第一章　C++的历史

C++是多年来追求完美的编程语言中的顶峰。当在将来确实要开发新的和更好的语言时，C++代表了当前的"最高水平"。C++的历史就是对更好的编程方式的探索。

在学习C++之前，弄清楚C++是怎样来的和是何因素促使它的诞生很重要。明白了导致C++的发展的力量和原因后，就会认识到它的许多独特的特性和品质。

本章的目的是提供一个简短的C++编程语言的历史、起源与其祖先(C)的关系、使用和它支持的编程哲学。本章还把C++与其他编程语言作比较。

1.1　C++的起源

C++的历史从C开始。原因很简单，C++是建立在C的基础之上。实际上，C++是C的一个超集(实际情况是全部C++编译器都可用来编译C程序!)。特别地，C++是扩展和加强后的引入了面向对象编程哲学的C语言新版本(面向对象编程在下一章描述)。C++还包括几个对C语言的改进。然而，大多数C++的精神和风格都是直接由C而来。要全面理解和欣赏C++，需要理解C的"如何和为什么"。

1.1.1　C的诞生

C语言震惊了计算机界。因为它从根本上改变了编程方式的探讨和思维，其影响是不可低估的。C被许多人认为是第一个现代的"程序员的语言"。在C语言的发明之前，计算机语言设计通常是为学术练习，或者是由官方的委员会来进行。C就不一样，它是由实际工作的程序员设计、补充和发展，反映的是他们探讨编程工作的方式　它的特性经过实际使用这个语言的人琢磨、测试、思考和再思考。这个过程的结果就是程序员喜欢使用它。确实，C很快吸引了许多对其有近于宗教热情的拥护者，并在程序员圈子里发现它被广泛而快速地接收。一句话，C是一种由程序员也是为程序员设计的语言。

C最初由Dennis Ritchie发明并用UNIX操作系统在DEC PDP－11上实现。C是以一种较老的语言BCPL为开始的发展过程的结果。BCPL由Martin Richards开发。BCPL影响了一种叫B的语言，这种语言由Ken Thompson发明，并于七十年代导致了C的发展。

多年来，事实上的C标准是由UNIX Ver5操作系统支持，并在Brian Kernignan和Dennis Ritchie的《C编程语言》(Prentice—Hall，1978)中描述的。随着个人计算机的出现，产生了许多C的实现，C变得很流行。然而，因为没有标准存在，在这些C的实现中存在着差异。为了改变这种状况，在1983年夏初成立了一个委员会来一次性地制定C语言的ANSI(美国国家标准学会)标准。最后一版标准于1989年12月采纳，1990初就有了最初的副本。ANSI标准C是建立C++的基础。

最初也许不易理解，C 通常被称为"中级"计算机语言。把中级这个词用于 C，并不是否定的含义，并不是指 C 不够强大，难于使用，或者不如"高级"语言先进，也不是指它像汇编语言不好使用(汇编语言，通常也称为汇编程序，只是表示计算机能执行的机器代码的符号)。把 C 看作中级语言，是因为它兼有像 Pascal 和 Modula－2 这些高级语言的因素和汇编程序的功能性。

从理论的角度，高级语言试图为程序员提供其可能明型的每一件东西，并使下语言内。低级语言除了提供访问实际机器的指令外没有别的。中级语言给程序提供一套简洁的工具，允许程序员根据自己的情况来开发出高级的结构。中级语言为程序员提供内建的功能以及灵活性。

作为中级语言，C 允许操纵位、字节、地址以及作为计算机组成部分的端口，也就是 C 并不试图把程序和机器的硬件进行很大程度的隔离。例如，不像许多高级语言能直接操作字符串来实现大量的串操纵，C 只能对单个字符进行操作，而且大多数高级语言有读写磁盘文件的内建语句，C 中，所有这些过程都是调用库例程来实现，而严格地讲，库例程并不是语言的一部分。这种办法增加了 C 的灵活性。尽管所有的 ANSI 标准都支持有磁盘 I/O 能力的函数，但只要愿意就可以忽略这些函数。

C 确实需要程序员定义例程来实现高级的操作，这些例程称为函数，对 C 语言相当重要。实际上，函数是 C 和 C++的构造块。可以很容易地制作一个函数库来实现程序中用到的不同任务。从这种意义上讲，可以把 C 个人化来满足自己的需要。

还有 C 的另一个方面需要明白，因为这对于 C++也很重要：C 是一种结构化的语言。结构化的语言最明显的特征是使用块。块是逻辑上有联系的一组语句。例如，假设一个 IF 语句将成功的执行五个分离的语句。如果这些语句可以组成一组并作为一个单元来调用，这些语句就形成一个块。

结构化的语句允许一系列的编程可能性，支持带有局部变量的子程序概念。局部变量就是只有在其内部进行定义的子程序认识它。结构化的语言还支持几处循环结构，如 while、do—while 和 for。但是对 goto 语句的使用是禁止的或不鼓励的，并不是像传统的 BASIC 和 FORTRAN 中那么普遍的程序控制。结构化的语言允许语句缩进而不要求严格的域概念(早期版来的 FORTRAN 要求这样)。

最后，也许是最重要的，C 是一种奇特的语言。C 的基本哲学是程序员是变化的，而不是语言。因此，C 可以让程序员做想做的任何事情，即使想让它做的事是异端的、危险的或者可疑的，C 可以对机器进行完全的控制。当然这个功能所带来的责任必须由程序员来承担。

1.1.2 理解对 C++的需要

有了前面的讨论，也许想知道为什么要发明 C++。既然 C 是一个成功的和有用的计算机编程语言，为什么还需要别的？答案很复杂。在编程的历史过程中，程序复杂性的增加就驱使了对管理这种复杂性的更好办法的需求。C++就是对这种需求的响应。要更好地理解这种关系，考虑下面的情况。

计算机发明以来，编程手段已发生了戏剧性的变化，变化的最初原因是要容纳程序复杂性的增加。例如，在计算机最初发明时，使用计算机的前端面板触发二进制机器指令来编程。

对于只有几百个指令的程序，这种方法还可以。程序增大后，发明了汇编程序，程序员可用表示机器指令的符号来处理更长的复杂性增加的程序。随着程序的继续增大，就发展了高级语言，给程序员更多的工具来处理其复杂情况。

第一个广为传播的语言当然是 FORTRAN。FORTRAN 是非常感人的第一步，它几乎就是一种鼓励编写清楚而又易于理解的程序的语言。六十年代诞生了结构化的编程，这种方法受到了如 C 这些语言的鼓励。最初，结构化的语言使得较容易地写出中等复杂的程序成为可能。然而，即使有了结构化的编程方法，一旦工程文件达到一定的大小，其复杂性就超出了程序员能管理的范围。目前，许多工程文件接近或达到了结构化的方法已无法正常工作的地步。为了解决这个问题，一种新的编程方法出现了，这种方法称为面向对象的编程(简称 OOP)。将会看到，C++就是 C 的面向对象版本。

最后的分析表明，尽管 C 是世界上最受喜爱，最为广泛使用的专业编程语言，还是有其能力所能处理的复杂性有限的时候。一旦程序达到 25,000 至 100,000 行代码，就变得相当复杂而难于从总体上进行把握。C++的目的就是要打破这种局限，帮助程序员理解和管理更大更复杂的程序。

1.1.3 C++的诞生

为了满足管理巨大的复杂性需要，C++诞生了。它是于 1980 年由 Bjarne Stroustrup 在新泽西州 Murray Hill 的贝尔试验室发明的。他最初把这种新语言称为“带类的 C”，但于 1983 年更名为 C++。

C++包括整个 C 语言。如前面提到的，C 是建立 C++的基础。C++包括 C 的全部特征、属性和优点，还坚持 C 的程序员是变化的而不是语言是变化的这一哲学。在这一点是，关键是要注解 C++的发明并不是要产生一种新的编程语言，而是对已经非常成功的语言的加强。

Stroustrup 对 C 所做大多数增加都支持面向对象的编程。Stroustrup 声明 C++的一些面向对象特征被其他的面向对象的语言如 Simula67 所吸收。因此，C++代表了两种有影响的编程方法的融合。

自 C++第一次发明以来，已经历了三次主要的修订。第一次在 1985 年。第二次在 1989 年。第三次修订发生在 C++的 ANSI 标准工作开始时。第一次提出的标准草案于 1994 年 1 月 25 日出台。ANSI C++委员会(作者为成员之一)全部保留了 Stroustrup 首次定义的全部特性并添加了几个新的。

标准化过程是一个典型的缓慢过程，在 C++标准最后采用之前可能需要几年的时间。因此要有 C++还是“正在发展”的概念，还有一点特性还会发展和添加到 C++上。但是本书提供的材料是稳定的。适用于当代所有的 C++编译器，遵从当前提出的 C++的 ANSI 标准。因此，可以放心地学习这本书。

在 C++发明时，Bjarne Stroustrup 知道保留 C 的原有精神很重要，包括它的高效、灵活性和哲学，同时添加了对面向对象编程的支持。所幸的是，他的目标达到了。C++仍然给程序员提供 C 的自由与控制，还有对象的功能。用 Stroustrup 的话来说，C++中面向对象的特性就是“程序的结构具有明析、可扩展性和易于维护，面不失高效性。”

尽管 C++最初是设计来帮助管理太大的程序，但并不局限于这种用途。实际上，C++

的面对象的属性可有效地应用于任何编程任务。对于C++用户诸如编辑器、数据库、个人文件系统和通信程序等工程文件也并非不常见。同时因为C++具有C的高效性，许多高性能的系统软件也是由C++构造。

需要记住的重点是：因为C++是C的一个超集，只要能用C++编程，也就可以用C编程。这样，实际上可以同时学习两种编程语言，所作的努力与学习一模一样。

1.2 什么是面向对象的编程

因为面向对象的编程是发展C++的基础，准确地定义什么是面向对象的编程很重要。面向对象的编程采用了结构化编程的最好思想，并把它们和几个鼓励程序以新的方式达到编程任务的有力的新概念相结合。进行面向对象风格编程时，把一个问题分解成相关部件的子组，用这些子组来考虑与每一组有关的代码和数据。再把这些子组组织为一个层次化的结构。最后，把这些子组翻译成称作对象的自主式单元。

所有面向对象的语言都有三个共有的东西：封装、多态性和继承。虽然要在本书的后面详细解释这些，还要先简单地看一下这些概念。

1.2.1 封装

也许已经知道，所有程序都由两个基本要素组成：程序语句(代码)和数据。代码是程序实现其行为的部分，数据是这些行为所影响的信息。封装是把代码和代码操纵的数据进行联编的编程机制，保持接口外面的安全和防止误用。

面向对象的语言中，以生成自主式黑匣子的方式来把代码和数据联编在一起。在黑匣子内是全部必需的数据和代码。代码和数据以这种方式链接后，就生成了一个对象。换句话说，对象就是支持封装的手段。

对象内，代码、数据或者两者都可是对象私有或公用的。私有代码和对象仅被对象的另外的部分识别和访问，也就是对象的私有部分不能被对象外的程序部分访问。代码或对象为公用的时，即使是在对象内定义的，程序的其他部分也能访问。典型地，对象的公用部分给对象的私有部分提供一个被控接口。

1.2.2 多态性

多态性(希腊语为"多个格式")是允许一个接口用于一通用类的行为的特性。特殊的行为是由状态的实际本质所决定的。一个简单的多态例子是在汽车的方向盘上。不管实际的转向机构是什么，方向盘(即接口)是一样的。也就是不管汽车的转向机构是手动的、动力的还是齿轮齿条的，方向盘都一样地工作。因此，只要知道怎样操作方向盘，就能驾驶各种类型的汽车。同样的原理可应用于编程。例如，考虑一个堆栈(先进后出的表)，可能有一个程序需要三个不同类型的堆栈。一个堆栈用于整型值，一个用于浮点型值，另一个有于字符。这种情况下，尽管存于栈中的数据不同，但实际每个栈算法却是一样的。在非面向对象的语言中，将要求生成三套不同的堆栈例程，以不同的名来调用，每一个有不同的接口。然而由于多态性，在C++中可只生成一个用于三种不同情况的例程(一个接口)。这种方式下，只要知道怎样使用一个堆栈，就能使用于各种情况。

更一般的情况，多态性的概念用“一个接口，多个方法”来表达。意思是可为一组相关的活动设计一个通用的接口。通过允许使用同一接口来指定一个通用行为类来帮助降低复杂性。在用于各种情况时，由计算机来选择这种指定的行为(即方法)。程序员不必手工地选择，只需记住和使用这种通用接口。

最初的面向对象编程语言是解释器，因此多态性自然在运行时间支持。但C++是一个编译语言，在C++中运行时间和编译时间都支持多态。

1.2.3 继承

继承是一个对象可从另一个对象获得其特性的过程。这一点很重要的原因是它支持层次分类的概念。思考一下这个问题，大多数知识都可由层次(自顶向下)分类进行管理。例如，一个好吃的红苹果是苹果分类的一部分，苹果又是水果类的一部分，水果又在更大的食物类的下面。也就是食物类具有某些品质(可食用，有营养等)，这些品质逻辑上又适用于其子类水果。除了这些品质，水果类又有特殊的特征(有汁，有甜味等)以区别于其他食物。苹果类定义了一个苹果才有的特征(生长于树上，在非热带地区等)，好吃的红苹果就继承它前面这些类的品质，只需定义表现其独特性的某些品质。

不使用继承，每个对象都必须显示地定义其特征。然而使用继承后，一个对象只需定义那些表现其独特于其他对象的品质，它可从其父处继承通用属性。这样，继承机制就可只用一个对象来作为多个一般情况的特殊实例。

1.2.4 C++实现OOP

进一步学习本书将会发现C++有的许多特征提供对封装、多态性和继承的支持。但要记住，可用C++以任何方式来写任何形式的程序。C++支持面向对象编程的事实并不意味着只能写面向对象的程序。和其前辈C一样，C++最好的一个优点是其灵活性。

1.3 将需要什么

要有效地使用此书，需要一台计算机和一个C++编译器。学习的最好方法是自己来试一下例子，用自己的变化来进行实验。动手进行实验是无法代替的。

如果使用PC，就有几种C++编译器可选用，保证使用一种时新的C++编译器。此书中的代码用Borland C++和Mirosoft C++测试过，但也可在任何当代的C++编译器上工作。

现在已知道了C++的历史，可以开始学习使用C++了。下一章介绍C++基础。

第二章　C++概览

学习编程程言最困难的事情就是语言没有一个元素是独立存在的。而且，语言各部分的工作是相互联系的。也就是在进入下一章考查用来示例语言各方面的举例程序之前，有必要先对C++程序的构成有一个一般的概念，包括一些基本控制结构、运算符和函数。为了这个目的，本章给出了C++程序的一般性情况，示出了几个简单的例子。并不深入到细节，而是把重点放在任何C++程序都有的一般性概念上。还介绍几乎所有C++程序都使用的结构。这里考查的大多数主题在下一章中讨论得更详细。

因为学习最好是通过做来完成，建议在计算机上调试这些例子程序。

2.1　第一个C++程序

在接触理论前，先来看一个简单的C++程序，输入并编译下列代码：

```
/* Program #1 - A first C++ program.

   Enter this program, then compile and run it.
*/

#include <iostream.h>

// main() is where program execution begins.
main()
{
  cout << "This is my first C++ program.";

  return 0;
}
```

运行时，这个程序在屏幕上显示下列文本：

```
This is my first C++ program
```

在继续学习之前，有必要定义两个名词。第一个是源代码。源代码就是人能读懂的程序形式，上面的清单就是一个源代码例子。程序的可执行形式称为目标代码或可执行代码。目标代码是在编译程序时生成的。

[术语：源代码是程序员生成的程序形式，目标代码是计算机可执行的程序形式。]

2.1.1　一行一行地解释

看一下这个程序的每一行，首先，程序以这些行开始

```
/* Program # 1 - A first C++ Program.
   Enter this program, then compile and run it.
*/
```

这是注释。和其他大多数编程语言一样，C++允许在程序的源代码中加入评注，编译器忽略注释的内容。注释的作用就是向读这个源代码的人描述或解释程序的操作。这个例子中的注释是对程序的标识。在比较复杂的程序中，将用注释来解释程序的特征和它是怎样工作的。换句话说，可用注释来提供一个对程序所做的工作的现场描述。

[术语：注释是嵌入程序中的评注。]

C++中，有两种类型的注释。刚才看到的称为多行注释，这种注释以/*(一个斜杠符加一个星号)开头，只有遇到*/才结束。这两个注释符之间的一切内容都将被忽略。第二种类型的注释将在这个程序的稍后看到，将进行简单的讨论。

下一行代码如下：

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

C++语言定义了几个称为头文件的文件，所含的内容对于程序来说是必需的或者是有用的。对于这个程序，需要文件 iostream.h(它用来支持C++的I/O系统)。这个文件由编译器提供。本书的后面将学到有关头文件更多的知识及它们为什么重要。

程序的下一行为

```
// main() is where program execution begins.
```

这一行所示为C++用到的第二种注释：单行注释。单行注释以//开始，在行末结束。典型地，C++程序员在写较大详细的注释时使用多行注释，而在需要小评注时使用单行注释。但这也与个人风格有关。

下一行如前面的注释所示，是程序执行的开始。

```
main()
```

所有C++程序都由一个或多个函数组成(不太严格地说，函数就是子程序)。每个C++函数必须有一个名称，而每个C++程序都必须包含的函数是如上所示的 main()。main()函数是程序开始和(大多情况下)结束的地方(严格地讲，C++程序以调用 main()函数开始，而大多数情况下在 main()返回时结束)。跟在 main()后面的左花括号标识 main()函数代码的开始。

[术语：main()是程序开始执行的地方。]

程序的下一行是

```
cout<<"This is my first C++ program.";
```

这是一句控制台输出语句，把消息 This is my first C++ program. 显示在屏幕上。这是用C++的标准输出操作符<<来完成的。<<操作符把它右边的任何表达方式输出到它左边指定的设备上。cout 是预定义的代表控制台输出的标识符，(最一般的情况)是向计算机的屏幕。这样，这个语句就使消息在屏幕上输出。注意这个语句以一个分号结束。实际上所有的C++语句都以分号结束。

消息"This is my first C++ program."是一个字符串。C++中，字符串是由双引号括起来的一个字符序列。可看到，C++中将频繁用到字符串。

程序中下一行为

```
return 0;
```

这一行终止 main()，使其把值返回 0 给调用过程(典型情况是操作系统)。对于大多数操作系统，返回 0 值表示程序正常终止(其他值表示程序由于错误而终止)。Return 是一个

C++的关键字,用来从函数返回一个值(return 将在本书后面详细讨论)。严格地,从main()的返回值是可选的,但要合乎需要。一般地,所有程序在正常终止中都要返回 0(即没有错误)。

程序末尾的右花括号正式地结束程序。尽管这个花括号实际上并不是程序目标代码的一部分,但概念上可以认为程序执行到 main()的右花括号时C++程序就结束。实际上,如果 return 语句不是这个例子程序的一部分,程序在遇到这个右花括号时也会自动结束。

2.2 处理错误

如果还没有输入、编译和运行前面的程序,就请做一下。从前面这个编程实践中可以知道,很容易在把代码输入计算机时敲入一些不正确的东西。所幸的是,如果在程序中输入了什么不正确的东西,编译器在试图编译这个程序时将报告语法上的错误消息。许多C++编译器都企图弄懂源代码而不顾所写的是什么东西,因此被告的错误可能并没有反映问题的实际起因。例如上面这个程序,如果偶然丢失了 main()函数后的花括号可能导致编译器报告说 cout 语句是语法错误源。这一点说明,在看到语法错误时,还要准备看一下程序的上面几行,以便能找到错误。

许多C++编译器不仅报告实际的错误,也提出警告。C++语言设计得比较宽容,允许对任何句法上正确的语句进行编译。然而有一些东西虽在句法上很正确,却相当可疑。当编译器遇到这种情况时就打印出一条警告。作为程序员,就可以来裁定这种怀疑是否公正。坦率地讲,一些编译器太热心了,会在非常正确的C++语句处作出警告的标志。也有一些编译器允许打开不同的选项来报告想知道的有关程序的信息。有时尽管没有什么值得警告的,这个消息还是以警告的形式报告出来。本书的程序遵从 ANSI 提出的C++标准,只要正确输入,就不会产生任何有麻烦的警告消息。

提示 尽管所有的C++编译都报告致命的错误,但大多数C++编译器提供几个层次的错误(和警告)报告。一般地,可以选择所希望的某种类型的错误报告。例如,大多数编译器提供报告选项,如低效率的构造函数或过时的特征。对于本书中的例子,希望使用编译器的缺省(或"常规")错误报告。但应该查一下编译器的用户手册看看根据配置应作什么选项。许多编译器有复杂的特性,能帮助找到细微的错而不致于成为大问题。花时间和作出努力来理解编译器的错误报告系统是值得的。

2.3 第二个举例程序

除程序的一般结构外,也许对于编译语言来说没有其他的结构有给一个变量赋值这么重要了。变量是可以赋一个值的有名称的存储位置。而且变量的值在程序执行过程中可以一次或多次改变。也就是说,变量的内容是可变的,而不是固定的。

下面的程序生成一个名为 value 的变量,给其赋值为 1023,然后在屏幕上显示消息 This program prints the value : 1023。

```
// Program #2 - Using a variable

#include <iostream.h>

main()
{
  int value; // this declares a variable

  value = 1023; // this assigns 1023 to value

  cout << "This program prints the value: ";
  cout << value; // This displays 1023

  return 0;
}
```

这个程序引入了两个新的概念。第一个语句：

```
int value;//this declares a variable
```

声明了一个名为 value 的整型变量。C++中，所有变量必须在使用前声明。而且，变量要保存的值的类型必须说明。这就叫作变量的类型。这个例子中，value 可以保存整型值。对于大多数编译器，这就意味着－32,768 至 32,767 的全部数值。C++中，把变量声明为整型时，在前面加关键字 int。后面就看到 C++支持许多内建的变量类型(还可以生成自己的数据类型)。

[术语：变量的类型决定了它能保存的值。]

第二个新特征在下一行代码中：

```
value=1023;//this assigns 1023 to value
```

如注释所说，这一句给 value 赋值为 1023。C++中，赋值运算符是一个等号，它把其右边的值复制到左边的变量中。赋值后，变量 value 将等于数 1023。

两个 cout 语句显示程序生成的两个输出。注意下面的语句如何用来显示 value 的值：

```
cout<<value;//This displays 1023
```

一般情况下，如果需显示一个变量的值，只需把它放在 cout 语句中<<的右边。对于这个例子，因为 value 含有数 1023，这就是要在屏幕上显示的数。在继续学习前，可以给 value 另外一个值，看一看结果。

2.4 一个更实用的例子

前两个举例程序介绍了几个C++语言的重要特征，但这两个程序不是非常有用。下一个举例程序完成了一件有意义的任务：把加仑转化为升。也示例了如何输入信息：

```
// This program converts gallons to liters.

#include <iostream.h>

main()
{
  int gallons, liters;

  cout << "Enter number of gallons: ";
  cin >> gallons; // this inputs from the user
```

```
    liters = gallons * 4; // convert to liters

    cout << "Liters: " << liters;

    return 0;
}
```

这个程序首先在屏幕上显示一个提示信息，等待输入一个加仑量的完整数字(记住，整型数不能有小数部分)。这个程序显示相当的大约升数。实际上10加仑有3.7854升，因为本例中使用整型数，因此把每加仑圆整为4升。例如，输入1加仑，程序以公制的相应量4升来作出反映。

这个程序中的第一件新事情是两个变量在int关键字后声明，以一种用逗号分隔开的列表形式。一般情况下，可用逗号隔开来声明任意个同类型的变量(另一种方法，程序可以用多个int语句来完成同样的任务)。

这个函数使用这个语句来输入由用户输入的值：

```
cin >>gallons ; // this inputs from the user
```

cin是C++编译器提供的另一个预定义的标识符，代表控制台输入(一般指从键盘输入)。输入操作符是>>符号。由用户输入的值(本例中必须是一个整数)被输入到>>右边的变量中(本例中为gallons)。

此程序还有新的东西，看下一行：

```
cout << "Liters:" << liters;
```

在一个输出语句中使用两个输出操作符。特别地，它输出字符串"Liters:"，后面跟着liters的值。一般地，在一个输出语句中，可以愿意多少就把多少个输出操作符串在一起，对于一个条目只使用一个<<。

2.5 一种新的数据类型

尽管加仑向升的换算程序对于大约的情况还比较好，但因其使用的是整数，在需要准确些答案时就有可能丢失一些所期望的东西。如前面提到的，整型数据类型不能表示任何小数部分的值。如果需要分数，就必须使用浮点数据类型，其中之一被称作float。这种类型数据的典型范围是3.4E－38到3.4E＋38。浮点数的运算保留了结果的全部小数部分，因此就提供了更准确的换算。

下面的换算程序版本使用了浮点值。

```
/* This program converts gallons to liters using
   floating point numbers. */

#include <iostream.h>

main()
{
  float gallons, liters;
```

```
cout << "Enter number of gallons: ";
cin >> gallons; // this inputs from the user

liters = gallons * 3.7854; // convert to liters

cout << "Liters: " << liters;
  return 0;
}
```

这个程序与前面的版本相比有两点变化。首先，gallons 和 liters 声明为 float。其次，换算系数指定为 3.7854，可有更精确的换算。只要 C++遇到一个带有小数点的数，就自动地知道这是一个浮点常量。另一点：注意与使用 int 变量的程序相比，cout 和 cin 语句没有变化。不管给出什么数据类型，C++的 I/O 系统自动地调整。

现在试一下这个程序。提示时输入 1 加仑，相当的升数就是 3.7854。

2.6 快速的复习

在继续学习前，复习一下已学过的最重要的东西：

(1) 所有 C++程序必须有一个 main()函数，这是程序执行开始的地方。

(2) 所有变量在使用前都要声明。

(3) C++支持许多数据类型，包括整型和浮点型。

(4) 输出操作符是<<，使用 cout 时，<<使信息在屏幕上显示。

(5) 输入操作符为>>，使用 cin 时，>>从键盘读入信息。

(6) 程序在遇到 main()的末尾时停止(或者执行 return 0;语句)。

2.7 函 数

C++程序是由称为函数的组成块构成的。函数是一个含有一个或多个 C++语句并实现一个或多个任务的子程序。写得比较好的 C++代码中，一个函数只完成一个任务。

[术语：函数是 C++程序的组成块。]

每个函数都有一个名称，并由这个名称来调用此函数。一般地，可以给出所喜欢的名称。但要记住，mian()是为开始程序的执行的函数而保留的。

C++中，一个函数不能嵌入另一个函数中。不像 Pascal，Modula－2 和其他一些编程语言，允许函数的嵌套，C++把所有的函数都看作一个单独的项目(当然，一个函数可以调用另一个)。

标识函数时，本书使用写 C++时比较通用的一个惯例。函数在其名字后面应该有括号。例如，函数的名为 getval，其名称用于一个句子中时将写作 getval()。这种标识法将在本书中把变量名和函数名区别开来。

第一个程序中，main()是唯一的一个函数。如前面已指出的，main()是程序开始执行时的第一个函数，它还必须包含在所有 C++程序中。程序中将会用到两种函数。每一种是自己写的，main()就是这种函数的一个例子。另一种是由编译器实现的，可从编译器的标准库

中找到(对标准库将进行简单的讨论,但一般地说,是一个预定义函数的集合)。所写的程序一般同时包含有自己生成的函数和由编译器提供的函数。

因为函数形成C++的基础,现在来进一步考查函数。

2.7.1 一个有两个函数的程序

下面的程序有两个函数:main()和myfunc()。运行这个程序(或读后面的描述)之前,仔细地看一下这个程序,想一下将在屏幕上显示什么。

```
/* This program contains two functions: main()
   and myfunc().
*/
#include <iostream.h>

void myfunc(); // myfunc's protoype

main()
{
  cout << "In main()";
  myfunc(); // call myfunc()
  cout << "Back in main()";

  return 0;
}

void myfunc()
{
   cout << " Inside myfunc() ";
}
```

这个程序如下工作:首先,main()开始,执行第一个cout语句。接着,main()调用myfunc()。注意是怎样实现的:函数名myfunc()后面跟着括号出现,以一个分号结束。函数调用是一个C++语句,因此后面跟有一个分号。下一步,myfunc()执行其cout语句,然后立即返回main()到紧跟函数调用的代码行。最后,main()执行其第二个cout语句后终止。因此,屏幕上的输出为:

```
In main() Inside myfunc() Back in main()
```

上面的程序中还有一个重要语句:

```
void myfunc(); // myfunc's prototype
```

如注释所讲的,这是myfunc()的原型。尽管将在本书后面详细讨论原型,这里必须讲几个新的词。函数原型的声明在其定义之前。原型让编译器知道函数的返回类型和函数可能有的参数个数和类型。编译器在第一次调用这个函数之前需要知道这些信息。这就是为什么原型出现在mian()之前。

[术语:原型在其第一次使用前声明一个函数。]

可以看到,myfunc()没有return语句。在原型和myfunc()定义之前的关键字void,从形式上说明myfunc()不返回值。C++中,不返回值的函数声明为void。

2.8 函数的参数

有可能给函数传递一个或多个值。传给函数的值称作参数。现在学习的程序中，还没有任何参数。特别地，上面例子中的 main()和 myfunc()都没有一个参数。但是，C++中的函数可能没有参数，也可能有许多参数。其上限取决于所用的编译器，但提出的 C++标准规定函数必须能带有 256 个参数。

[术语：参数是函数被调用时传给它的值。]

下面是一个使 C++标准库(即内建)函数 abs()的短程序，显示一个数的绝对值。abs()函数有一个参数，把参数转换为绝对值，返回结果。

```
// Use the abs() function.
#include <iostream.h>
#include <stdlib.h> // required by abs()

main()
{
  cout << abs(-10);

  return 0;
}
```

这里，值-10 作为参数传给 abs()。abs()函数接收被调用时使用的参数，返回其绝对值，本例中为 10。尽管 abs()只有一个参数，其他的函数却可有几个。这里的关键是，函数需要参数时，就通过在跟在函数名后面括号内的指定数值来传递。

abs()的返回值由 cout 语句来在屏幕上显示-10 的绝对值。通过这样工作的原因是，只要一个函数为较大的表达式的一部分，它就自动被调用来返回能获得的值。本例中，abs()的返回值变为<<操作符的右边部分，因此被显示在屏幕上。

注意上面程序中的另一件事，它还包含头文件 stdlib.h，这是 abs()所要求的头文件。一般地，只要使用了库函数，就必须包含其头文件。头文件在其他地方提供库函数的原型。

生成带有一个或多个参数的函数时，接收这些参数的变量也必须说明。这些变量称作函数的参数。例如，下一个示例的函数打印调用这个函数的两个整型参数的结果。

```
void mul(int x,int y)
{
  cout << x * y <<" ";
}
```

[术语：函数的参数就是函数定义来接收参数的变量。]

每一次的 mul()调用，都将把传给 x 的值乘以传给 y 的值。但要记住，x 和 y 只是调用函数时接收所用值的可操作变量。

考虑下面的短程序，示例了如何调用 mul()：

```
// A simple program that demonstrates mul().
```

```
#include <iostream.h>

void mul(int x, int y); // mul()'s prototype

main()
{
  mul(10, 20);
  mul(5, 6);
  mul(8, 9);

  return 0;
}

void mul(int x, int y)
{
  cout << x * y << " ";
}
```

这个程序将在屏幕上打印 200、30 和 72。调用 mul()时，C++编译器把每一个参数的值复制到匹配的参数。也就是在第一次调用 mul()中，100 被复制到 x，20 被复制到 y。第二次调用中，5 被复制到 x，6 被复制到 y。第三次调用中，8 被复制到 x，9 被复制到 y。

如果从未使用过允许参数化的函数的语言，上面这个过程看起来就有些奇怪。不必担心，看多了 C++程序的例子，参数、函数的参数和函数的概念就会变得清楚了。

记住　引用值的参数用来调用函数。接收参数的变量称为函数参数。实际上，带有参数的函数称作参数化的函数。

C++函数中，有两个或更多的参数时，用逗号分开。本书中，参数表将指用逗号分开的参数。mul()的参数表为 x，y。

2.8.1 函数返回值

许多将用到的 C++库函数都返回值。例如，前面用到的 abs()函数返回其参数的绝对值。自己写的函数也可能返回值给调用例程。C++中，使用 return 语句来返回值。return 的通用格式为：

```
return value;
```

这里 value 是要返回的值。

为了说明函数返回值的过程，前面的程序可写为下列形式。这个形式中，mul()函数返回其参数的结构。注意这个函数放在赋值语句的右边把其返回值赋给一个变量。

```
// Returning a value.

#include <iostream.h>

mul(int x, int y); // mul()'s prototype

main()
{
  int answer;
```

```
  answer = mul(10, 11); // assign return value
  cout << "The answer is " <<  answer;

  return 0;
}

/* This function returns a value. */
mul(int x, int y)
{
  return x * y; // return product of x and y
}
```

这个例子中，mul()用return语句返回x * y的值，然后这个值赋给answer。也就是由return返回的值变成调用例程中mul()的值。

因为mul()现在返回一个值，其前面不再有关键字void(记住，void只用于不返回值的函数)。和有不同类型的变量一样，也有不同类型的返回值。函数的返回类型在函数的原型和定义前面都出现。如果没有指明返回类型，就假定这个函数返回一个整型值。这样，因为没有指明其他返回类型，mul()就以缺省方式返回int类型。记住也可以显式地指明整型返回类型，如下所示：

```
/* This function returns a value */
int mul(int x,int y)
{
  return x * y; // return product of x and y
}
```

但是大多数C++程序员都不添这个麻烦来指明int，因为它将自动地指明。

也可以在return语句不与任何值相联系的情况下使用return语句来返回函数，但这种return的形式只用于那种没有返回值并声明为void的函数，也可以有不只一个return的返回。

2.8.2 main()函数

已经知道，main()函数比较特别，因为它是程序执行时第一个调用的函数，它标识了程序的开始。不像其他编程语言总是从程序的"顶部"开始执行，C++总是以调用main()函数来开始，不论这个函数放在程序中的什么位置(但以main()函数为第一个函数是比较好的形式，这样易于找到)。

程序中只能有一个main()。如果试图包含进多于一个的main()，程序将不知道从哪里开始执行，实际上，大多数编译器都会发现这个错误并报告。

2.8.3 C++函数的一般形式

上面的几个例子示出了几种特定类型的函数。但所有的C++函数都有一个通用的格式，如下所示：

```
返回类型    函数名(参数列表)
{
  .
  .函数体
```

}

进一步考查一下组成函数的不同部分。

函数的返回类型在缺省状态下是整型的。但将在本书后面看到，可以指明任何类型的返回类型。记住没有必须返回值的函数。如果不返回值，其返回类型为 void。如果返回一个值，这个值必须与函数的返回值兼容。

每一个函数必须有一个名称，名称后是括起来的参数列表。参数列表指明了将传给信息的变量的名称和类型。如果函数没有参数，括号就是空的。

接着，括号把函数体括起来。函数体由定义函数要做什么的 C++语句组成。函数到达右花括号或遇到一个 return 语句时终止并返回。

2.9 一些输出选项

到此，还没有机会来把输出推进到下一行——即执行一个回车换行序列。但是，这个需求很快就会出现。C++中，回车换行序列由换行字符来产生。要把换行字符放到字符串中，使用代码：\n(返斜杠跟一小写 n)。要看回车换行序列的例子，请试一下下面的程序。

```
/* This program demonstrates the \n code, which
   generates a new line.
*/
#include <iostream.h>

main()
{
  cout << "one\n";
  cout << "two\n";
  cout << "three";
  cout << "four";

return 0;
}
```

这个程序产生下列输出：

```
one
two
threefour
```

换行符可放在字符串中的任何位置，而不只在串尾。现在可以试一下回车换行符，要确保弄明白它是怎样工作的。

2.10 两个简单的命令

下面几章中开发的几个例子很有意义，必须弄明白在其最简单的形式中，有两个 C++命令：if 和 for。后面几章中，对这两个命令进行全面的研究。

2.10.1 if 语句

C++的 if 语句与其他任何语言的 IF 语句一样工作。其最简单的形式为

if(条件) 语句;

此处条件为结果要么为真要么为假的表达成。C++中,真为非零,假为零。如果条件为真,就执行这个语句;否则,不执行这个语句。下面的一段把 10 is less than 11 显示在屏幕上。

```
if (10<11) cout << "10 is less than 11";
```

[术语:if 在两条执行路径中作选择。]

比较运算符如<(小于)和>=(大于等于)与其他语言中的相似。但在C++中,等号运算符为==。下面的语句将不会执行,因为等于条件为假;即 10 不等于 11,语句不在屏幕上显示 hello。

```
if (10==11) cout << "hello";
```

当然,if 语句内的操作符不需为常量,可以是变量,甚至是函数调用。

下面的程序示例了一个 if 语句的例子。提示用户两个数,如果每一个值小于第二个值时显示一个报告。

```
// This program illustrates the if statement.

#include <iostream.h>

main()
{
  int a, b;

  cout << "enter first number: ";
  cin >> a;
  cout << "enter second number: ";
  cin >> b;

  if(a < b) cout << "First number is less than second.";

  return 0;
}
```

2.10.2 for 循环

for 循环用来把一个语句指定重复的次数。for 循环可像其语句言如 Pascal 和 BASIC 中的 FOR 循环一样操作。其最简形式为

for(初始值,条件,增量)语句;

此处初始值把循环控制变量设置一个初值。条件是每一次循环重复的表达式。只要条件为真(非零),循环继续进行。增量决定循环控制变量在每一次顺利地进行循环重复(即每一次条件的值为真)时如何增减。

[术语:for 是 C++中使用的循环语句的一种。]

例如,下面的程序在屏幕上打印数字 1 到 100。