第1单元 C++概览

第1课 C++入门

目的：介绍C++程序设计语言及其环境。

即使你不曾编过计算机程序，本书所介绍的内容也足够初学者学习C++ 语言所需的一切基础知识。写计算机程序不同于使用。当今的绝大多数字处理器易于学习、使用，但大量的工作是深入研究、开发它们。字处理器只是程序设计工作的例子之一。许多程序员花费了数小时、数天和数周书写人们使用的程序。

C++程序设计语言是当今最新的程序设计语言之一。它基于C程序设计语言，是C的改进版本，因而成为新一代程序设计语言。

本课介绍C++程序设计语言以及一般的程序设计概念。读者将会从中了解到有关C++的历史，以及C++与C程序设计语言的异同点。

1.1 程序设计入门

程序只不过是告诉计算机做什么的一组指令。计算机只是一台“愚蠢”的机器，它不会思考，只能按顺序执行人们书写的程序。

解惑：对计算机来说，程序就像烹调用的食谱。食谱并不比烹调的程序（一系列指令）多什么东西。食谱的结果是完成的菜碟，而计算机程序的结果是一种应用程序，如字处理器或工资程序。自然而然，计算机并不知道字处理器是什么样的。不过，通过遵照程序员书写的详细的指令系列，来执行做字处理所需的动作。

如果希望计算机做家务预算、通讯录，或玩“纸牌”游戏的话，就必须提供一个程序，以便它知道如何去做这些事情。这样的程序可以去买也可自己写。

写自己的程序有很多优点。因为它们能准确地做你希望做的一切。对需要使用的每一个程序，虽然都自己写是愚蠢的，但有一些应用程序确实太专用，以致不容易找出刚好能满足要求的程序。

想一想：公司通常雇佣自己的程序员书写用在该公司内的所有程序。当公司获得一套计算机系统时，它又不希望改变做商务的方式。那么这时程序员必须设计和书写正好适用该公司的程序。

1.2 程序设计过程

为了把C++程序设计指令送到计算机里，需要一个编辑器和一个C++编译器。编辑器类似于字处理器。它能使你把一个C++程序键入到内存，能够修改程序（如移动、拷贝和

插入文本),以及永久地保存该程序到一个磁盘文件上。在使用编辑器键入程序之后,必须在运行之前用C++编译器对它进行编译。

C++程序设计语言是一种编译型语言。编译器(在运行C++程序之前必须要有C++程序设计语言)接受C++语言指令,并把它翻译成计算机能读的形式。C++ 编译器是一种计算机用来理解程序中C++语言指令的工具。许多编译器附带有自己的编辑器(两种最流行的PC C++编译程序是:Borland 的 Turbo C++和 Microsoft 的C++编译程序,它们都带有自己的编辑器与集成程序设计环境)。如果编译器包含一个编辑器,那么该程序设计环境就具有较高的集成度。

由于本书讲授通用的AT&T C++标准程序设计语言,不打算介绍任何编辑器或与特定编译器有关的命令。在市场上有多种C++编译器的不同版本,我们不可能在一本书中介绍它们每个特定的内容。AT&T是推出C++语言的第一个公司。所以,在过去的几年里,AT&T的C++就是事实上的C++语言标准。

只要书写符合AT&T规范的程序(像本书所有的程序那样),它们便可以在市场上几乎任意一个C++编译器上工作(如果把大量与编译器有关的内部函数放到你的程序代码中,则会背离AT&T标准)。绝大多数编译器开发公司会告诉用户该公司开发的编译程序与AT&T C++的哪一版本兼容。只要你的编译器符合AT&T C++ 2.1或以上版本,本书中的所有程序都可以很好地工作。

解惑:程序也称代码。在写好C++代码之后,就可以使用C++编译器的编译指令进行编译,并运行该程序。这个程序的输出就是它的结果。程序的用户(有时是程序员)是指使用该程序但很少关心(或不关心)产生该输出的底层程序内容的那些人。

图1.1是一个书写和执行一个C++程序所需步骤的示意图。

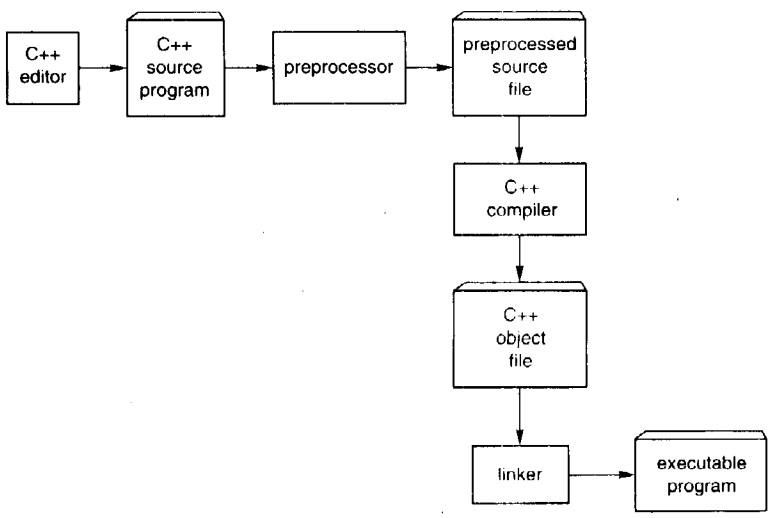

图1.1 书写和执行C++程序所需的步骤

注意,C++程序在被编译之前必须经过预处理器处理。预处理器读程序中的预处理器指令控制该程序的编译过程。C++编译器自动地执行预处理器步骤,所以除了把预处理器

指令放在程序内之外，你不必额外学习更多的东西。本书将在第7课“预处理器指令”中讲授两个主要的预处理器指令。

试一试：请查阅你的编译器的参考手册或询问你公司的数据处理人员，学会如何使用自己的C++程序设计环境编译程序。编译器的手册常常包含如何使用编辑器和编译用户程序的简短辅导性材料。学习编译器的整个程序设计环境，包括编辑器的所有输入和输出和编译器的选项，并不像学习C++语言那么重要。编译器只是把程序从源程序(C++指令)变换成一个可执行文件。

正如图1.1所示，一个程序必须在经过了编译的最后一步后，在运行之前，还要经过所谓的连接一步。在连接程序时，称作为“连接器”的程序给被编译的程序提供所需的运行时信息。也可以通过连接它们将几个编译过的程序组合成一个可执行程序。然而，多数情况下，全部的连接工作由编译器来做(Borland 和 Microsoft 的C++编译程序都自动地做连接)。在书写高级应用程序之前，几乎不必关心连接这一步。

1.3 检查错误

由于你是在给机器键入指令，所以一定要非常仔细。如果拼错一个字、遗漏一个引号或出现其他错误时，C++编译器就报出一条出错消息。最常见的错误是语法错，通常意味着拼错了一个字。

当你编译自己的程序并且有错时，编译器会告诉你是什么样的错误。如果该程序属键入错，则它不会通过连接编辑器发送出错消息。所以，必须读编译器的报错消息，弄清楚属什么问题，并且返回到编辑器状态改正错误。

程序错通常称为“故障”(bug)。如果不理解错误消息，那么必须查一查编译器参考手册或查找你的程序代码直至找到出错的代码行为止。错误定位与改正的过程称为调试。

注意：编译器不会捕捉到程序的全部错误。有时逻辑错误会不知不觉地隐藏在代码中。逻辑错要比键入错更难找出，因为编译器不会告诉有关它们的情况。例如，如果写一个打印工资单的程序，但告诉计算机打印所有负额工资单，计算机会服从该指令。显然，指令本身就不正确了，这种错误就属于逻辑错误。

本书始终提供查找程序中错误的机会。经过一、两次课后，就会看到用查错标志示意的查找不正确的程序。这是培养读者调试技能的好机会。

1.4 C++与其他语言的比较

如果你以前编过程序，就会理解C++与其他程序设计语言的不同之处。C++是一种效率高且比其“前辈”C有更强的类型检查语言。C被称为弱类型语言。换句话说，变量数据类型不必非得拥有相同的数据类型(函数原型和强制类型转换有助于减轻这个问题)。

例如，如果声明一个整型变量且给它赋一个字符值，C允许这样做。数据可以不是所期望的格式，但C能按最好的方面去处理。这是与诸如COBOL和Pascal这样的强类型语言最大的差别之处。

解惑：倘若上述讨论使你有些迷惑不解，请不必着急。有关这些专题的内容在随后几课中将详细介绍并提供许多例子加以说明。

C++是一种设计精巧且只有48条命令(称关键字)的程序块结构的程序设计语言。为了弥补其较少的关键字，C++又是拥有最多运算符的语言之一(仅次于APL程序设计语言)，如+，-和&&。C++的大量运算符可诱导程序员去书写只有少量代码的"加密"程序。不过，本书强调程序的可读性比减少程序代码量更重要。在保持程序可读性的同时，还将介绍如何最大程度地使用C++运算符。

C++的大量运算符(几乎等于关键字数)要求更合理地使用运算符优先级表。附录D包含了完整的C++运算符表。与运算符只有4到5层优先级的大多数语言不同的是，C++有17层(有些书组合了几层只产生15层)。在学习C++时，必须掌握这17层的每一种。这虽不像所说的那样困难，但其重要性并没被夸大。

C++也没有输入/输出语句。也就是说，它没有执行输入/输出的命令。这就是为什么C++能在众多不同的计算机上都可用的最重要的原因之一。大多数语言的I/O(输入/输出)语句都与具体硬件有关。例如，BASIC几乎有20条I/O命令，包括写到屏幕、打印机、调制解调器等命令。如果在微机上写一个BASIC程序，在不作修改的情况下，很有可能不能在大型机上运行。

我们可以通过大量使用运算符和函数调用，来执行C++的输入和输出。每个C++编译器都带有标准的I/O函数库。I/O函数与硬件无关，这意味着它们可以在符合AT&T C++标准的任何设备和计算机上工作。

要想彻底精通C++，就得了解比大多数其他语言所需的更多的计算机硬件知识。你无疑不必是一位硬件专家，但了解内部数据表示可以更有效、更有意义地使用C++。

如果熟悉二进制和十六进制数，也很有好处。在开始学习C++之前，你也许应该读附录B，它是有关这方面专题的辅助材料。即使你不想学习这些专题，仍然可以成为一个好的C++程序员，但知道了"底层"是什么东西，在学习C++时会更有意义。

1.5 C++和微机

C语言在有微机版本之前，相对来说鲜为人知。随着微机的发明与发展，C已成为世界流行的计算机语言，而C++进一步扩充了其在微机的使用。《C++程序设计教程》的大多数读者可能都将工作在基于微机的C++系统上。典型的微机称为PC，它起源于广泛使用的IBM PC。早期的PC没有为政府和大型商业机构所用的大型计算机的存储能力。尽管如此，PC的拥有者仍然需要有一种能在这类机器上编程的工作方式。BASIC是用在PC机上的第一种程序设计语言。几年以后，其他的一些语言从较大型的计算机上移植到PC机。不过，还没有一种语言能像C那样成功地成为世界范围内的标准程序设计语言。C++似乎将成为下一个标准。AT&T在1981年推出C++，在过去的三年里，C++已成为正在使用的最流行的程序设计语言之一。

本书的第2课"C++程序"将直接把你带入C++程序设计的语言世界里，以便能尽可能迅速地开始书写程序。

第 2 课　C++程序

目的：使读者熟悉 C++的程序结构。

本课给出几个 C++示例程序。这些程序可以使读者熟悉简单的 C++程序的整体外观，不要强求自己看懂每一行程序。事实上，越是不关心每个程序的细节，就越能学好本课。

2.1　第一个 C++程序

下面是一个 C++程序例子。虽然这个程序相当简单，但它却包含了一个合法的 C++程序的所有必要元素。

```
// Filename: CFIRST.CPP
// This program displays a message on the screen
#include <iostream.h>
void main()
{
    cout << "I will be a C++ expert!"; }
```

即使是一个简单的 C++程序，也可能会吓倒初学者。但决不要把你吓回去！只要你坚持下来，定会取得成功。如果把这个程序敲到 C++编辑器中，然后编译并运行它（通常是在下拉式菜单中选择 Run），则在屏幕上将会看到下面的输出内容：

I will be a C++ expert!

现在，先不要关心此程序代码的细节。本书的其余部分将会予以解释。在整个 7 行程序中，只有一行是值得注意的（即以 cout 开始的行），其余部分是为构造程序而用的。

想一想：上面的程序中包含了 7 行代码，而这 7 行中只有一行能够产生我们看得到的东西，更高级的 C++程序可能包含 500 行甚至更多。“7 比 1”，这样的“构造行与工作行”的比率并不总是出现在每个 C++程序中。否则，它会导致程序员要做太多的工作。随着程序量的增长，用于构造的代码数量也会随之减少。

2.2　C++的特殊字符

C++是少数几种几乎可以使用键盘上每个键的程序语言之一。它非常挑剔按下的键。注意上一节中的程序包含左和右花括号{和}。而如果使用圆括号(和)或方括号[和]代替花括号，则 C++会发出出错信息。

请特别小心地使用可用的字符集。计算机是非常精确的机器，它并不像人们那样对敲入的含糊字符持宽容态度。在通读完《C++程序设计教程》后，你将学会何时使用每个字符及其含义。在此之前，要特别仔细地敲入正确的字符。

C++区分大写字母 O 和数字 0（零），而小写字母 l 也不能替换数字 1。由于是与机器

打交道，因此当C++希望数字时，应敲入数字，而当C++需要字母时，应敲入字母。以及在需要的情况下敲入确切的特殊字符(即那些既不是字母也不是数字的字符，如括号和加号等)。

试一试：把左边的特殊字符用一条直线与右边的描述相匹配，如下所示：

特殊字符	描述
[	反斜线
<	左方括号
}	右尖括号
|	右圆括号
\	前斜线(或斜线)
]	左尖括号
{	左圆括号
)	右花括号
(	竖线
>	左花括号
/	右方括号

2.3 自由格式风格

大多数情形下，可以在C++程序中放入许多空格，而C++编译器并不会报错。C++程序员经常在程序中放许多额外的空格和空行以使得程序更加可读。这些额外的空白部分称为空白空间。利用这些空白，C++程序员可以使人而不是使C++编译器更加可读。

对于C++编译器来说，下面的程序代码与前面的程序代码完全相同：

```
// Filename: CFIRST.CPP //Program displays a message on the
          //screen
#include <iostream.h>
void main(){cout << "I will be a C++ expert!";}
```

对你来说，哪一个程序更可读呢？第一个版本还是第二个版本？很显然是第一个版本。C++被称为是自由格式的程序设计语言。自由格式意味着编译器允许在任何所希望的地方插入空格和空行。程序行既可采取缩进格式，也可采取左对齐格式。

计算机本身是一台机器，而它勿需额外的空白空间来读懂程序。一旦遵循了C++的所有代码规则，则编译器将会非常高兴地接受你所提供的代码。尽管C++ 编译器并不关心程序外表上是否美观，但你自己却要特别关心这一点。在一些相似的代码行前后增加几行空行可使程序更易于阅读和理解。

注意：在读到本书中的其他程序时，要留意某些C++的空白空间约定，并开发自己的某些约定。

想一想：在写C++程序的时候，就要考虑到某一天你自己(或与你一起工作的人)可能会不得不改变那些程序，你可以尽可能地挤出一些程序空间供它用，但这样做，将会一无所获(可能会节省几个计算机存储字符，但不足以弥补可读性差的程序所带来的弊病)。

如果添加了额外的空白行以使得程序更加可读的话，则程序在以后将会很容易更改。在

这个变化无穷的世界上，为适应这些变化不得不更改程序，届时，修改可读性好的代码比修改可读性差的代码的工作要快得多。更新和修改程序称为维护程序。可维护的程序是一个可读的程序。

解惑：如果你现在感到困惑，这很正常！你不必搞懂上面两个程序的细节，因为本课的目的是使你熟悉C++程序的整体外观，而不是它们的细节。如果你已经理解了C++非常在意所敲入的字符，并且认识到程序对人们来讲应该是可读的，那么你就可以对迄今为止所学课程获得一个A+分。

2.4　大写字母和小写字母

虽然C++不在意空白空间，但却严格区分大写字母和小写字母之间的不同。大多数情形下，C++更喜欢小写字母。C++对小写字母的偏爱使得它有别于大多数其他的程序设计语言。对于许多程序设计语言来说，下面的语句是等同的：

```
if (netpay > grosspay)
if (Netpay > GrossPay)
IF (NETPAY > GROSSPAY)
```

而对于C++来说，这三行是完全不同的。在学习C++语言时，你将看到何时使用小写字母以及何时使用大写字母。而在大多数时候，是用小写字母编程的。

C++包含固定的命令词汇，有时称为关键字，也称为保留字。附录A列出了C++的全部命令。命令是C++认识的部分有限词汇。例如，将一个值从程序中的一个位置转换到另一个位置的命令称为return。你必须像所有C++的其他命令一样将return小写。

技巧：在学习C++命令时，要经常参考附录A，特别是第2单元中“使用数据编程”中开始的一些专用命令。

如果希望将消息打印到屏幕或打印机上，则消息本身可采用大写、小写或二者的混合。例如，前面显示到屏幕上的消息：

```
I will be a C++ expert!
```

由于这是一条给程序用户读的消息，因而希望C++用常规的大写和小写字母将其打印出来。因为消息不是关键字，因此对它没有小写的限制。

想一想：在继续下面的学习之前，需要复习一下前面的几节。C++要求命令用小写字母，并要求键入的特殊字符绝对正确。然而，空白空间完全是另一码事。C++并不关心为了可读性的目的而添加了多少空白空间。

2.5　较长的程序

前面给出的示例程序非常简单。而某些C++程序需要几十万行的代码。初露头角的作者是不会从事《战争与和平》的续集创作的。同样地，新的C++程序员也不应从事巨大的编程项目。目前所写的大多数程序相对来说应是非常小的，也许只有10到100行的代码。

即使是一个大的程序，通常也并不是保存在磁盘上一个文件中。程序员通常是将大的

程序分成几个小程序。而这些小程序做象预制板，在需要的时候将它们组合在一起以处理某类应用问题。

试一试：下面是一个比前面所见到的C++程序要稍长一些的程序。先不要品味其细节。大致看一下程序，并逐渐习惯C++认识的不同的特殊字符。

```
// Filename: 1STLONG.CPP
// Longer C++ program that demonstrates comments,
// variables, constants, and simple input/output
#include <iostream.h>
void main()
{
    int i, j; // These three lines declare four variables
    char c;
    float x;

    i = 4; // i is assigned an integer literal
    j = i + 7; // j is assigned the result of a computation
    c = 'A'; // Enclose all character literals
             // in single quote marks

    x = 9.087; // x is a floating-point value
    x = x * 12.3; // Overwrite what was in x with something else

    // Sends the values of the four variables to the screen
    cout << i << ", " << j << ", " << c << ", " << x << "\n";

    return; // Not required, but helpful
}
```

下面几课将深入地讨论本程序中的命令。现在只是给你灌输可读性很重要的思想。对C++编译器来说，下面的程序与上面的程序一样，但对以后做程序维护的人来说，却是截然不同的。

```
// Filename: 1STLONG.CPP // Longer C++ program that demonstrates
// comments, variables, constants, and simple input/output
#include <iostream.h>
void main(){int i, j; // These three lines declare four variables
char c;float x;i = 4; // i is assigned an integer literal
j = i + 7; // j is assigned the result of a computation
c = 'A';        // Enclose all character literals// in single quote
                //marks
x = 9.087; // x is a floating-point value
x = x * 12.3; // Overwrite what was in x with something else
// Sends the values of the four variables to the screen
cout << i << ", " << j << ", " << c << ", " << x << "\n";return;
// Not required, but helpful
}
```

2.6 问答题

1. 程序维护的意思是什么？
2. C++程序中的某些行并不产生结果，但出于其他目的却是需要的，那么它们是用

来做什么的呢？

3. C＋＋关心输入或删除程序中的空白空间吗？

4. C＋＋关心程序中是否混杂有大写和小写字母吗？它何时关心，何时不关心？

第3课 C++注解

目的：使读者领会C++注解的作用。

在第2课“C++程序”中，我们解释了空白空间的重要性。程序不光是给计算机用的，还要给你自己或其他人某一天可能进行修改而用。不管在C++ 程序中插入多少空白空间，其可读性仍然远远比不上英语，并且某些程序会相当复杂。

尽管如此，我们仍然可以通过在程序中插入大量的注解和消息，而进一步地改进程序的可维护性。最好是一开始写几行程序时就加上注解，好的习惯应尽早养成。

3.1 注解的重要性

假定你的汽车在开出一半路程的时候突然出了故障，这时又没有其他的汽车开过来。现在的问题是手边没有工具或零件，它们都放在工具箱里，而你对汽车的知识又一无所知。这时当你打开工具箱时，对零件放在哪儿以及如何排除故障茫然无措。

正在你趋于绝望时，突然发现工具箱底部放着两本汽车修理的书。第一本是修理工用的专业书籍，它所用的技术术语是你从未听到的，你扔掉了这本对自己毫无价值的书，转而拿起了第二本。这本书的书名是《非汽车修理工指南》。在粗略地翻阅几页以后，你笑了起来。第二本书假定你对汽车修理知识知之甚少，并采用清晰的，非技术性语言和十分友好的风格解答了如何修理汽车故障的问题。

你发现第二本包含了与第一本书完全相同的事实。然而，第一本书无法帮你解除困境，它既没有教你什么又没有向你解释什么，因为它假定了你自己都知道如何写这本书。而第二本书却用了极易懂的语言解释每一个概念。只花10分钟，你就立刻修好了汽车，并驾驶它到餐厅去用餐(这时如果是在沙漠中或别的什么地方，你会怎么想呢?)。

3.2 程序的注解

上面那个汽车的故事也许要：

(1) 从此故事引出一个观点。
(2) 本书印刷出现了错误。
(3) 相信本书的作者好长时间都没有抓住主题。

很明显，它引出一种观点，即人们更喜欢接受非技术性的材料。当然，如果解释得相当好的话，任何东西都是很容易接受的。如果专家们把热核反应空间传递原理用一种非常简单的方式解释的话，你也可以把它搞懂。

C++程序不大好懂，即使是对编制它的程序员来说也不例外。在第2课“C++ 程序”中，我们解释了程序很少保持一种格式。程序员的一生中不仅包括写新的程序，而且还包括

更改以前写的程序。就像在写一个C++程序时，添加空白空间以使之易读一样，在程序中添加注解也同样是重要的。

注解不是C++命令。事实上，C++忽略程序中所有的注解。它不过是解释程序要干什么的消息。注解是给人用的，而不是给计算机。

想一想：如果你的C++程序只包含C++代码而没有注解的话，那么以后将很难把它搞清楚。要记住，C++是非常难于读懂的。许多雇用程序员的公司都要求他们在自己写的程序中加上注解。想想这些公司为什么要这样要求呢？这是因为人员在变、工作在变，而公司必须要保证由一个人所写的程序能够被另一个人所理解。

3.3 C++注解的语法

在计算机术语中，“语法”代表命令的拼写形式、特殊字符的顺序以及语言元素的放置。在学习了C++命令或操作的语法后，也就学习了你需要它做的，而它又认识的格式。

既然注解这样重要，那么我们就要在学习任何C++命令之前先学习它的语法（即怎样写它才能使C++知道它们是注解。在程序中加入大量的注解以便人们在以后读你的程序时能够有一个清楚的思路，这一点至关重要。C++ 程序员要追踪你的代码并弄清程序的意图，而注解更能加速这一进程。人们越是更快地读懂你的代码，也就越能更快地做一些必要的修改使之适应于其他的工程项目。

一个注解以两行向前的斜线开始，有时称为双斜线。该注解一直持续到行尾。换句话说，不能在同一行中放置命令、注解，然后再是另一个命令。下面是一个带有一注解的C++例子：

```
return (( a>b)? a:b); // Grabs the larger of two values
```

下面是同样一个没有注解的语句行：

```
return ((a>b)? a:b;
```

利用上面的注解，即便你不知道C++，也能够知道该语句是干什么的。正如你所看到的，这个语句看起来就像一些毫无意义的杂乱字符。

双斜线非常重要，没有它，即使 return ((a>b)? a:b)是一个合法的语句，但它仍然拒绝接受该行。如果忘记了写双斜线符号，则C++将读进Grabs the larger of two values，并不知道用它们来做什么。简而言之，C++不认识英语，它只懂C++。注解不是给C++用的，而是给看程序的人用的。

注解可以出现在程序中所有的行上。有时注解可能比程序的语句还要多。注解对于描述一段程序以及程序员本人和整个程序都是非常有用的。例如，下面的一小段C++程序含两行占据整行的注解，还含有三行位于代码右边的注解。

```
// Filename: COMAVGE.CPP
// Program to compute the average of three values
#include <iostream.h>
void main()
{
    float g1, g2, g3; // Variables to hold student grades
```

```
    cout << "What grade did the first student get? ";
    cin >> g1;
    cout << "What grade did the second student get? ";
    cin >> g2;
    cout << "What grade did the third student get? ";
    cin >> g3;
    float avg = (g1 + g2 + g3) / 3.0; // Compute average
    cout << "The student average is " << avg; // Print average
    return;
}
```

本书附带一张含所有程序的软盘片，这些程序是按文件名存储的。在把程序调入内存并运行它之前必须先告诉C编译器其文件名是什么。为了能更快地试一试本书中的例子，在每一程序的第一行都包含了与软盘完全相同的文件名。例如，对前面的例子，可以通过在软盘中检索到名为COMAVGE.CPP的文件，然后把它装进来。

许多公司都要求程序员将自己的名字写在所写程序开始处的注解中。如果以后有人对该程序有疑问，则可以很快找到原来的程序员。如果几个人要改动程序，那么他们也要将每人的名字加上，对其改动加上较简短的附注。由于所有这些行都是注解，则C++会自动跳过它们。

另外，在程序中也要随时随处插入注解。如果写了一些难以理解的代码，则一定要多加几行注解，并在需要的时候，解释一下下几行代码的作用。

想一想：你现在应感到非常骄傲，因为你在一点儿都不懂C++ 代码的情况下，却已知道前面几行程序的作用。这就是注解的用处之所在。它们是程序的文档，它可使你不必完全搞懂那些冗长乏味的代码就能知道程序在做什么。

查错：下面的程序包含三个注解错误，你看一看能否把它们找出来。

```
// This program computes taxes
#include <iostream.h>
void main()
{
    The next few lines calculate payroll taxes
    // Computes the gross pay float gross = 40 * 5.25;
    float taxes = gross * .40; / Just computed the taxes
    cout "The taxes are " << taxes;
    return;
}
```

因为你还不会用C++编程，因此也还不知道程序是干什么的，但应该有种感觉，即在3个地方有错。第一个问题出现在第5行，该行是The next few lines。在此行之前，应该有双斜线。该行是用英语解释的，它不是C++代码，因此它应该是一注解。

第二个问题是第6行。注解应总是以双斜线开始并延伸到行尾。C++将忽略注解后面的计算。它认为整行都是给人们看的英语消息。如果你把该计算部分与注解分开的话，则C++将会按照要求去做了。

最后一个问题非常容易。以float taxes=gross开始的第8行，这一行的注解没有正确开始，按照规则"单斜线并不是注解符"。

解惑：一般来说，C++程序员本人即使是没有一点注解，也还是能够读懂其程序的。一旦你学习了C++语言，那么就能够在没有注解的情况下读懂C++代码，并在需要的时候予以改变。注解不过是为了帮助描述程序是干什么的。

3.4 为自己写注解

假定你写的程序是为自己使用和娱乐而用的，除了你以外没有人会再去读这些代码了。你想过没有你是否还需花费一些时间在自己的程序中添加注解呢？有很多理由需要这样做。

假定你写的是自己的银行记帐程序，一年以后，银行允许你从存单中自动支付票据了。那么旧程序就不再适用于新的银行系统了。这时，就拿出C++ 代码开始修改。然而，你已不记得原来做了什么，而代码又太简练很难搞懂。所幸的是：在代码中放入了一些注解。这样就可以直接读懂程序并找到需要改动的地方。这里，由于你知道了代码要做什么，因而可迅速地做出修改。

要养成在写程序的同时并加上注解的习惯。许多人不写注解，而想着自己以后再加上。事实上，往往做不到。那么当程序需要维护的时候，与那些在最初编程期间添加注解的程序相比，它要花费两倍的时间用于修改上。

查错：下面是与前一课结束时相同的程序，只有一点不同，程序员忘记了用双斜线标识注解。在编译时，有20多条错误信息，可能是漏掉什么了。看一看你是否能帮助程序员在需要的地方插入正确的注解来改正错误。

```
// Filename: 1STLONG.CPP
Longer C++ program that demonstrates comments,
variables, constants, and simple input/output
#include <iostream.h>
void main()
{
    int i, j;          These three lines declare four variables
    char c;
    float x;

    i = 4;             i is assigned an integer literal
    j = i + 7;         j is assigned the result of a computation
    c = 'A';           Enclose all character literals
                       in single quote marks

    x = 9.087;         x is a floating-point value

    x = x * 12.3;  Overwrite what was in x with something else

        Sends the values of the four variables to the screen
cout << i << ", " << j << ", " << c << ", " << x << "\n";

    return;            Not required, but helpful
}
```

在翻回看第2课“C++程序”中正确的程序之前，数一数放在程序中的双斜线数。如果没有添加12个双斜线(或者如果添了多于12个双斜线的话)，那么再试一遍。

3.5 使用良好的判断

与注解是非常重要的一样，在程序中减少一些没有必要的注解也是重要的。代码中的注解只有在需要的时候才添加。本书中前面几个程序的注解比后面要多。当学习了简单的命令后，本书将试图通过在程序中加入额外的注解而简化它们。

不管怎样，冗余的注解与根本没有注解都同样糟糕。例如，即使你一点儿不懂C++命令，也会同意这种观点，下面的代码包含了没有价值的注解：

```
totalsales = oldsales + newsales;// Adds the old sales and
                                 // the new sales to get
                                 // total sales
cout << "Happy Birthday";      // Sends the Happy Birthday
    // message to the cout
return; //Return
```

这些注解中的每一个都是冗余的。它们没有真正地解释除了C++ 代码本身以外做了什么。然而，考虑这个语句：

```
for (int ctr = 10; ctr >0; ctr--) {     // prints the numbers
cout << ctr << endl;}                   // from 10 to 1
                                        // on-screen
```

即使你不懂C++，但也可以看到这两行代码的目的被隐藏在难懂的C++语言中。C++程序员能够讲清楚这个代码，但注解却使其隐秘全无。正如前面的三个代码段，注解经常多于一行。如果一行放不下，则它可以延续到下一行。

查错：请看下面的5行代码，哪一个包含有用的注解，哪一个包含无用的注解。

```
clog << '\n'; //Sends an end-of-line to the error log
radius3 = radius1 + radius2; //Calculates radius3
// The following code contains a C++ program
// The following code tracks your investments
clog << '\n'; //Sends '\n' to clog
```

3.6 C风格的注解

C++支持另外一种以前你可能见过的注解。C++是基于C语言的，但其注解的语法不同于C。C++的设计者们决定保持旧的C风格的注解语法，以便C程序在C++ 编译器下可以做尽量少的甚至是不加改变地通过。尽管如此，双斜线标识的注解更优越，不过为以防万一，也应该学一点儿C语言注解。

C的注解以字/*开始而以*/结尾。不同于C++注解，你必须为C注解做结尾标记。如果没有放入标记*/，则C将假定下一行(或下面的100行)仍然是注解，直到它找到另一个*/为止。下面一行是C风格的注解：

```
Char name[25]; /* Reserves space for a 25-character name */
```

由于C的注解只有在到达*/时才终止，所以下面的三行构成一个注解：

```
/* The following program calculates stock statistics
using the most modern technical analysis techniques
available. */
```

当然,这三行也可以如下形式:

```
/* The following program calculates stock statistics */
/* using the most modern technical analysis techniques */
/* available. */
```

尽管你应该熟悉C的注解,但真正的C++程序员应尽量避免这样写。双斜线更容易,因为没有必要记住注解的结尾。C风格的注解是很容易出错的。如果在一个C风格的注解中又嵌入另一个注解的话,则C++将会被搞混淆。尽可能使用C++的注解,这样不管是你自己还是C++编译器都将会使其更加健壮并更有生命力。

3.7 问答题

1. 什么是注解?
2. 注解是对C++编译器有帮助呢?还是对C++ 程序员或者两者兼而有之都有帮助呢?
3. 判别真假:C++注解以双斜线开始并以下一个C++命令终止。
4. 为什么C++注解优于C的注解?
5. 为什么不应在每一行中都加入注解?

3.8 练习题

1. 写一个有两行的C++注解,它解释了一个存款程序的工作。
2. 将练习1的注解转换成C格式的注解。

第4课 C++程序的格式

目的:解释C++程序的组成部分。

C++ 程序可以各不相同 ,甚至差别很大 ,但它们都拥有相似的结构 。正如第 2 课“C++程序 ”所解释的那样,其程序中的某一部分只是为建立它的其他部分而用的。无论是写一个支票簿、工资单程序,还是化学分析程序,都需要理解 C++ 程序的整体结构。本节课对 C++程序片段作一些解释,以便在进入第 5 课 “C++数字数据”学习之前能对程序格式有所了解。

4.1 C++程序的格式

所有C++程序都有一个公共部分:main()函数。前几节课已给出了一些C++程序。即使你大致看一下这些程序,也会看到每个程序中都有 main()。进一步,还会看到在 main()之后有一个{括号和稍后有一个}括号。决不会找到只有左{括号而无右}括号的情况。

C++程序还拥有另一个公共特征:几乎所有程序都有一行或多行#include。图 4.1 给出了C++程序的简单轮廓。

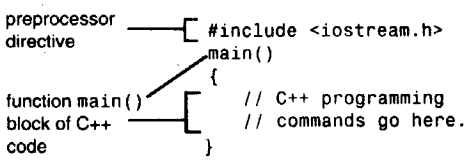

图 4.1 一个简单的C++程序轮廓

想一想:为什么每个左{括号必须配对一个右}括号?在程序中还有每个左(括号配对一个右)括号,以及每个左 < 括号配对一个右 >括号。(),{}和< >括住C++程序中的语言成分。正如在书面语言中使用圆括号那样,这些符号括住了程序段,必须在该段结束处用终止符表明结束。

现在,虽然你还没有理解 main(),#include,{ } 以及C++程序的所有其他部分,但已不会觉得生疏。下一节考虑一个较长的C++程序并对它进行分片,使读者更易于步入C++的世界中。

4.2 程序分片

下面提供一个接近于考试的程序。其目的是想在读者进入第 5 课“C++数字数据”开始介绍具体语言元素之前,对C++程序有一个初步“轮廓”性理解。

```
// Filename: DISSECT.CPP
// Simple program for you to analyze
```

```
#include <iostream.h>

void main()
{
    int age, weight; // These lines define four variables
    char initial;
    float salary;

    age = 13; // Assign an integer literal to age
    weight = age * 6; // Assign a weight based on a formula
    initial = 'D'; // All character literals are enclosed in single quotes
    salary = 200.50;      // salary requires a floating-point value
                          // because it was defined to be a floating-point
    salary = salary * 2.5; // Change what was in salary

    // The next line sends the values of the variables to the screen
    cout << age << ", " << weight << ", " << initial
        << ", " << salary;

    return; // Good habit to end programs and functions with return
}
```

请记住这个程序很简单，它并不实际执行有意义的工作。尽管它简单，你也可能在没读后面的解释之前还不能透彻地理解它，但应该看到，这毕竟是学习 C++ 的一个良好开端。

试一试：在浏览完这个程序后，看看是否能理解该程序的某一部分或全部。倘若你是新学程序设计的，应该理解计算机是这样处理程序的：从第 1 行开始读每行程序并作相应的工作，直到完成该程序中的全部指令（每行上的语句）。（当然，在见到程序从计算机上输出之前首先得编译和连接该程序，这正如第 1 课“C++入门”所描述的那样）。

下面是运行这个程序的输出结果：

```
13, 78, D, 501.25
```

4.2.1　程序

经过第 3 课“C++注解”的学习，你已知道程序中前 2 行的内容，它们包含描述文件名和该程序有关信息的注解。注解也可分散在整个代码中以解释有关部分。

下面这一行称为预处理指令：

```
#include <iostream.h>
```

这条看起来有些奇怪的语句并不是一条 C++命令，而是 C++编译器的指令。该指令支配着编译器（这种说法易于记住）。#include 告诉 C++ 读出尖括号内的文件名，并把该文件装到 C++程序的这个位置。换句话来说，该程序的长度就变成 DISSECT.CPP（前一节给出的程序）的长度加上 iostream.h 的长度。用 #include 打头的绝大多数包含文件是以.h 为扩展名结束的，因为这些文件是头文件。

名为 iostream.h 的头文件来自 C++编译器。它有助于保证稍后用 cout 产生的输出正常地工作。第 7 课“预处理器指令”将更详细地解释 #include。

想一想：你可能见过除 iostream.h 之外的其他一些头文件。有时 C++ 程序员使用 C 程序名为 stdio.h 的头文件来替代 iostream.h。注意，附录 A 中 #include 没有像其他 C++命令那样被列出来，因为它不是一条 C++命令，而是一条简单的预备命令（因而得其名：