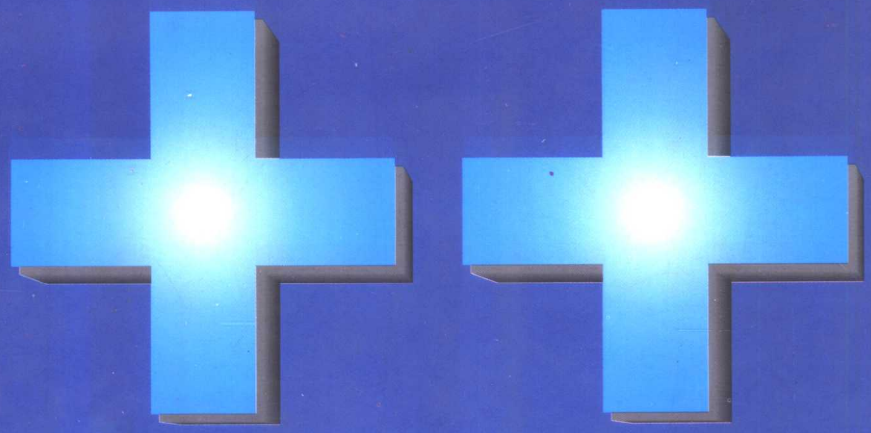
C++ 使用手册

李长江　赵志辉　陈　杰　编



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

C++使用手册

李长江　赵志辉　陈杰　　编

电子工业出版社

内容提要

本书是一本全面介绍 C++ 技术的工具书。本书不仅包括 C++ 的基础知识，还包括大量应用实例，对 C++ 的特性作了详尽说明。本书主要内容包括：面向对象程序设计技术、C++ 语言基础、C++ 专有特性、C++ 函数、类和数据封装、类的应用、运算符重载与类型转换、继承、虚函数与多态性、C++ 的 I/O 流、C++ 模板、常用数据结构的 C++ 实现、Borland C++ 集成开发环境、Borland C++ 实用工具、Borland C++ 预处理程序指令、构造 WINDOWS 应用程序、视频函数、内存模式与覆盖管理、汇编语言接口、面向对象的窗口开发技术等内容。本书内容详实，实用性强，可作为软件开发人员及广大计算机用户的工具书。

C++ 使用手册

李长江　赵志辉　陈杰　编

特约编辑：仁　忠

责任编辑：陈晓莉

电子工业出版社出版

北京市海淀区万寿路 173 信箱（100036）

电子工业出版社发行　各地新华书店经销

顺义县天竺新华印刷厂印刷

开本 787 × 1092 毫米 1/16　印张 31 7/8　字数 810 千字

1995 年 10 月第一版　1995 年 10 月北京第一次印刷

印数 5000 册　　定价 40.00 元

ISBN 7-5053-2890-X/TP·959

前　言

C++语言是一种混合语言，它溶合了面向对象技术，同时又保留了C语言的特征。C++为程序员提供面向对象的能力，同时又未丧失运行时间和空间效率。C++保持了与C的高度兼容性，使C程序员可以很容易地转向C++，同时又可以使用已有的C库。C++语言的一个最重要的方面是支持面向对象的程序设计。面向对象程序设计的主要目标是为程序员提供一个优良的程序设计环境，以生产出模块化、可重复使用及可维护性好的程序。面向对象程序设计不只是一种新的程序设计技术，它是一种全新的设计和构造软件的思维方法。由于C++提供了把数据和在数据之上的操作封装在一起的类、对象和方法的机制，并实现了面向对象程序的几个关键要素，即数据抽象、封装、继承和多态性，使得软件开发和维护变得更加容易。

本书全面介绍C++技术。本书不仅包括C++的基础知识，还包括大量应用实例，对C++的特性作了详尽说明。主要内容包括：面向对象程序设计技术、C++语言基础、C++专有特性、C++函数、类和数据封装、类的应用、运算符重载与类型转换、继承、虚函数与多态性、C++的I/O流、C++模板、常用数据结构的C++实现、Borland C++集成开发环境、Borland C++实用工具、Borland C++预处理程序指令、构造WINDOWS应用程序、视频函数、内存模式与覆盖管理、汇编语言接口、面向对象的窗口开发技术等内容。全书共分二十章，读者可以顺序阅读，也可以按内容查阅。

本书由李长江、赵志辉和陈杰编著，参加本书编写的还有王华、许林肖、程一凡、高加文和林程立。本书在编写出版中得到了张琼和陈晓莉的大力帮助，在此一并表示感谢。

编者

一九九五年五月

目　　录

第一章　面向对像程序设计技术

用计算机解决问题时需要用程序设计语言对问题的求解方法和过程加以描述。实质上，软件是问题求解的一种表述形式。显然，如果软件能够直接地表现求解问题的方法，则软件不仅易于被人理解，而且易于维护和修改，提高了软件的可靠性和可维护性。此外，如果能按人们通常的思维方式来建立问题域的模型，则可以提高公共问题域中的软件模块化和重用化的可能性。面向对像的基本原则是：按人们通常的思维方式建立问题域的模型，设计尽可能自然地表现求解方法的软件。

为了实现上述基本原则，必须建立直接表现组成问题域的事物以及这些事物间的相互联系的概念，还必须建立适应人们一般思维方式的描述范式。在面向对像的设计方法中，对像（object）和传递消息（message passing）分别是表现事物及事物间相互联系的概念。类（class）和继承（inheritance）是适应人们一般思维方式的描述范式。方法（method）是允许使用于该类对像上的各种操作。这种对像、类、消息和方法的程序设计范式的基本点在于对像的封装性（encapsulation）和继承性。通过封装能将对像的定义和对像的实现分开，通过继承能体现类与类之间的关系以及由此带来的动态束定（dynamic binding）和实体的多态性（polymorphism），从而构成了面向对像程序设计技术的基本特征。

1.1　对像

客观世界的问题都是由客观世界的实体及实体间的相互关系构成的，我们把客观世界的实体称为问题域的对像。显然，“对像”不是固定的。一本书可以是一个对象，一家图书馆也可以是一个对像。可见，任何事物都是对像，是某一个对像类的一个元素。复杂的对像可由相对比较简单的对像以某种方法组成，甚至整个世界也可以看成是由一些最原始的对像，经过层层组合而成的。

用计算机解题是借助某种语言规定对计算机实体施加一系列的动作，以此动作的结果去映射解，我们把计算机实体称为求解域的对像。

从动态的观点来看，对像的操作就是对像的行为。问题域对像的行为是极其丰富多彩的，而解空间对像的行为是极其死板的。因此，只有借助于极其复杂的算法才能操纵解空间对像而得到解。传统的程序设计语言限制了程序员定义解空间对像。而面向对像语言提供了“对像”概念，这样，程序员就可以自己去定义解空间对像。

从存储的角度来看，“对像”是一片私有存储，其中有数据也有方法（例程或单个指令）。其他对像的方法不能直接操纵该对像的私有数据，只有对像私有的方法才可操纵它。

从对像的实现机制来看，“对像”是一台自动机，其中的私有数据表示了对像的状态，该状态只能由私有的方法改变。每当需要改变对像的状态时，只能由其它对像向该对像发送消息，对像响应消息后按照消息模式找出匹配的方法，并执行该方法。

在面向对像的程序设计中，“对像”是系统中的基本运行实体。换句话说，“对像”是具

有特殊属性（数据）和行为方式（方法）的实体。对像占有存储空间且具有传统程序设计语言的数据，如数字、数组、字符串和记录。给对像分配存储单元就确定了给定时刻对像的状态。与每一个对像相关的方法定义了该对像上的操作。

1.2 消息和方法

如何要求对像完成一定的处理工作？对像间如何进行联系？所有这一切都只能通过传递消息来实现。消息用来请求对像执行某一处理或回答某些信息的要求；消息统一了数据流和控制流；某一对像在执行相应的处理时，如果需要，它可以通过传递消息请求其他对像完成某些处理工作或回答某些信息；其他对像在执行所要求的处理活动时，同样可以通过传递消息与别的对像联系。因此，程序的执行是靠在对像间传递消息来完成的。

发送消息的对像称为发送者，接收消息的对像称为接收者。消息中只包含发送者的要求，它告诉接收者需要完成哪些处理，但并不指示接收者应该怎样完成这些处理。消息完全由接收者解释，接收者独立决定采用什么方式完成所需的处理。一个对像能够接收不同形式、不同内容的多个消息；相同形式的消息可以送往不同的对像；不同的对像对于形式相同的消息可以有不同的解释，能够作出不同的反映。对于传来的消息，对像可以返回相应的回答信息，但这种返回并不是必须的，这与子程序的调用/返回有着明显的不同。

消息的形式用消息模式（message pattern）刻画，一个消息模式定义了一类消息，它可以对应内容不同的消息。对于同一消息模式的不同消息，同一个对像所作的解释和处理都是相同的，只是处理的结果可能不同。对像的固有处理能力按消息分类，一个消息模式定义对像的一种处理能力。这种处理能力是通过该模式及消息引用表现出来的。所以，只要给出对像的所有消息模式及相应于每一个消息模式的处理能力，也就定义了一个对像的外部特性。消息模式不仅定义了对像所能受理的消息，而且还定义了对像的固有处理能力，它是定义对像接口的唯一信息。使用对像只需了解它的消息模式，所以对像具有极强的“黑盒”性。

把所有对像分成各种对像类，每个对像类都定义一组所谓的“方法”，它们实际上可视为允许使用于该类对像上的各种操作。

当一个面向对像的程序运行时，一般要做三件事：首先，根据需要创建对像；其次，当程序处理信息或响应来自用户的输入时，要从一个对像传递消息到另一对像（或从用户到对像）；最后，若不再需要该对像时，应删除它并收回它所占用的存储单元。

1.3 类和类层次

在面向对像程序设计中，“对像”是程序的基本单位，相似的对像可以和传统语言中的变量与类型关系一样，归并到一类（class）中去。程序员只需定义一个类对像就可以得到若干个实例（instance）对像了。

具体来说，类由方法和数据组成，它是关于对像性质的描述，包括外部特性和内部实现两个方面。类通过描述消息模式及其相应的处理能力来定义对像的外部特性，通过描述内部状态的表现形式及固有处理能力的实现来定义对像的内部实现。类的形式如图 1.1 所示。

一个类实质上定义的是一种对像类型，它描述了属于该类型的所有对像的性质。

对像是在执行过程中由其所属的类动态生成的，一个类可以生成多个不同的对像。同一个类的所有对像具有相同的性质，即其外部特性和内部实现都是相同的。一个对像的内部状态只能由其自身来修改，任何别的对像都不能够改变它。因此，同一个类的对像虽然在内部状态的表现形式上相同，但它们可以有不同的内部状态，这些对像并不是完全一模一样的。

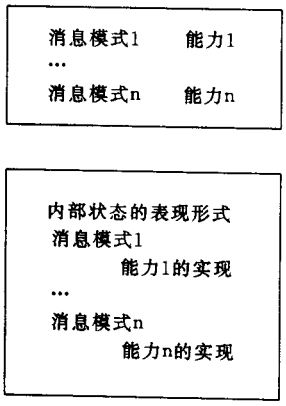


图 1.1 类的形式

从理论上来讲，类是一个 ADT（抽象数据类型）的实现。信息隐蔽原则表明类中的所有数据是私有的。类的公共接口是由两种类型的类方法组成。一种是返回有关实例状态的抽象辅助函数，另一种是用于改变实例状态的变换过程。

一个类的上层可以有超类（superclass），下层可以有子类（subclass），形成一种层次结构。这种层次结构的一个重要特点是继承性，一个类（直接）继承其超类的全部描述。这种继承具有传递性，即如果 C1 继承 C2，C2 继承 C3，则 C1（间接）继承 C3。所以，一个类实际上继承了层次结构中在其上面的所有类的全部描述。因此，属于某个类的对像除具有该类所描述的特性外，还具有层次结构中该类上面所有类描述的全部特性。

在类的层次结构中，一个类可以有多个子类，也可以有多个超类。因此，一个类可以直接继承多个类，这种继承方式称为多重继承（multiple inheritance），如图 1.2（a）所示。如果限制一个类至多只能有一个超类，则一个类至多只能直接继承一个类，这种继承方式称为单重或简单继承（single inheritance），如图 1.2（b）所示。在简单继承情况下，类的层次结构为树结构，而多重继承是网状结构。

抽象类（Abstract class）是一种不能建立实例的类。抽象类将有关的类组织在一起，提供一个公共的根，其它一系列的子类都从这个根派生出来。抽象类刻画了公共行为的特征并将这些特征传给它的子类。通常一个抽象类只描述与这个类有关的操作接口或者这些操作的部分实现，完整的实现被留给一个或几个子类。有时抽象类描述了这个类的完整实现，但只有在将这个类和其它的类组合在一个新的类中时它才有用。抽象类已为一个特定的选择器集合定义了方法，并且这些方法服从某种语义，所以抽象类的通常用途是用来定义一种协议（或

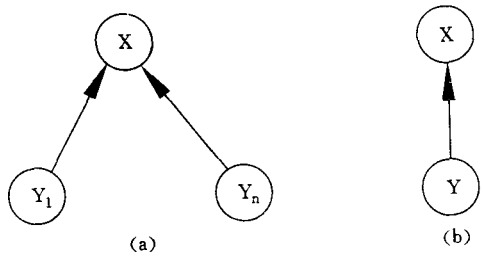


图 1.2 继承方式（X 是 Y，Y1，Y2，...，Yn 的超类）
(a) 多重继承； (b) 简单继承

概念）。

例如类 Person，如果它只用来说明 Teacher 和 Student 所具有的公共特征，如名字、地址以及对它们的访问，单独的 Person 实例在实际中是不具有意义的，只有当它和它的子类 Teacher 和 Student 结合起来才能用来描述一个部门的具体对像，因此我们可以认为 Person 是一个抽象类。

判别一个类是否是抽象类，最容易的办法是使用它。如果有必要建立一个类，但是只使用从它那儿继承过来的子类，则这个类是抽象类。

一个抽象类包含了操作接口但没有实现，就定义了一种协议。如果包含有某种实现，则抽象类用缺省实现定义了一种协议。

综上所述，类是对一组对像的抽象，它将该种对像所具有的共同特征（包括操作特征和存储特征）集中起来，由该种对像所共享。在系统构成上，类则形成了一个具有特定功能的模块和一种代码共享的手段。

1.4 继承性

继承性（inheritance）是自动地共享类、子类和对像中的方法和数据的机制。每个对像都是某个类的实例，一个系统中类对像是封闭的。如果没有继承性机制，则类对像中数据和方法就可能出现大量重复。

继承关系常称“即是”（is_a）关系。这是因为当类 Y 继承类 X 时，由继承性可知 Y 现已具备 X 的全部性质，所以 Y 即是 X。无疑，Y 也可能包含了 X 中没有的特性，它具有比 X 更多的性质。

因此，继承关系常用于反映抽象和结构。图 1.3 是一个图形系统的例子。Rectangle（矩形）是一种特殊类型的 Polygon（多边形），这很容易从继承关系中获得。当 Rectangle 继承 Polygon 时，Rectangle 获得了 Polygon 的全部性质。另外，Polygon 是 Closed Figure（闭合图形），所以 Rectangle 也继承 Closed Figure 的所有性质。

图形系统中最高层的图形具有象素宽度、颜色以及平移和旋转等特性，这些是在根类 Figure 中定义的。另外，Closed Figure 有边界和区域，Polygon 增加了边数属性。所以，根据继

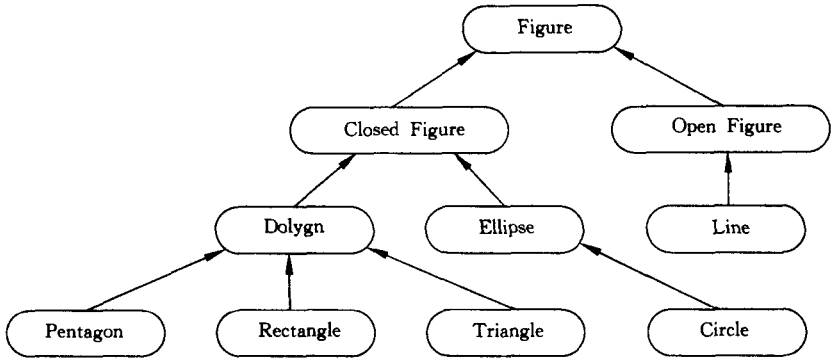


图 1-3 一个继承图的例子

承性 Rectangle 有象素宽度、平移过程和计算区域函数等。当然，一些性质可直接继承，也有一些应在继承时做某种修改再用。如，类 Figure 定义的象素宽度可能适合 Rectangle，但计算区域面积过程应重新定义，以便给出更有效的实现。

可见，在面向对像的图形系统中，继承性减轻了增加新图形原语的工作。同样地，面向对像方法利用继承性也有助于开发快速原型。

1.5 封装性

封装是一种信息隐蔽技术，用户只能见到对像封装界面上的信息，对像内部对用户是隐蔽的。封装的目的在于将对像的使用者和对像的设计者分开，使用者不必知道行为实现的细节；只须用设计者提供的消息来访问该对像。

封装的定义为：

(1) 一个清楚的边界，所有对像的内部软件的范围被限定在这个边界内；

(2) 一个接口，这个接口描述这个对像和其他的对像之间的相互作用；

(3) 受保护的内部实现，这个实现给出了由软件对像提供的功能的细节，实现细节不能在定义这个对像的类的外面访问。

封装的基本单位是对像，这个对像的性质由它的类说明来描述，这个性质被具有同样类的其他对像共享。借助对封装的这种定义，一个类的每一个实例在一个问题求解中是一个单独的封装。

对像用于封装的概念可以和集成电路芯片作一类比。一块集成电路芯片由陶瓷封装起来，其内部电路是不可见的，也是使用者不关心的。芯片的使用者只关心芯片引脚的个数、引脚的电气参数以及引脚提供的功能，通过这些引脚，硬件工程师对这个芯片有了全面的了解。硬件工程师将不同的芯片引脚连在一起，就可以组装一个具有一定功能的产品。软件工程师通

过使用类也努力达到这个目的。

面向对像的语言以对像协议（protocol）或规格说明（specification）作为对像的外界面。协议指明该对像所能接受的消息，在对像的内部，每个消息响应一个方法，方法实施对数据的运算。对数据方法的描述是协议的实现部分或类体。

显式地将对像的定义和对像的实现分开是面向对像系统的一大特色。封装本身即模块性，把定义模块和实现模块分开，就使得用面向对像技术所开发设计的软件的维护性、修改性大为改善，这也是软件技术追求的目标之一。

1.6 多态性

在大多数面向对像的语言中，如果类 P 是子类 S 的父类，则子类 S 的对像 s 可以用在父类 P 的对像 p 使用的地方，这就意味着一个公共的消息集（即操作）可以送到类 P 和类 S 的对像上。当同样的消息可以被送到一个父类的对像和它的子类的对像上时，这被称作多态性（Polymorphism）。在不同的对像上对一个相似的操作使用相同消息的能力与人类在解决问题中的思考方式是相容的，不必为打印整数、浮点数、字符、字符串和数据记录而使用不同的术语。

所谓多态（polymorphic）即是指一个名字可具有多种语义。在面向对像的语言中，多态引用表示可引用多个类的实例。由于多态具有可表示对像的多个类的能力，因此，它既与动态类型有关又与静态类型有关。

动态类型是在程序执行期间内随时可以改变的类型。在强类型面向对像环境中，运行系统根据动态类型自动地保持标记的全部多态引用。静态类型是在程序正文中声明实体时确定下来的类型。在编译时已经知道并确定了对像在运行时的有效类型集合。

继承性的“即是”性质与强类型面向对像语言的多态性思想有着密切的联系。该思想是：如果 Y 继承 X，那么 Y 即是 X，所以，在任何期望 X 的实例的地方，也允许使用 Y 的实例。

1.7 C++与面向对像

C++的名字是 Rick Maseitti 提出来的，名字强调了从 C 变化过来的演化特性。C++的名字直到 1983 年夏才定下来。这个语言在早期的版本中通称“带来的 C”，1980 年就已开始使用。C++的出现主要归因于 C。C 在 C++中作为子集保留下来，因为 C 着重于那些低级得足以应付多数系统程序设计需求的设施。C 的成功要归因于它的前身 BCPL，事实上，BCPL 的注释约定//就再次引入到 C++中。C++的另一个有启发性的来源是 Simula67，类的概念（包括派生类和虚函数）就来自于它。C++的重载运算符设施，以及可以把声明自由地置于语名能出现的任何地方都类似于 Algol 68。C 是 C++的基语言。C 的基本目标之一是在最常用的系统程序设计任务中代替汇编编码。在设计 C++时不折不扣地要继承这方面的成果，C 和 C++的不同主要在于对类型和结构的重视程度上。C 易于表达且比较自由，C++更易于表达，不过为了表达得更好，程序员必须非常注意对像的类型。知道了对像的类型，编译程序才能正确地处理表达式，否则，程序员就必须不厌其烦地指明该对像运算的细节；知道了对象的类型，编译程序才可以检查出错误，否则这些错误就要一直保留到测试时刻。使用类型系统可使函数参数得到检查，

保证数据免遭有意无意的误用，提供新类型以及新的运算符等，这些都不会增加运行时刻的时空开销。

一个程序设计任务的理想途径可分为三个阶段：首先对问题应有一个清楚的了解，然后标识解中包括的关键概念，最后用程序表达该问题解。然而，问题的细节和解的概念常常只有在程序表达它们之后才能有清晰的理解——这就是选择程序设计语言的问题所在。

在大多数应用中，有些概念在程序中无论是作为一种基本类型还是作为一个没有关联静态数据的函数都不易表示。一旦存在这种概念，就应在程序中声明一个类去表示它。类也是一种类型，它指明此类对像的行为，即如何去建立、操纵直至撤消它们。类还指明对像如何表示，但在程序设计的早期阶段，这不是（也不应该是）关注的主要问题。写一个良好程序的关键在于设计类，以使每个类都清晰地表示一个单一的概念。通常这就意味着程序员必须考虑以下问题：该类对像如何建立？该类对像能复制或撤消吗？这些对像上能实施哪些操作？如果没能对这些问题作出很好的回答，可能是因为在第一阶段中概念尚未弄清楚，最好是再细分析该问题及所要求的解，而不要立即编程。

C++支持基本的面向对像概念：对像、类、方法、消息、子类和继承性（包括多重继承性）。表 1.1 给出了 C++对这些概念的命名约定。

表 1.1　C++的面向对像概念的命名约定

概　念	C++
对　象	对　象
类	类
方　法	成员函数
实例变量	成　[illegible]
消　息	函数调用
子　类	派生类
继函性	派　生

C++程序设计语言是在 C 语言的基础上，扩充了类（class）、内联函数、运算符重载、常量类型、引用、自由存储管理运算符、函数参数检查以及新函数定义竺语法。

第二章　C++语言基础

2.1　基础知识

本节讲述两个重要的编程概念，即从源代码到可执行程序的产生方法和程序的执行流程。首先将讨论一个程序如何由源代码编译成可执行代码，然后再介绍如何控制程序的流程。

2.1.1　源文件、目标文件和装载模块

向计算机中敲入程序的过程是建立源程序文件的过程，源程序是一种用类似英语的语言编写的、程序员能够读懂的程序。这种程序不能运行，因为计算机不能理解源程序文件，必须把源文件转换为计算机能够识别的语言。

程序生成过程的第一步是使用文本编辑器编写一个C++语言源文件（计算机所遵循的指令表）。C++语言是一种计算机语言，它表达了程序的工作方式。但是计算机并不懂得用户所写的这种类似英语的指令。

为了让计算机能够理解用户提供给它的指令，必须将源程序转换成计算机可以理解的程序，这就是编译，它将C++语言源文件转换成由机器语言指令组成的目标模块（目标代码），也就是我们常说的可执行文件。

将C++语言程序转换成机器语言程序时，编译程序要执行下列动作：

- ☐ 读源文件，并且将它转换为一连串的预处理符号和空白字符。在这一阶段，特殊字符得以转换，占两行或三行的语句指令合并在一起，预处理符号是编译程序能够处理的简单元素。

  例如：操作符、常数和关键字都属于符号。空白字符包括空格符、制表符和注释，在这个过程中，注释被简单地用一个空格替换掉。
- ☐ “执行”预处理器指令，这些指令包括#indude和#define等。#include指令告诉编译程序从另一个文件里读取源代码。这些源代码放置在一起进行与前面相同的初始化处理，被定义的宏也在此时展开。
- ☐ 进一步进行字符和串的处理，分析程序中的语法和语义。在整个编译期间，任何非法的编码错误都会被找出来。
- ☐ 如果未发现严重错误则生成目标文件。

预处理程序的#include指令使用户能够一次编译多个源文件。#include指令是使预处理程序暂停处理当前的源文件，而把#include中规定的源文件合并到该文件中。

例如，下列指令指示预处理程序暂时停止对程序进行处理：

```
#include <stdio.h>
```

预处理程序读取stdio.h头文件，把该文件里的C代码包括到源代码中来。#include不限制一定是头文件，而可以包含用户指定的任何文件。