北京科海培训中心系列教材

C++程序设计语言

〔美〕Bjarne Stroustrup著

麦中凡　童长忠　译

金茂忠　校

清华大学出版社

北京科海培训中心系列教材

C++程序设计语言

[美] Bjarne Stroustrup 著

麦中凡 童长忠 译

金茂忠 校

清华大学出版社

内 容 提 要

C++语言是C语言为适应近代软件工程而发展的超集。它增加了面向对象程序设计所需要的抽象数据类型——类，以及围绕类增加了10种语言机制，并消除了C中宏的不安全性，使程序员编程更方便好用。

全书共分八章，第一章是C++的概貌介绍，第二、三、四章和C大同小异，第五章以后介绍类、运算符重载、派生类和用于输入/输出的流。全书从设计程序的角度介绍语法，书末附有“C++参考手册”，是一本学习C++新的设计风格的好书。

本书适用于广大软件工作者，大专院校本科高年级学生和硕士生。此书可作为学习C++程序设计语言的教材。

C++程序设计语言
[美] Bjarne stroustrup著
麦中凡　童长忠　译
金茂忠　校
☆
清华大学出版社出版
北京　清华园
蔚县印刷厂印刷
新华书店北京发行所发行
☆
开本：787×1092　1/16　印张：17　字数：411千字
1988年12月第1版　1988年12月第1次印刷
印数：0,0001—0,4000 定价：7.45元
ISBN 7—302—00351—8/TP·121

致 读 者

本章由对本书的概览、参考资料目录及C++的一些附加注释所组成。这些注释涉及到C++的历史，影响C++设计的某些思想以及用C++编程的思想风格。本章不是导论，这些注释不是了解后面各章的先决条件，某些注释还要用到C++的知识。

本书的结构

第一章对C++的主要特征很快浏览一遍，其目的在于使读者对本语言有个印象。用C编程的程序员很快就可看完本章的前半部分，它包括与C通用的C++特征。后半部分讲述C++中定义新类型的一些设施；初学者可在学完第2、3、4章以后再来仔细地研究这部分。

第2、3、4章讲述C++中除定义新类型之外的特征：即基本类型、表达式和C++程序的控制结构。换句话说，讲的是C++的子集，即C++的精华部分，涉及到比第一章更多的细节，但完整的信息还要到参考手册中去找。然而，这些章节提供的例题、观点、建议、警告和练习是手册中不曾有的。

第5、6和7章讲述C++中定义新类型的设施，它们是C所没有的特征。第5章提出类（class）的基本概念，展示了一个由用户定义类型的对象如何初始化、访问、直至最终清除。第6章讲述如何为用户定义的类型定义单目和双目运算符，如何指明用户定义类型之间的相互转换，以及如何去建立、删除、拷贝用户定义类型的值以便处理。第7章讲述派生类的概念，它使程序员可以借助简单类建立更加复杂的类。派生类提供一个类的替换界面，并可以一个有效的、根据上下文确定类型的方式来处理那些其类型在编译时刻不能确定的对象。

第8章介绍标准库中为输入输出提供的istream和ostream类。本章有双重目的：介绍一个有用的设施，同时也是C++使用的实际例子。

最后是C++参考手册。

参考本书的章节用§2.3.4的形式表示(第2章第3.4节)，注明r的章表示参考手册，例如§r.8.5.5。

有关实现的注释

截至写书时为止，所有C++的实现都采用了单一的前端编译版本。它适用于大量机型，包括以AT&T 3B系列、DEC VAX系列、IBM 370系列以及Motorola 68000上运行的UNIX操作系统的版本。本书的程序片段都直接取自运行UNIX系统V第2版[15]的源文件，有些取自VAX 11/750，还有一些在CCI Power 6/32的BSD4.2 UNIX[17]上运行过。本书讲述的是“纯C++”，但当前的编译也都实现了一些“不合潮流的保留”（在§r.15.3中叙述），它们很容易从C转移到C++上来。

练习

在每章末尾都有练习，主要是练习写一个程序的多样性，常常为了编译和运行的需要

写出足够的代码，并需带有少量测试用例。练习的难度差别较大，所以都标有估计它们难度的标记，难度呈指数等级，若（*1）练习花你5分钟，（*2）练习则需一小时，（*3）练习就得花一天时间。为写出并测试一个程序所需的时间不完全取决于练习本身，更多地要取决于读者的经验。如果读者在一新型计算机系统上运行而不得不熟悉机器，（*1）练习也可能花一天时间。反之，对于碰巧收藏有合适程序的人，一个（*5）练习一小时也可能做完。任何论述C程序设计的书都可作为第2—4章练习的来源。Aho等人[1]根据抽象数据类型提出了许多通用的数据结构和算法，因而也可以作为第5—7章练习的来源。然而，那些书上使用的语言缺少成员函数和派生类。因此，C++经常能采用用户定义的类型使问题表达得更加优美。

设计注释

简单性是一个重要的设计原则：在简化手册及其文档与简化编译程序之间有一选择，我们选择了前者。与C保持兼容性也十分重要，这就不得不顾及C的语法。

C++没有高级的数据类型，也没有高级的元操作。例如，没有带运算符的矩阵类型，也没有带并运算符的串类型。若用户需要这些类型，他完全可以用语言本身去定义。事实上，定义新的通用或专用类型是用C++进行最基本的程序设计工作。对于一个良好的设计，用户定义类型和内部定义类型的差别仅仅是它们定义的方式不同，使用上没有什么两样。

本设计极力避免那些甚至还未使用就要支付运行和存储开销的特征。例如：必须为每个对象存储自己的“内务信息”的想法一定会遭到拒绝；若用户声明了由两个16位的量组成的结构，则该结构将组装到一个32位的寄存器之中。

C++的设计用的是颇为传统的编译技术和运行环境。C++没有诸如需要不寻常装载程序和运行时刻支持的异常处理或并发程序设计的那些设施。因而C++的实现非常容易移植。然而，有更多的理由要使C++在更为有效的支持环境下工作，诸如动态装载、增量编译以及类型定义数据库等设施都可以利用，而不会对语言有影响。

C++的类型和数据隐藏设施依赖编译时刻的程序分析，以防止数据的偶然误用。它们并不提供保密或针对有人故意破坏规则的检查。然而，它们可自由地使用而不会引起运行时间和空间的额外开销。

历史注释

显然，C++的出现主要归因于C[7]。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[9]，事实上，BCPL的注释约定//就再次引入到C++中。如果你清楚BCPL，就会注意到C++仍然没用$valof。C++的另一个有启发性的主要来源是simula 67[23]，类的概念（包括派生类和虚函数）就来自于它。simula 67的inspect语句是故意未引进C++的，其理由是，使用虚函数有助于模块性。C++的重载运算符设施，以及可以把声明自由地置于语句能出现的任何地方都类似于Algol 68[14]。

C++的名字直到1983年夏天才定下来。这个语言早期的版本通称“带类的C”[13]，1980年就开始使用。语言最初的创建是因为作者想写一个以事件制导的仿真程序

而引起的。除了效率因素外，该程序要具有simula 67的思想。“带类的C”主要用于仿真课题，在这些课题中写程序的设施所用的时空要经过严格的最小化测试。“带类的C”没有运算符重载、引用、虚函数和其他许多细节。1983年7月，C++第一次向作者的研究小组之外发表，然而，好几个当前C++的特征当时还没有开发出来。

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。“++”是C的增量运算符，略为短一些的名字C+是一个语法错误，它也曾作过另一个语言的名字。C语义学家们发现C++要次于++C。语言之所以没叫D语言，是因为它是C语言的扩充，也没有打算去掉C的某些特征。对于名字C++的另一些解释请参阅Orwell [8] 著作的附录。

设计C++首先不能认为作者和他的朋友们将不再使用汇编、C或其它现代高级语言去编程序，主要目的是使得各个程序员能更容易且更加愉快地写出良好程序。C++从未有过设计论文。设计、文档和实现同时进行。自然，C++的前端是用C++写出，从来没有“C++项目”，也没有“C++设计委员会”，在C++演进和逐次演进的历程中，始终是针对用户碰到的问题，通过作者和他的朋友、同事的讨论来解决。

C所以选作C++基语言是因为：

a. 多功能、简洁且相对低级；

b. 足以应付多数系统程序设计任务；

c. 处处均可运行；

d. 适合于UNIX程序设计环境；

C也有它自己的问题，但是一个语言的设计在它的起点上总会有某些问题存在，而且我们已熟知C的问题了。最重要的是，在把类似simula的类加到C中的最先设想的数月中，用C工作就能使“带类的C”成为一种有用（即使是笨拙的）的工具。

随着C++日益广泛的应用，以及它提供的设施日益明显地超出C，是否保留兼容性的问题会一而再地提出来。显然，如果拒绝继承C的某些特征，这些问题是可以避免的。（参阅Sethi的例子 [12] ）之所以没有这样做是因为：

a. 如果不必把C程序完全改写为C++，则有数百万行C代码可为C++所用；

b. 假定C++能和C完全兼容地相连接，且语法非常类似C，则有数十万行用C写的库函数和实用软件代码可以用于C++程序中；

c. 有数万熟知C的程序员，他们仅需学习C++新特征的使用，并不需再学习其基本部分；

d. 由于C++和C要被同样的人在同样的系统上使用好些年，为使其错误和混乱尽可能地少，因而其差别要么很大要么很小；

最近几年，C语言本身也在发展，部分地受到C++发展的影响（参阅Rosler [11]）。ANSI C标准的最初方案 [10] 包括一个从“带类的C”中借鉴过来的函数声明语法。借鉴有两种方式：例如，void*指针类型就是ANSI C发明的，并首次在C++中得以实现。每当ANSI C标准稍许有些发展，就应再次复审C++以消除毫无道理的不兼容性。再如，若需更新预处理程序（§r.11），则浮点算术规则也可能要调整。使C和ANSI C都很接近C++的子集应该是不困难的（参阅§r.15）。

效率和结构

C++是从C程序设计语言发展而来，除极个别的例外情况外，保留C作为它的子集。基础语言，即C++的子集C，其设计使得其类型、运算符与它的语句及计算机直接处理的数、字符和地址之间紧密呼应，除了自由存储运算符new和delete外，纯粹的C++表达式和语句一般都需要非隐藏的运行时刻的支持或例程。

C++的函数调用和返回序列与C用法相同。当这种相对有效的机制开销过于昂贵时，C++的函数可用内联（inline）置换，这样，既享有函数的习惯表示，又无额外的运行时刻开销。

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象的类型。知道了对象的类型，编译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象运算的细节。知道了对象的类型，编译程序还可以检查出错误，否则这些错误一直要保留到测试时刻。注意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等，这些都不会增加运行时刻的时空开销。

在C++的设计中，强调结构势必增加从前用C实现的程序规模。你可以不顾及良好风格的各条准则，用蛮力去设计一个小程序（小于1000行）。但对较大程序这样做就不行了。如果一个10000行程序的结构不好，你将会发现引入新错误和消除老错误一样快。C++的设计就使得较大程序能以合理的方式加以构造，所以，一个人能对付25000行以上代码就不足为奇了。多数已有的较大程序，通常由许多差不多是独立的部分组合到一起工作的。每一部分完全处于前面提到的限定之下。自然，程序编写和维护的困难不完全决定于程序正文的行数，还要取决于应用的复杂性，所以，用来说明上述思想的数字不要死抠。

然而，并非每一个代码片段都可以构造得很好，并独立于硬件，易于阅读等等。C++具有以直接有效的方式操纵硬件设施的特征，但未考虑完全性和易理解性。它还具有把代码隐藏在优美的安全界面后面的那些特征。

本书着重介绍的技术是：提供通用设施、普遍有用的类型和库等等。这些技术不仅可供编写小程序的程序员使用，也可满足编写大程序的程序员。因为所有正规的程序都是由许多基本独立的部分所组成，系统和应用程序员利用这些技术可写出这些部分。

用十分详细的类型结构来说明程序也许会觉得它会导致较长的源程序正文。C++不会如此，C++程序声明函数参数类型、使用类等等，一般说来，较之不用这些设施的等价的C程序还要短一些。

哲学注释

程序设计语言服从于两个相关的目的：提供一个工具使程序员去指定要执行的动作，以及提供一组概念使程序员用来思考能作些什么。前者要求语言“与机器紧密相连”才理想，使得程序员能看到机器各个主要部位是如何简单而有效地进行工作。C语言原本主要

直以这种思想设计的。后者要求语言“与求解的问题紧密相连”才理想，使得解的概念能接且又简捷地表达出来，把这些设施加到C上从而产生了C++，它基本上按后一种思想设计的。

用以思考/编程的语言与我们能想象到的问题和解之间的联系十分紧密。正因为如此，限制语言特征以削减程序员的错误，即使作最乐观的估计，也是一种危险做法。和自然语言一样，能用两种语言会有极大的好处。语言为程序员提供了一组概念工具；如果不足以应付任务，它们将被忽略。例如：严格限制指针的概念仅仅强制程序员用一个向量加上整数运算去实现结构、指针等。语言特征过少，就不能满足良好设计和消除错误的需要。

类型系统应明显地有助于任务。事实上，C++类的概念业已证明它本身就可作为一种强有力的概念工具。

用C++作程序设计的设想

一个程序设计任务的理想途径可分成三个阶段：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，则在程序中声明一个类去表示它。类也是一种类型，它指明此类对象的行为，即如何去建立、操纵直至撤消它们。类还指明对象如何表示，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致力于以下问题：该类对象如何建立？该类对象能复制或撤消吗？这些对象上能实施哪些操作？如果对这些问题没有作出很好的回答，可能是因为在第一阶段中概念尚未弄“清晰”，最好是再多想想这个问题及要求的解，而不要立即开始本问题的“代码解”。

具有传统数学形式化的概念是最容易处理的，如：全排序数、集合、几何形状等。表示这些概念实际应该具有类的标准库，但在写书时还未到这一地步。C++还很年轻，它的库还没有成熟到和语言本身那样的程度。

概念并不存在于真空中，总包括在相关概念的簇中。组织程序中类与类之间的关系，即决定包含在解中不同概念间的准确关系，常常比在第一阶段中建立单个类还难。所以，最好不要使各个类（概念）弄得相互牵扯。设有两个类A和B，若其间存在关系“A调用B中的函数”，“A建立B”，和“A有一个B的成员”很少会出问题。若关系是“A用B中数据”（只要不使用公有数据成员），有了问题一般也能消除。难点通常在于把关系表达为“A是B与…”。

管理复杂度的最强有力的智能工具之一是层次安排；即把相关的概念组织到一个树结构之中，最通用的概念处于根部。在C++中派生类就可表示这种结构。通常程序能组织成树的集合（森林？），这就是说，程序员指定了一些基类，每个基类都具有自己的派生类的集合。通常虚函数（§7.2.8）可用来为一个概念的最通用版本（基类）定义一组操作集。如果必要，这些操作可按特定情况（派生类）精炼地作出解释。

自然，这种组织有其限度。尤其当一概念直接依赖于一个以上其他概念时，把概念集

合组织成无循环有向图有时会更好些，例如："A是B与C与…"。在C++中没有能直接支持这种关系的结构。这种关系如果不要求那么雅致，并附加上一些额外工作（§7.2.5），还是可以表示的。

有时，甚至无循环有向图都不能胜任组织一个程序的概念，某些概念似乎本身就固有多重依赖关系。如果一组多重依赖的类相当小而不难理解，则环形依从性并不存在什么问题。在C++中，friend（友元）类的思想可用来表示多重依赖类的集合。

如果把你的程序概念组织成一般的图（不是树或无循环有向图），而你又不能分散多重依赖性，那么你最有可能陷入困境，而程序设计语言也不能为你解脱。除非你能很容易地构思基本概念之间的状态关系，程序很可能变得不可管理。

要记住，多数程序设计只用基本类型、数据结构、简单函数以及少量的取自标准库的类，即可简单清晰地作出。涉及定义新类型的全部设施除非真正需要它们，否则不应该使用。

"如何用C++编出精致的程序？"的问题非常类似于"如何写出精美的英语散文？"，对这样的问题有两种回答："先搞清楚你想说什么"及"实践。模仿好的作品"。这两种劝告象它们适合英文一样，似乎也适合于C++——并且象英文一样难以遵循。

经验与教训

这里介绍一组"准则"，它们是你在学习C++时应该领会的。随着你对C++更熟练，你能够将它们演变成适合于你的应用及程序设计风格。它们有意说得非常简单，因而缺少细节，不要逐字斟酌它们。要想编出一个精致的程序，需要智慧、爱好和耐性。你第一次也许不能正确地理解它们，这只有通过实践！

[1] 当你编程序时，你会产生对某一问题解的思想的具体表示。应使程序结构尽可能直接地反映这些思想：

- [A] 若你能认为"它"是一个单独的思想，则用类表示它。
- [B] 若你能认为"它"是一个单独的实体，则用类的一个对象表示它。
- [C] 若两个类具有某些重要的共同之处，则用一基类反映它们。程序中大多数类都具有某些共通之处，故应存在一个(近似)全称基类，设计它时要非常慎重。

[2] 当你定义一个类而它不是用来实现象矩阵和复数等数学实体或者象链表等低级类型时：

- [A] 不要使用全程数据。
- [B] 不要使用全程（非成员）函数。
- [C] 不要使用公有数据成员。
- [D] 除非为了避免[A]、[B]或[C]，否则不要使用友元。
- [E] 不要直接访问另一对象的数据成员。
- [F] 不要在类中放置"类型域"，应使用虚函数。
- [G] 除了要着重进行优化以外，不要使用内联（inline）函数。

对C程序员的注释

对C越精通，似乎越难以避免用C风格编写C++程序，因而将丢失C++的某些潜在优点。

因此，请认真看一看参考手册中“与C之不同点”一节（§r.15）。这儿少量介绍了几个方面，用C++做它们比用C做具有更多的方便之处。宏定义（#define）在C++中几乎不需要：可用const（§2.4.6）或enum（§2.4.7）来定义常量和用inline（§1.12）可避免函数调用的开销。试一试声明所有的函数，并指定所有参数的类型——几乎没有更好的理由不这样做。与之类似 几乎没有更好的理由去声明一局部变量而不初始化它。这是由于声明能出现在语句可出现的任何地方——不要在你不需它之前去声明一个变量。不要使用malloc（），而new运算符（§3.2.6）能使这一工作做得更好。多数联合不需要名字——你可以试一试无名联合（§2.5.2）。

参考资料

本书没有几本直接参考资料，这里只列出了一个简短的教材或论文清单，它们在本书里直接或间接地提及。

[1] A.V.Aho, J.E.Hopcroft & J.D.Ullman, "Data Structures and Algorithms, "Addison-Wesley, Reading, Massachusetts, 1983.

[2] O.J.Dahl, B.Myrchaug and K.Nygaard, "Simula Common Base Language, "Norwegian Computing Center S-22, Oslo, Norway, 1970.

[3] O.J.Dahl and C.A.R.Hoare, "Hierarchical Program Construction, " in "Structured Programming, "Academic Press, New York, 1972, pp174-220.

[4] A.Goldberg and D.Robson,"SMALLTALK-80:The Language and its Implementation, "Addison Wesley, Reading, Massachusetts, 1983.

[5] R.E.Griswold et al., "The Snobol 4 Programming Language, "PrenticeHill, Englewood Cliffs, New Jersey, 1970.

[6] R.E.Griswold and M.T.Grisworld, "The ICON Programming Language, "Prentice-Hill, Englewood Cliffs, New Jersey, 1983.

[7] Brian W.Kernighan and Dennis M.Ritchie, "The C Programming Language, "Prentice-Hill, Englewood Cliffs, New Jersey, 1978.

[8] George Orwell, "1984, "Secker and Warburg. London, 1949.

[9] Martin Richards and Colin Whitby-strevens, "BCPL-The Language and its Compiler, "Cambridge University Press, 1980.

[10] L.Rosler, "Preliminary Draft Proposed Standard--The C Language, " X3 Secretariat: Computer and Business Equipment Manufactures Association, 311 First Street, NW Suite 500, Washington, DC 20001, USA.

[11] L.Rosler, "The Evolution of C-Past and Future, " AT&T Bell Laboratories Technical Journal, Vol.63 No.8 Part 2, Oct. 1984 , pp1685-1700.

[12] Ravi Sethi, "Uniform Syntax for Type Expressions and Declarations," Software Practice & Experience, Vol.11 (1981) , pp623-628.

[13] Bjarne Stroustrup, "Adding Classes to C: An Exercise in Language Evolution," Software Practice & Experience, 13 (1983), pp139-161.

[14] P.M.Woodward and S.G.Bond, "Algol 68-R User Guide," Her Majesty's Stationery Office, London, 1974.

[15] "UNIX System V Release 2.0, User Reference Manual," AT&T Bell Laboraries, Murray Hill, New Jersey, Dec. 1983.

[16] "UNIX Time-Sharing System: Programmer's Manual,Research Version, Eighth Edition," AT&T Bell Laboratories, Murray Hill, New Jersey, Feb. 1985.

[17] "UNIX Programmer's Manual," 4.2 Berkeley Software Distriution University of California, Berkeley, California, Mar. 1984.

目　录

第 1 章　C++ 概述

本章快速浏览C++程序设计语言的主要特征。先拿出一个C++程序，看它如何编译和运行，再看它如何读入并产生输出。这样介绍之后，本章的第三节开始讲述C++比较常见的特征：基本类型、声明（declarations）、表达式、语句、函数以及程序结构。其他内容介绍C++的其他设施：新类型定义、数据隐藏、用户定义的运算符以及用户定义的类型体系。

1.1 引言

本章将引导你见识一些C++程序和某些程序片断，这样你就对C++的设施有了大概的印象，并且有了写简单程序的必要知识。要知道这些概念的精确完整的解释，哪怕是最小的完整实例都需要写上几页定义。为了使本章不致于陷于手册或泛泛讨论，所用的例子都只是为提到的术语作最简短的定义。当讨论到较大例题时，有些术语将再回过来作深入的解释。

1.1.1 输出

我们首先写一个完整程序，它显示一行输出：

```
#include <stream.h>
main ( )
{
  cout ≪"Hello, world\n";
}
```

#include <stream.h> 那一行指示编译程序去包容（include)标准的流(stream)输入输出设施的声明，这些设施可在stream.h文件中找到。没有这些声明，表达式cout ≪"Hello, world\n"就没有什么意义。运算符≪（“送到”）是把它的第二个参数写到第一个参数中去（本例中是把串"Hello, world"写到标准输出流cout中）。串是用双引号括起来的字符序列。串中的反斜杠符\紧跟着一字符表示是单一的特殊字符，本例中，\n是换行字符，也就是所显示的字符是 Hello,world 并另起一行。

程序的其余部分

```
main ( ) {...}
```

定义了名为main的函数，每一个程序都必须有一个命名为main的函数，程序由此函数开始执行。

1.1.2 编译

输出流cout输往哪里？实现输出运算符≪的代码从哪里来？C++程序必须编译并产生可执行代码；编译过程和C基本上是一样的，大部分程序是共同的。程序正文读入后作分析，如果没有发现错误就生成目标码。程序先查找那些已经使用但尚未定义的名字和运算符（本例是cout和≪），然后，从库中补足程序中缺少的定义（所谓库，指标准库，用户自己也可以提供库）。本例中cout和≪是在stream.h中声明的，亦即：给出了它们的类型，但未提供它们实现的详细细节。标准库包含了cout的初始化代码和空间占用的规格说

明，以及≪代码。当然，标准库还要做在stream.h中声明了的许多其他事情，但为补足我们的程序，仅将所需的库的子集加到它的编译版本中去。

C++的编译命令是大写CC，和C程序的命令小写cc用法一样，其细节可参看你的手册。假定"Hello, world"程序存放在叫做hello.c的文件中，你可以按下述命令编译和运行它（$是系统提示符）：

```
$ CC hello.c
$ a.out
Hello, world
$
```

a.out是可执行的编译结果的缺省名；使用-o选择，你就可以给你的程序的目标码命名：

```
$ CC hello.c -o hello
$ hello
Hello, world
$
```

1.1.3 输入

按下述转换程序的提示（尽管很噜嗦），你可以键入一个英寸数，键入后，它将打印相应的厘米数。

```
#include <stream.h>

main ( )
{
  int inch = 0;
  cout ≪ "inches=";
  cin  ≫ inch;
  cout ≪ inch;
  cout ≪ "in="
  cout ≪ inch*2.54;
  cout ≪ " cm\n";
}
```

main ( ) 的第一行声明了一个整型变量inch，它的值是利用≫运算符（"取自"），从标准的输入流cin中读入的。cin和≫的声明当然可在 <stream.h> 中找到。本程序运行后，在你的终端上可看到：

```
$ a.out
inches=12
12 in = 30.48 cm
$
```

本例是每个输出都对应有一个语句，这确实很噜嗦。输出运算符≪可以用到它产生的结果上，故此，后四句输出操作能改写为一句：

```
cout≪inch≪" in="≪inch*2.54≪"cm\n";
```

在后面的章节里，将以较大篇幅讲述输入和输出。事实上，本章都在力图说明用不提供输入和输出运算符的语言如何写出上述程序！前面给出的程序实际上是通过使用库和#include文件等C++“扩充的”I/O操作写出的。换句话说，按参考手册上讲的C++语言并未定义输入和输出设施，程序员仅有的可用设施只是定义了运算符≫和≪。

1.2 注释

插入程序正文的注释常常只是为了人们阅读方便，编译程序并不理它。C++中有两种方式写注释。

双字符/*开始注释，以双字符*/终止。整个注释序列等价于一个空白字符（就象空格一样）。对于多行注释和不编辑的代码，这是有用的，但要注意/*、*/注释不能嵌套。

双字符//也开始注释，该注释在它所在行结束处终止。同样，整个注释序列也等价于一个空白字符，它多半用于短注释。

1.3 类型和声明

每一名字和每一表达式都有一个类型，类型决定了对它进行的操作，例如，声明(declaration)

```
int inch;
```

指明inch是整型的，即inch是一个整型变量。

声明是一个语句，它把名字引入程序。声明指定该名字的类型。类型定义了一个名字或者一个表达式的用法。整型定义了+、-、*和/等运算。程序在包含了stream.h之后，int就成为≪的第二操作数，它的第一个参数是ostream。

一个对象的类型不仅决定了能在其上的操作，而且还确定了这些操作的意义。例如语句

```
cout≪inch≪"in="≪inch*2.54≪"cm\n";
```

就能正确地区分四个输出值。串按字母打印，而整数inch和浮点值inch*2.54都从它们的内部表示转换为人眼可识的字符表示形式。

C++有几种基本类型，并有多种途径去创建新类型。下一小节介绍C++的类型，后面章节讲更有意思的类型。

1.3.1 基本类型

与硬件设施最直接相关的基本类型是：

```
char short int long float double
```

前四种类型用于表示整数，后两种表示浮点数。一个char类型变量的固有大小为给定机器上表示一个字符的大小（最典型的是字节（byte）），一个int类型的变量，其固有大小为给定机器上整型算术所用大小（典型情况是字（word））。整数的值域由其类型所定义的大小来表示。C++中，以char的大小作为度量其他大小的因子，所以按定义char的大小为1。各基本类型之间大小关系可写出如下：