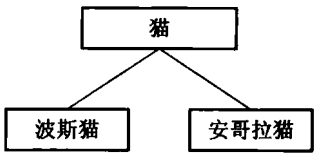


图 1.6　猫的继承关系图

每一个子类继承了父类的数据和操作，但是子类并不仅仅局限于父类的数据和操作，还可以扩充自己的内容。继承的主要益处是可以复用父类的程序代码。

3. 多态

多态是指对于相同的消息，不同的对象具有不同反应的能力。多态在自然语言中应用很多，以动词“关闭”的应用为例来看，同一个“关闭”应用于不同的对象时含义就不相同，如关闭一个门、关闭一个银行账户或关闭一个窗口。精确的含义依赖于执行这种行为的对象。在面向对象的程序设计中，多态意味着不同的对象对同一消息具有不同的解释。

4. 面向对象程序设计过程

面向对象程序设计方法是遵循面向对象方法的基本概念而建立起来的，它的设计过程主要包括面向对象的分析(OOA，Object Oriented Analysis)、面向对象的设计(OOD，Object Oriented Design)、面向对象的实现(OOI，Object Oriented Implementation)三个阶段。

(1) 面向对象的分析(OOA)。OOA 的主要目的就是自上而下地进行分析，即将整个软件系统看做是一个对象，然后将这个大的对象分解成具有语义的对象簇和子对象，同时确定这些对象之间的相互关系。

(2) 面向对象的设计(OOD)。OOD 的任务是将对象及其相互关系进行模型化，建立分类关系，解决问题域中的基本构建。在这个阶段确定对象及其属性，以及影响对象的操作并实现每个对象。

(3) 面向对象的实现(OOI)。OOI 是软件具体功能的实现，是对对象的必要细节加以刻画，是面向对象程序设计由抽象到具体的实现步骤，即最终用面向对象的编程实现建立在

OOA 基础上的模型。

通过上面的介绍可以看出，面向对象的程序设计完全不同于结构化的程序设计。后者是将问题进行分解，然后用许多功能不同的函数来实现，数据与函数是分离的；前者是将问题抽象成许多类，将数据与对数据的操作封装在一起，各个类之间可能存在着继承关系，对象是类的实例，程序是由对象组成的。面向对象的程序设计可以较好地克服结构化程序设计存在的问题，使用得好，可以开发出健壮的、易于扩展和维护的应用程序。

1.4 C/C++ 的特点

C 语言是集汇编语言和高级语言的优点于一身的程序设计语言，既可以用来开发系统软件，也可以用来开发应用软件。

C 语言是从 B 语言衍生而来的，它的原型是 ALGOL 60 语言。1963 年，剑桥大学将 ALGOL 60 语言发展成为 CPL(Combined Programming Language)语言。1967 年，剑桥大学的 Matin Richards 对 CPL 语言进行了简化，于是产生了 BCPL 语言。1970 年，美国贝尔实验室的 Ken Thompson 对 BCPL 进行了修改，提炼出它的精华进而设计出了 B 语言，并用 B 语言编写了第一个 UNIX 操作系统。1973 年，美国贝尔实验室的 D.M.Ritchie 在 B 语言的基础上设计出了一种新的语言，他取了 BCPL 的第二个字母作为这种语言的名字，这就是 C 语言。

为了使 UNIX 操作系统得以推广，1977 年，D.M.Ritchie 发表了不依赖于具体机器系统的 C 语言编译文本《可移植的 C 语言编译程序》。1978 年，B.W.Kernighan 和 D.M.Ritchie 出版了名著《The C Programming Language》，从而使 C 语言成为目前世界上流行最广泛的高级程序设计语言。

随着微型计算机的日益普及，出现了许多 C 语言版本。由于没有统一的标准，使得这些 C 语言之间出现了一些不一致的地方。为了改变这种情况，美国国家标准化协会(ANSI)对 C 语言进行了标准化，于 1983 年颁布了第一个 C 语言标准草案(83 ANSI C)，后来于 1987 年又颁布了另一个 C 语言标准草案(87 ANSI C)。随后的 C 语言标准是在 1999 年颁布并在 2000 年 3 月被 ANSI 采用的 C99，但由于未得到主流编译器厂家的支持，直到 2004 年 C99 也未被广泛使用。

在 C 语言被创建不久，出现了新的概念——面向对象程序设计。但 C 语言并不支持对象，于是美国贝尔实验室的 Bjarne Stroustrup 在 C 语言的基础上弥补了 C 语言存在的一些问题，增加了面向对象的特征，于 1980 年开发出一种既支持过程也支持对象的语言，称为“含类的 C”，1983 年取名为 C++。

1. C 语言的特点

C 语言作为目前世界上使用最广泛的程序设计语言，被许多程序员选择来设计各类程序，它的优势主要取决于 C 语言具有结构化、语言简洁、运算符丰富、移植性强等诸多特点。

(1) 结构化。C 语言是结构化的程序设计语言，其主要结构成分是函数，可通过函数实现不同程序的共享。另外，C 语言具有结构化的控制语句，支持多种循环结构，复合语句也支持程序的结构化。这些特点使得 C 语言层次清晰、结构紧凑，比非结构化的语言更易于使用和维护。