21 世纪高等学校规划教材

C++ 面向对象程序设计简明教程

康 丽 主 编
张瑞玲 郑立华 孙龙清 副主编

中国电力出版社
http://jc.cepp.com.cn

21 世纪高等学校规划教材

C++ 面向对象程序设计简明教程

康 丽 主 编

张瑞玲 郑立华 孙龙清 副主编

王莲芝 程新荣 杨 璐 参 编

徐红升 王玉娟 冀荣华

中国电力出版社

http://jc.cepp.com.cn

内容提要

本书为21世纪高等学校规划教材。

本书介绍了基本的面向对象程序设计思想、概念和技术，使得读者通过阅读、编程和上机实践，掌握面向对象程序设计的规律和步骤。本书注意重点突出C++对一些良好程序设计思想的支持。本书的具体内容包括数据类型、表达式、程序结构、子程序、递归、类和对象、继承和派生、多态与虚函数、模板、输入/输出及异常处理等。

本书具有很强的操作性和实用性，可作为高等院校“C++程序设计”或“程序设计技术”课程教材，也可作为各类培训班“C++程序设计与开发”课程教材，同时本书也是广大C++编程爱好者非常实用的自学参考书籍。

图书在版编目（CIP）数据

C++面向对象程序设计简明教程 / 康丽主编. —北京：中国电力出版社，2008
21世纪高等学校规划教材
ISBN 978-7-5083-7823-7

Ⅰ. C… Ⅱ. 康… Ⅲ. C语言－程序设计－高等学校－教材 Ⅳ. TP312

中国版本图书馆CIP数据核字（2008）第136178号

丛 书 序：21世纪高等学校规划教材
书　　名：C++面向对象程序设计简明教程
出版发行：中国电力出版社
地　　址：北京市三里河路6号　　邮政编码：100044
电　　话：（010）68362602　　传　　真：（010）68316497，88383619
服务电话：（010）58383411　　传　　真：（010）58383267
E-mail：infopower@cepp.com.cn
印　　刷：北京市同江印刷厂印刷
开本尺寸：184mm×260mm　　印　　张：20　　字　　数：451千字
书　　号：ISBN 978-7-5083-7823-7
版　　次：2008年9月北京第1版
印　　次：2008年9月第1次印刷
印　　数：0001—3000册
定　　价：29.80元

前　言

随着计算机应用领域的不断拓展，社会对计算机软件开发人员的需求也在急剧增长，利用各种计算机语言编写程序以解决各个领域的计算机应用问题，已经是一个非常普遍的现象。

计算机的工作是由程序与相关数据来控制的。程序是对需解决问题的数据、方法和步骤的描述，具体方式就是将这些数据、方法和步骤编写成一条条指令并输入到计算机的存储设备中。文档是为了便于人们理解程序所需的资料说明，供程序开发、用户培训和系统维护使用。

本书以介绍基本的程序设计方法和技术为中心，将C++作为入门语言，强调了面向对象思想在C++语言中的具体实现。具体内容包括数据类型、表达式、程序结构、子程序、递归、类和对象、继承和派生类、多态与虚函数、模板、输入/输出以及异常处理等。

全书共分11章，分别是：计算机与C++编程、基本数据类型和表达式、程序的基本控制结构与算法、函数与程序结构、构造数据类型、类与对象、继承与派生、多态性、模板、流类库与输入/输出和异常处理。

本书由康丽组织编写。参加编写的人员都是长期从事计算机程序设计课程教学、具有教学实践经验的老师。参加本书编写的有王玉娟（第5章 构造数据类型和第6章 类与对象）、张瑞玲（第4章 函数与程序结构）、郑立华（第10章 流类库与输入/输出）、孙龙清（第11章 异常处理）、康丽（第8章 多态性和附录）王莲芝（第1章 计算机与C++编程）、程新荣（第2章 基本数据类型和表达式）、杨璐（第3章 程序的基本控制结构与算法）、徐红升（第9章 模板）、冀荣华（第7章 继承与派生）。全书由康丽、张瑞玲统稿并修改。

与本书配套的《C++面向对象程序设计习题与实验指导》已由中国电力出版社出版。

本书免费提供电子教案和所有示例程序，读者可到中国电力出版社网站“http://www.infopower.com.cn/”中的“下载中心”下载。

由于作者水平有限，时间仓促，书中难免存在不足之处，恳请读者不吝指教，以便在以后的版本中进行改进。可发E-mail至kangli.cau@gmail.com与作者交流。

编　者

2008年6月

目　　录

第 1 章　计算机与 C++编程

自从 1946 年第一台电子计算机 ENIAC（埃尼阿克）诞生以来，计算机的发展“秒”新“分”异。现在计算机已经渗透到人类社会活动的各个领域，如信息管理系统、GPS 车辆导航定位服务、Internet 网络信息服务等，计算机已成为信息社会不可缺少的工具。

计算机系统由硬件和软件两部分构成。

硬件是指构成计算机的元器件和设备，如中央处理器、显示器、键盘、光驱等。

软件是指与计算机系统操作有关的程序、规程及任何与之有关的文档资料和数据。如 Windows XP 操作系统及其联机帮助信息、Microsoft Office 办公软件及其使用说明书等。

硬件是计算机的物质基础，没有硬件就没有计算机；软件是计算机的灵魂，没有软件，计算机就无法提供任何服务。一台计算机的性能主要由硬件决定，而软件则提供了一定的功能。

随着计算机应用领域的不断拓展，社会对计算机软件开发人员的需求也在急剧增长，利用各种计算机语言编写程序以解决各个领域的计算机应用问题，已经是一个非常普遍的现象。

1.1　计算机和程序设计语言

1.1.1　计算机工作模型

计算机作为一个信息加工处理的机器，面对人们提出的各种问题，是如何工作的？在计算机解决问题的过程中，程序设计究竟起着怎样的作用？这需要了解计算机的硬件结构和软件的工作原理。

1. 硬件结构

目前的计算机基本上依然采用的是冯・诺依曼（Von Neumann）存储程序式体系结构。图 1-1 给出了冯・诺依曼计算机的硬件构成。

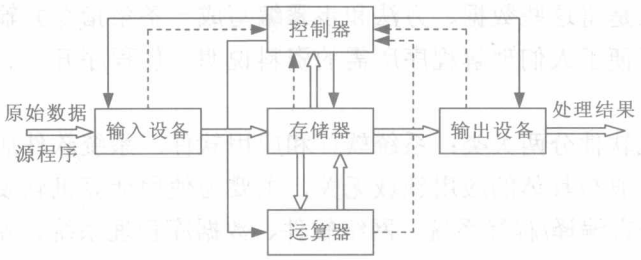

图 1-1　冯・诺依曼计算机的硬件构成

图中，实线是控制线，虚线是反馈线，双线为数据线。

冯·诺依曼计算机硬件由5个部分组成，各组成部分的功能如下。

（1）输入设备：用来输入程序和原始数据，如键盘等。

（2）存储器：用来存放程序、原始数据和运算结果，是计算机存储信息的部件。

（3）运算器：是对数据进行处理和运算的部件。进行算术运算和逻辑运算，称为ALU（Arithmetic Logical Unit）。

（4）输出设备：用来输出计算机的处理结果，如显示器、打印机等。

（5）控制器：用来实现计算机本身的自动化，实现指令的自动装入和自动执行。包括：

①从存储器中取出指令。

②对指令进行译码分析。

③发出执行指令相应的控制信号。

④从存储器（或输入设备）获得运算所需的数据。

⑤将运算器处理结果存放在存储器或由输出设备输出。

在实际结构中，将控制器和运算器集成在一个芯片上，组成计算机的核心部件，称为中央处理器，简称CPU（Central Process Unit）。

冯·诺依曼模型“存储程序”的工作原理是：一是能够存储程序和数据，二是能够自动地执行程序。计算机利用存储器（内存）来存放所要执行的程序，CPU可以依次从存储器中取出程序中的每一条指令，并加以分析和执行，直至完成全部指令。

由于构成计算机的各个部件在速度上存在一定的差别，快速部件往往要花费大量的时间等待慢速部件，所以，在冯·诺依曼计算机中存在着几个影响程序执行效率的瓶颈，它主要体现在下面两点。

（1）CPU与内存之间的数据传输。

（2）内存与输入/输出设备之间的数据传输。

现代计算机往往利用程序执行和对数据访问所具有的局部特征，通过缓存机制来解决部件之间速度不匹配问题，从而提高计算机的整体性能。缓存中存储的是近期用过的、今后可能还要用到的一些内容。例如，现代计算机大都在CPU中为内存提供高速缓存（Memory Cache），在内存中为外存提供高速缓存（Disk Cache）。

2. 软件

计算机的工作是用程序与相关数据来控制的。程序是对需解决问题的数据、方法和步骤的描述，具体方法是将这些数据、方法和步骤编写成一条条指令并输入到计算机的存储设备中。文档是为了便于人们理解程序所需的资料说明，供程序开发、用户培训和系统维护使用。

传统上把计算机软件分两大类：系统软件和应用软件。系统软件居于计算机系统中最接近硬件的一级，一般与具体的应用领域无关，主要为使用计算机软硬件资源提供方便，如操作系统、各种语言编译/解释系统、网络软件、数据库管理系统、界面工具箱等支持计算机正常运作的通用软件。应用软件是指解决某一应用领域问题的软件，如财会软件、计算机辅助设计（CAD）软件、公司网站网页等。

一台计算机上提供的系统软件的总和叫做软件（开发）平台，在此平台上编制应用程序就是应用开发。应用程序通用化、商品化后就是应用软件。应用软件的使用者是最终用户。

一个软件从无到有，再到消亡的过程称为软件的生命周期。软件生命周期分为：需求分析、软件设计、编程实现、测试及运行和维护等几个阶段。即软件的开发首先从需求分析入手，明确将来实现的软件需要解决的问题，并使用软件需求说明书描述出来；然后依据该需求说明书进行软件设计，给出概要设计和详细设计；接下来依据软件设计说明，采用某种程序设计语言编程实现；下一步进行软件测试，确认该软件是否满足需求说明书中的规定；最后是使用软件并在使用过程中发现和改正程序中的错误。

随着软件规模的不断扩大和复杂度的不断提高，早期的软件开发模式对程序的正确性难以保证，使得软件生产率急剧下降。为了解决这种软件危机，大多数软件生产商已经采用软件工程方法来开发软件。软件工程从系统工程的角度看待软件产品，试图以工程化的办法大量生产软件。软件工程强调工程管理，文档是管理的依据。软件=程序+文档。可以说，不会写文档就不懂得现代软件开发。

1.1.2　机器语言与汇编语言

程序设计语言（用于编程的语言）种类繁多，根据它们与计算机指令系统和人们解决问题所采用的描述语言（如：数学语言）的接近程度，常常把程序语言分为以下两种。

（1）低级语言：机器语言和汇编语言。

（2）高级语言：如 BASIC、C、C++、Java 等。

本节先介绍低级语言。

计算机之所以能够为我们工作，是因为我们能操纵它执行程序，所以计算机诞生之初，人们就设计了操纵计算机的语言——计算机语言。最初的语言是机器语言，机器语言是面向机器的，一台计算机的硬件系统能够识别的所有指令集合称为计算机指令系统。计算机硬件系统可以识别的二进制指令组成的语言称为机器语言。机器语言在内存中开辟两个区：数据区和指令区。数据区存放数据，指令区存放指令。CPU 从指令区第一个地址开始逐条取出指令并执行，直到所有的指令都被执行完。指令格式如下：

操作码　操作数 操作数 操作数

例如，二进制指令：

0000100 10100001 00101110

表示执行的操作是“加”，两个操作数分别是“10100001”、“00101110”。显然，使用这种语言编写程序是一般人所不能忍受的，所以人们就想到用八进制数或十六进制数来表示二进制代码指令，很快人们就发现用常见的英文单词代替约定的操作码等指令中的成分，读写程序就容易多了，于是出现了汇编语言。汇编语言将机器指令映射成可读性强的英文单词（称为助记符），如 ADD、SUB 等。如上例中的汇编指令为：

```
MOV  AX A1H
ADD  AX 2EH
```

显然，汇编语言源程序必须转换成二进制代码（即机器语言）才能被计算机接受执行，

这个工作由汇编程序翻译完成。

使用汇编语言编程需要直接安排存储，规定寄存器、运算器的动作次序，还必须知道计算机对数据约定的表示（定点、浮点）等，这对于非专业人士来讲，是一件复杂的事情。而且，由于汇编语言是符号化的机器语言，依赖于具体的计算机型号，不同的计算机在指令长度、寻址方式、指令表示等方面都不太一样，这就使得汇编程序难读、难移植。这导致了高级语言的出现。

1.1.3 高级语言

高级语言屏蔽了机器的细节，提高了语言的抽象层次，程序中可以采用具有一定含义的数据命名和容易理解的数学公式等执行语句。这使得在编写程序时可以联系到程序所描述的具体事物。

例如，计算 a+b*c–d 的值，用汇编语言可写成：

```
mov ax,b
mul ax,c
add ax,a
sub ax,d
mov r,ax
```

而用高级语言则可写成：

```
r = a+b*c-d
```

所以，高级语言是面向计算过程的，只与解决问题的步骤有关。采用高级语言编程，程序员的工作重心也为之改变，即问题变成：需要解决的问题涉及到哪个领域，有哪些对象，每个对象有哪些特征，解题的步骤等。

计算机只能根据机器语言指令来执行程序，高级语言程序也必须经过翻译变成机器语言程序，这个工作由翻译程序完成。将高级语言程序翻译成机器语言程序的翻译程序有两种：编译器和解释器。

1. 编译器工作原理

如图 1-2 所示，编译器（Compiler）是将一种高级语言完整地翻译成另一种低级语言的翻译程序。高级语言源程序经编译后得到的目标程序，一般不能立即装入机器执行，因为在源程序中用到的语言系统提供的系统函数、类模板等成分，需要连接到编译后得到的目标程序中形成可执行程序。执行时，把可执行程序装入内存中合适的位置就可执行出结果了。

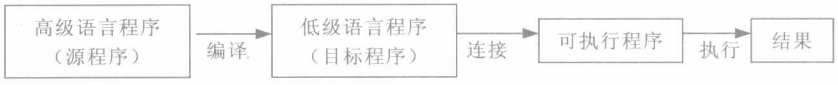

图 1-2 编译器的工作原理

编译器的重要性在于它使得多数计算机用户不必考虑与机器有关的烦琐细节，使程序员独立于机器而只专注于编程。

2. 高级语言程序的解释执行

由于使用编译器可以对源程序进行优化，所以得到的目标代码效率很高，因此大多数软件开发都采用带有编译开发环境的语言，如 C、C++、Pascal 等，这对于大型软件开发是非常有益的。不过，在有些时候，如调试程序或者教学时，需要随时检验程序的执行结果或找出程序中的问题，这时用高级语言的解释执行方式比较方便。

解释执行需要有一个解释器（Interpreter），它将源程序逐句读入，逐句执行，直到程序结束。即对源程序边解释边执行，不产生目标程序。

操作系统命令、Basic 语言、Java 语言等都是解释执行的（其中有些语言也可以编译执行），各种应用软件提供的界面语言多半是解释自行的。解释器不大，工作空间也不大，它的优点是能根据程序执行情况决定下一步做什么。不过，解释执行难于优化，效率低，这是这类语言的缺点。

目前应用比较广泛的高级语言有 FORTRAN、BASIC、PASCAL、C、C++、Java 等。

1.1.4　面向对象语言

面向对象语言属于高级程序设计语言，它是面向过程语言的进一步发展。面向过程程序只需按照解题模型精心设计数据和算法过程，即可写出程序，其结构是层层调用，如同一棵树，下层程序除自己声明的数据外共享上层和上层程序声明的数据。所以，一个子程序改动了共享数据则另一个子程序必然受其影响。这种编程方式中，只有数据类型及数据结构、过程（算法描述）和嵌套过程，把现实世界硬拆成面向过程程序表达，使用者要知道许多内部细节，设计者调试起来也极不方便。面向对象的程序设计提供了对象这一语法单位来描述现实世界中的实体，可以更直接地描述客观世界中存在的事物（对象）以及它们之间的关系。面向对象的编程语言将客观事物看作具有属性和行为的对象，通过抽象找出同一类对象的共同属性（静态特征）和行为（动态特征），形成类。通过类的继承与多态实现代码重用。

所以，采用面向对象语言编写的程序，很容易模拟客观世界对象，支持软件技术要求的局部化、重用性、可扩充性、分布性等性能，使程序能够比较直接地反映问题域的本来面目，软件开发人员能够利用人类认识事物所采用的一般思维方法来进行软件开发。具体的面向对象语言如 C++、Ada95、Object Pascal 等。

1.2　程　序　设　计

程序设计是为计算机编制程序的过程，涉及到程序设计方法、语言和步骤等方面的内容。按照软件工程的思想，程序设计是软件生存周期中编程实现阶段的工作。但在实际编程过程中，它也会涉及到软件生存周期中其他阶段的工作，所以程序设计不仅仅是用某种语言来实现设计好的软件，还需要结合软件质量、性能、测试、维护等因素统筹考虑。

1.2.1 程序设计方法

几十年来，出现了多种程序设计方法，典型的有面向过程、面向对象等。

1. 面向过程

面向过程程序设计是一种以功能为核心、基于功能分解的程序设计方法。它采用自顶向下、逐步求精的设计方法，要开发的系统采用模块分解与功能抽象，自顶向下、分而治之。

面向过程程序设计的程序结构是把要开发的系统按功能划分为若干个基本模块，形成一个树状结构。各模块间的关系尽可能简单，功能上相对独立。每一模块内部均是由顺序、选择和循环三种基本结构组成。每一模块的具体实现对应于程序中的子程序。子程序是为完成某一个子功能而编制的程序段，是对相应功能的抽象。面向过程程序的执行过程体现为一系列的子程序调用。数据大多数是依附于子程序的，在子程序调用时作为参数传递给子程序使用。经典的面向过程程序设计是：

程序 = 算法 + 数据结构

这里的算法是指对数据的一系列操作步骤的描述，数据结构是对该算法中处理的数据的描述。过程式程序设计方法开始于计算机产生初期，主要用于解决诸如科学计算一类的问题。随着计算机应用范围的不断扩大，已经不能适应庞大、复杂的软件系统的开发和维护。这种方法的主要问题是可重用性差、数据安全性差、难以开发图形界面的应用等。

2. 面向对象设计

面向对象程序设计是一种以对象为核心、基于数据抽象的程序设计方法。它采用自底向上的开发策略，大量重用原先已开发出来的过程、函数、模板等构件，要开发的系统是通过设计少量上层模块来直接调用已有构件完成的。

面向对象程序设计的程序结构是由一些对象构成的，对象是由一些数据及对这些数据进行的操作的封装体。对象的特征由相应的类描述，一个类可以从其他类继承。面向对象设计方法的关键是抽象出对象，面向对象方法中的对象是系统中用来描述客观事物的一个实体，它是用来构成系统的一个基本单位。对象由一组属性和一组行为构成。属性是用来描述对象静态特征的数据项，行为是用来描述对象动态特征的操作序列。面向对象程序的执行过程体现为各个对象之间相互发送和处理消息。面向对象程序设计是：

程序 = 对象/类+消息

对象/类=数据+操作

在面向对象程序设计中，数据和对该数据的操作是封装在一起的，对数据的操作必须通过相应的对象来进行。把对象的属性和行为结合成一个独立的系统单位，尽可能隐蔽对象的内部细节。对外形成一个边界（或者说是一道屏障），只保留有限的对外接口使之与外部发生联系。由于对象在现实世界中的相对稳定性和程序实现中的封装性，使得由对象构成的程序具有安全性、可重用性、可维护性的优点。而且，面向对象程序设计中对象与现实世界中实体的对应关系，决定了它能更直接地描述客观世界中存在的事物（对象）以及

它们之间的关系，从而有利于大型复杂软件系统的分析和设计。缺点是不适合于小型应用系统的开发，程序效率有时不高。

1.2.2　程序设计的步骤

程序设计是软件开发的核心，一般的程序设计步骤是：需求分析、系统设计、编程实现、测试与调式、运行与维护。

1. 需求分析

首先要明确需要解决的问题是什么，相关的数学模型、功能要求、性能要求，与其他程序的关系等等，应给出该程序的规格说明。

2. 系统设计

应该怎样用计算机来解决程序规格说明中的问题，系统的软件结构、硬件要求、模块划分与设计等，需考虑的软件质量指标有：可靠性、安全性、可维护性、可移植性、适应性、可测试性等。按照系统要求有选择地给出系统软件结构图、模块划分图、类图、程序流程图、测试方案等。

3. 编程实现

按照系统设计结果要求，选择一种或多种语言对设计结果编程实现。此时，应注意采用良好的程序设计风格编写程序，如良好的标识符命名约定、适当增加程序注释的习惯、不写只能在一种编程环境中可运行的代码等。

4. 测试与调式

程序写好后，有可能出现三种错误：语法错误、语义错误和运行异常。语法错误是指程序的书写不符合语言的语法规则，这类错误可由编译程序发现。语义错误是指程序设计不当造成程序没有完成预期的功能。运行异常是指由程序运行环境问题造成的程序异常终止，如：内存空间不足、打开不存在的文件进行读操作、程序执行了除以 0 的指令等等。

程序的语义错误和运行异常错误一般可以通过测试来发现。测试方法有很多，如静态测试和动态测试、白盒测试和黑盒测试等，不管采用何种测试手段，都只能发现程序有错，而不能证明程序正确。

测试工作可以与编程结合在一起，编写一部分、测试一部分，最后进行整体测试。

调试（Debug）就是给程序出错定位，找出错误的位置和原因。调试一般需要运行程序，分段观察和测试找出错误的所在。

5. 运行与维护

程序通过测试就开始正式运行了。由于没有测试手段能证明程序是完全正确的，所以在程序的使用过程中可能会出现原来意想不到的错误。程序的维护指在使用中发现并改正程序错误的过程。程序的维护分为：正确性维护、完善性维护和适应性维护。正确性维护是指为保证程序的正确运行而改正程序中的错误；完善性维护是为了使程序的功能更加完善而对程序进行的修改；适应性维护是将程序移植到不同的计算平台或环境中。

1.3 C++语言

1972 年，为编写 UNIX 操作系统，美国贝尔实验室的 Dennis Ritchie 在 B 语言的基础上设计并实现了 C 语言。后来,由于 C 语言的许多优点和 UNIX 的广泛使用，C 语言普遍应用于各种类型的应用程序。而且 Dennis Ritchie 也由于在 C 语言和 UNIX 操作系统上的突出贡献，于 1983 年获得了计算机科学领域的最高奖——图灵奖。

尽管如此，C 语言毕竟是一个面向过程的编程语言，所以也存在着一些问题。当 C 语言程序达到一定规模（代码达到 25000 行以上）后，维护和修改就变得相当困难。为了满足管理程序复杂性的需要，根除 C 中存在的问题，美国电话电报公司（AT&T）的贝尔实验室的 Bjarne Stroustrup 博士于 1979 年开始对 C 语言进行了改进和扩充，并从 Simula67 引入了面向对象程序设计的内容，最初取名为“带类的 C”，1983 年正式命名为 C++。在经历了 3 次重大修订后，于 1994 年制定了标准 ANSI C++草案，后又经不断完善，于 1998 年 11 月被国际标准化组织（ISO）批准为国际标准，成为目前的 C++。

1.3.1 C++特点

C++语言的主要特点表现在下列几个方面。

（1）全面兼容 C。即 C++是 C 语言的超集，C++由两部分组成：一是过程性语言部分，遵守 ANSI C 标准；二是类和对象部分，它是面向对象程序设计的主体。绝大多数 C 语言程序可以不经修改直接在 C++环境中运行，并且 C++比 C 更安全，C++的编译系统能检查出更多的类型错误。

（2）C++支持面向对象程序设计方法。它包含了几乎全部面向对象程序设计的特征。

（3）C++继承了 C 语言的高效率、灵活性等优点，正如 Bjarne Stroustrup 博士所说，C++使程序“结构清晰、易于扩展、易于维护而不失效率”。

C++语言的突出优点是支持面向对象的特征。但由于兼容 C，编程人员可以采用面向过程和面向对象两种不同风格来编写程序，所以应注意使用 C++编程未必就是使用了面向对象的程序设计方法编程，关于这一点在后面的有关章节中会说明。

1.3.2 C++程序结构

1. 一个简单的 C++程序

为了使读者对 C++程序的基本结构有一个总体的印象，下面给出一个例子。

【例 1-1】输出“Hello! Welcome to C++!”的 C++程序。

```
#include <iostream>
using namespace std;
int main()
{
 cout<<"Hello!\n";
 cout<<"Welcome to C++!\n";
}
```

上述程序的运行结果为：

```
Hello!
Welcome to C++!
```

这里，程序中的各行内容解释如下：

```
#include <iostream>
```

指示编译器对程序预处理时将文件 iostream 中的代码嵌入到程序中该指令所在位置，其中#include 被称为编译指令，文件 iostream 中声明了程序所需要的输入和输出操作的信息。

```
using namespace std;
```

是针对命名空间的指令，关于命名空间的概念，会在第 4 章介绍。

```
int main()
```

main 是主函数名，函数是 C++程序中最小功能单位。在 C++程序中，必须有且只能有一个名为 main()的函数，它是程序执行的开始点。main()函数之前的 int 表示 main()函数的返回值类型（关于函数返回值将在第 4 章介绍）。

```
{
 cout<<"Hello!\n";
 cout<<"Welcome to C++!\n";
}
```

以上部分是 main()函数的函数体，用一对大括号括住。

2. C++程序的组成

逻辑上，一个 C++程序由一些函数、类、全局变量、对象的定义构成，其中必须有且仅有一个 main()函数。如例 1-1 是仅由一个 main()函数构成的程序。函数由函数名、参数、返回值类型、局部变量、对象定义和函数体构成，其中的函数体是由一系列语句构成的。类由数据成员和成员函数构成，变量和对象的定义可以出现在函数的外部或函数体内部，语句只能出现在函数内部。C++程序总是从函数 main()开始执行。

物理上，一个 C++程序可以存放在一个或多个源文件中，每个文件包含一些类、函数、或外部变量、对象的定义，其中只有一个文件中包含唯一的函数 main()。每个源文件可以分别编译。C++源程序文件一般存储为后缀 .cpp 或.h 的文件（含有 main()函数的文件为后缀 .cpp 的文件，类的定义文件一般是为后缀.h 的文件），称为 C++源文件。

1.3.3　C++字符集和词法

1. C++字符集

任何一种语言都是由一些基本符号构成的，C++字符集指在 C++语言中除字符型数据外，可以出现在程序中的所有字符的集合。C++字符集包括 26 个大小写英文字母、10 个数字以及一些特殊符号构成。

（1）大小写英文字母：a～z，A～Z。

（2）数字：0～9。

（3）特殊字符：!、#、%、^、&、*、_、-、+、=、～、<、>、/、\、|、,、:、;、？、‘、“、(、)、[、]、{、}、空格、横向制表、纵向制表、换页、换行。

2. C++单词及构词规则

单词是由字符集中的字符按照一定规则构成的具有一定意义的最小语法单位。构成C++的单词有标识符、关键词、操作符和分隔符等。

（1）标识符。

标识符是对程序员自己定义的实体，如函数名、变量名、常量名、类名和对象名等起的名字。C++规定标识符的构成规则是：以英文大、小写字母或下划线_打头的由字母、数字、下划线_所构成的字符序列。使用标识符定义的实体包括常量、变量、函数、类、对象、标号等。

例如，student、student_name、x_1、_name1 等都是合法的标识符，而 No.1,1_ex 不是合法的标识符。使用标识符时需要注意以下几点。

①大写字母和小写字母代表不同的标识符，如 abc、Abc 与 ABC 是不同的标识符。

②具体实现（编译程序）可能会限制标识符的长度。

③标识符不能是 C++关键字，而且最好避开系统内部标准库等出现的名字。

④对不同种类的程序实体应尽可能采用不同形式的标识符，以提高程序的易读性。如：符号常量，采用全部大写的标识符，如圆周率 PI；自定义类型，采用相关的英文单词，且每个单词的第一个字母大写，如 StudentType；变量/对象和函数，采用小写字母，如 student、print。

（2）关键词。

关键词是 C++预先声明的单词，在程序中有不同的使用目的。常用的见表 1-1 C++关键词。

表 1-1　C++关键词

asm	auto	bad_cast	bad_typeid
bool	break	case	catch
char	class	const	const_cast
continue	default	delete	do
double	dynamic_cast	else	enum
except	explicit	extern	false
finally	float	for	friend
goto	if	inline	int
long	mutable	namespace	new
operator	private	protected	public
register	reinterpret_cast	return	short
signed	sizeof	static	static_cast
struct	switch	template	this
throw	true	try	type_info
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	while		