第1章 C语言概述

要理解C++，就要理解导致它诞生的理由，使它成形的力量，它继承的传统。因此，C++的故事是从C开始的。在本章中，我们将介绍C语言的概貌、它的起源、它的使用情况和它的基本原理。因为C++是建立在C语言之上的，本章也将介绍C++产生的历史背景。

1.1 C语言的起源和历史

C语言是由Dennis Ritchie发明并首先在使用UNIX操作系统的DEC PDP-11计算机上实现的。C是一种较早的语言BCPL发展改进后的产物。BCPL是由Martin Richards开发的，它影响了由Ken Thompson发明的B语言，而B语言又导致了20世纪70年代C语言的发展。

许多年来，C语言事实上的标准是UNIX操作系统上配备的C语言版本。它最初是由Brian Kernighan和Dennis Ritchie在他们所著的“The C Programming Language”（Englewood Cliffs, N.J.: Prentice-Hall, 1978）一书中描述的。1983年夏，成立了一个委员会来创建定义C语言的ANSI (American National Standards Institute)标准。这个标准化的工作进行了六年（比当时我们预想的要长得多）。

1989年12月，ANSI C标准最终被采纳并于1990年初开始使用。这个标准也为ISO（International Standards Organization，国际标准化组织）所接受，其后的标准通常称为ANSI/ISO标准C。1995年，C的修改版1（Amendment 1 to the C）标准被采用，其中添加了几个新的库函数。1989年版本的C标准，与修改版1一道成为了标准C++的基本文档，定义了C++的C子集。1989年标准定义的C版本通常称为C89。

1989年后，C++作为主角登场，在20世纪90年代，对C++标准的开发吸引了许多程序员的注意，到1998年底，一个C++标准开始被采纳。然而，对C进行的工作在继续。最终的结果是1999年版的标准C，通常称为C99。一般来讲，C99保留了C89的几乎所有特征，本质上与C89所描述的一样。C99标准化委员会的重点在两个主要的领域：增加了几个数值库并开发了几个特殊用途、但是具有高度革命性的新特征，例如变长数组和restrict指针限定符。在一些情况中，起源于C++的特征，如单行注释，也被集成进C99中。因为C++标准是在C99创建之前完成的，C99的创新并没有在标准C++中出现。

1.1.1 C89与C99

尽管C99中的创新从计算机科学的角度来讲是重要的，现在它们并没有什么实际的效果，因为在写本书时，还没有广泛使用的编译器实现C99。是C89定义了大多数程序员认定并且所有的主流编译器识别的C。此外，正是C89形成了C++的C子集。尽管有几个新特征是由C99添加的，它们最终会集成进C++的下一个标准中，但现在这些新特征和C++是不兼容的。

因为C89是形成C++的C子集的标准，也因为它是大多数C程序员当前所知道的C版本，所有我们在第一部分讨论这个C版本。因此，当使用术语C时，应把它认为是由C89所定义的

C。然而，与C++相关的C89和C99之间的重要区别都会标出，例如，当C99添加了一个改进与C++的兼容性的特征时，就会标示出来。

1.2 C语言是中级语言

C语言经常被称为中级计算机语言。这不意味着C不够强大、很难使用或者比像BASIC或Pascal这样的高级语言更少在开发中使用，也不是暗指C具有像汇编语言那样的笨拙特性。C之所以被认为是中级语言，是因为它把高级语言中最好的元素和汇编语言的控制和灵活性结合起来了。表1.1显示了C在计算机语言中的地位。

作为中级语言，C支持对位、字节和地址这些计算机功能的基本元素进行操作。尽管如此，C代码也是可移植的。可移植性意味着很容易将为一种计算机或操作系统编写的软件加以修改而在另一种机器上实现。例如，如果能很容易地对为UNIX所写的程序加以修改，使之可以在Windows下运行，那么这个程序就是可移植的。

所有的高级语言都支持数据类型的概念。数据类型定义了一个变量能够存储的一组值和可对那个变量执行的一组运算。常见的数据类型有整型、字符型和实型。虽然C语言有五种基本的内嵌数据类型，它不是强类型语言，就像Pascal和Ada一样。C语言支持几乎所有的类型转换，例如，可以在一个表达式中混合字符和整型数据。

表1.1 C语言在计算机语言中的地位

高级	Ada
	Modula-2
	Pascal
	COBOL
	FORTRAN
	BASIC
中级	Java
	C#
	C++
	C
	Forth
	Macro-assembler
低级	汇编语言

不像高级语言，C语言几乎不支持诸如数组越界等运行时错误检查，这种检查是由程序员完成的。

同样，C语言并不要求在参数和变元之间类型完全兼容。从以往的编程经验中可以知道，高级计算机语言一般要求变元的类型与接受变元的参数的类型完全相同。然而，对C来讲，并不必如此。相反，C允许变元是任意类型的，只要它可以合理地转换为参数的类型。还有，C语言提供了所有的自动转换来实现这一点。

C语言的特点是它支持对位、字节、字和指针的直接操作。这使得它非常适用于系统级的编程，在系统级编程中经常见到这些操作。

C语言的另一个重要的特点是它仅有几个关键字，这些关键字是构成C语言的命令。例如，C89仅定义了32个关键字，C99又添加了5个。某些计算机语言有的关键字是C语言的几倍多，

比如，大多数版本的BASIC语言都有超过100个关键字。

1.3 C语言是结构化语言

从以往的编程经验中，你可能听说过应用于计算机语言的术语“块结构”（block-structured）。

尽管术语块结构语言并不严格地适用于C语言，但我们还是把C简单地称之为结构化语言，因为它与其他结构化语言，如ALGOL, Pascal和Modula-2有许多相似之处。

注意： C(和C++)语言从技术上讲不是块结构语言的原因是：块结构语言允许过程或函数在其他过程或函数内声明。然而，由于C不允许在函数内创建函数，所以不能称之为块结构语言。

结构化语言的一个显著特征是：代码和数据的分离。这是一种把所有信息和必要的指令与其他程序部分分离以执行特定任务的语言的能力。获得分离的一种方式是调用使用局部（临时）变量的子例程。使用局部变量，可以写出子例程，以便在子例程内出现的事件对程序其他部分不会产生副作用。这种功能使得程序更容易共享代码段。如果开发了一些分离的函数，你仅需要知道函数做什么，而不是怎么做。记住，过度使用全局变量（可以被整个程序访问的变量）会由于意外的副作用而把错误引入程序中（用标准BASIC语言编过程的人对此都深有体会）。

注意： C++使分离的概念进一步得到了扩展。特别地，在C++中，程序的一部分可以严格控制程序的其他部分中哪些是可以访问的。

结构化语言为设计者提供了许多程序设计功能。它直接支持一些循环结构，例如while, do-while和for。在结构化语言中，禁止或不提倡使用goto语句，goto语句也不是常见的程序控制形式（例如，像在标准BASIC和传统的FORTRAN中那样）。结构化语言允许你把语句放在一行上，并不要求严格的格式概念（像某些FORTRAN语言那样）。

下面是结构化和非结构化语言的一些例子。

非结构化语言	结构化语言
FORTRAN	Pascal
BASIC	Ada
COBOL	Java
	C#
	C++
	C
	Modula-2

结构化语言是现代语言。事实上，老的计算机语言的标志是：它是非结构化的。今天，对于重要的新程序，很少有程序员会考虑使用非结构化语言。

注意： 许多老的语言的新版本尝试添加结构化元素。BASIC是一个例子，尤其是Microsoft的Visual Basic。然而，这些语言的缺点永远不会完全消除，因为从一开始它们就没有设计包含结构化特征。

C语言的主要结构化组成部分是函数——C独特的子例程。在C语言中，函数是所有程序活动出现于其中的构建块。它们允许你分别定义和编码程序中不同的任务，因此程序是模块化的。在创建了一个函数后，可以依赖它在各种情况中正确地工作，而不会对程序的其他部分产

生副作用。在较大的项目中，能否创建出独立的函数是特别重要的，因为在这种项目中，一个程序员的程序代码必须不会意外地影响其他人的程序。

在C语言中，实现结构化和代码分离的另一种方法是使用代码块。代码块是一组逻辑上相互关联的程序语句，可被作为一个单元处理。在C语言中，可以通过在一组语句之间加上花括号来创建代码块。在下面这个例子中，

```
if (x < 10) {
  printf("Too low, try again.\n");
  scanf("%d", &x);
}
```

如果x小于10，在if后面、花括号之间的两条语句都被执行。这两条语句连同花括号一起表示一个代码块。它们是一个逻辑单元：其中所有语句必须一起执行。代码块允许许多算法得以清晰、优美和有效的实现，而且有助于程序员更好地抓住所要实现的算法的本质。

1.4 C语言是程序员的语言

使人吃惊的是，并不是所有的计算机编程语言都是为程序员设计的。考虑两个非程序员语言的经典例子：COBOL和BASIC。COBOL的设计对程序员来说并不太好；程序员难以改进所编写代码的可靠性，甚至不能改进代码编写的速度。COBOL设计时部分地考虑到要使非程序员更容易阅读和理解程序（然而不太可能）。BASIC的创建本质上是要允许非程序员在计算机上编程以解决相对简单的问题。

与之相反，C是由真正的程序员创建、修改和进行现场调试的。最终的结果是C给了程序员他们想要的东西：很少限制、很少抱怨、块结构、独立的函数和一小组关键字。使用C语言，既可获得汇编代码的高效率，又可具有ALGOL或Modula-2的结构。因此我们并不奇怪C和C++是一流专业程序员最常用的语言。

在使用汇编语言的地方经常使用C语言的事实是C语言在程序员中广泛流行的一个主要因素。汇编语言使用一种实际二进制代码的符号表示，计算机能够直接执行这种符号表示。汇编语言的每种操作都映射为计算机执行的单一任务。尽管汇编语言使程序员能够以最大的灵活性和最高的效率完成任务，但是当开发和调试程序时就相当难了。还有，因为汇编语言是无结构化的，最后形成的程序可能像“意大利面条”——混乱的跳转指令、调用指令和下标。这种结构化的缺乏使汇编语言程序很难阅读、改进和维护。或许最重要的是，汇编语言程序不能在具有不同中央处理器的机器间移植。

最初，C语言被用于进行系统编程。系统程序形成了计算机操作系统或它的支持实用程序的一部分。例如，下面就是通常所称的系统程序：

- 操作系统
- 翻译程序
- 编辑程序
- 编译程序（编译器）
- 文件实用工具
- 性能改进程序

- 实时执行程序
- 设备驱动程序

随着C语言的普及，许多程序员开始使用它来编程各类任务，因为它的可移植性和高效率，也因为他们喜欢它。在C语言创建之时，C是编程语言中最重大的变革。当然C++也具有这种传统。

随着C++的出现，某些人认为C作为一种不同的语言会慢慢消亡。事实并非如此。首先，并不是所有的程序都要求由C++提供的面向对象编程特征的应用。例如，像嵌入式系统这样的应用仍然是用C编程的。第二，世界上很多系统仍然运行在C代码上，那些程序将继续被改进和维护。C的最大的遗产是作为C++的基础，因此在未来的许多年内，C将继续是一种有影响力的广泛使用的语言。

1.5 C程序的结构

表1.2列出了32个关键字，它们和正式的C语法相结合，形成了C89这个C++的C子集。当然，所有这些也都是C++中的关键字。

表1.2 由C++的C子集定义的32个关键字

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

此外，许多编译器增加了几个关键字，更好地利用了操作系统环境。例如，有的编译器包含了管理8086处理器系列的内存组织的关键字，以支持交互语言程序设计并访问中断。下面是一些常用的扩展关键字：

```
asm         _cs         _ds         _es
_ss         cdecl       far         huge
interrupt   near        pascal
```

有的编译器可能还支持其他扩展，以充分利用特定的环境。

注意，所有的关键字都是小写的。C/C++是区分大小写的，因此，在C/C++程序中，大写和小写是不同的。这意味着else是一个关键字，而ELSE不是。在程序中，不能把关键字用做其他目的，即不能把它用做变量或函数名。

所有的C程序都是由一个或更多的函数组成的。必须出现的惟一一个函数是main()，它是在程序开始执行时第一个被调用的函数。在写好的C代码中，main()实际上包含程序要执行的动作的轮廓。这个轮廓由函数调用组成。尽管main()不是一个关键字，但我们可以把它看做关键字。例如，不要尝试把main用做变量名，因为这会给编译器造成混乱。

图1.1演示了C程序的一般形式，其中f1()到fN()表示用户定义的函数。

```
global declarations
return-type main (parameter list)
{
statement sequence
}
return-type f1 (parameter list)
{
statement sequence
}
return-type f2 (parameter list)
{
statement sequence
}
.
.
.
return-type fN(parameter list)
{
statement sequence
}
```

图 1.1 C程序的一般形式

1.6 库和链接

从技术上讲，纯粹用自己编写的语句来创建一个有用的C或C++程序是可能的。然而，这种情况少之又少，因为C和C++都没有提供任何关键字来执行像I/O操作、高级数学计算或字符处理这样的功能。因此，大多数程序都包含对标准库中所含的各种函数的调用。

所有的C++编译器都是和标准函数库一起提供的，标准函数库执行大多数常见的任务。标准C++规定了被所有编译器支持的最小函数集，然而，有的编译器可能包含更多的函数。例如，虽然标准库没有定义任何图形函数，但是有的编译器可能会包括一些。

C++标准库可以分为两个部分：标准函数库和类库。标准函数库是从C语言中继承来的，C++支持由C89定义的整个函数库。因此，所有的标准C函数在你编写C++程序中都可使用。

除了标准函数库外，C++也定义了它自己的类库。类库提供了程序可能使用的面向对象的例程，它也定义了标准模板库（Standard Template Library，STL），后者提供了对各种编程问题的现成的解决方案。本书后面将讨论类库和STL。在第一部分中，仅使用标准函数库，因为它是由C定义的惟一的一个。

标准函数库包含了你将使用的大多数通用的函数。调用一个库函数时，编译器“记住”它的名字。链接程序把你写的代码同标准库中找到的目标码组合起来。这个过程称为链接。某些编译器有自己的链接程序，而其他的则使用由操作系统提供的标准链接程序。

库中的函数是可重定位的。这意味着各种机器码指令的内存地址并没有绝对的定义——只有偏移量是确定的。当程序与标准库中的函数相链接时，使用这些内存偏移量来创建实际使用的地址。有关的技术手册和参考书详细地讲解了这一过程。然而，在C++中，不需要进一步解释实际的重定位过程。

编写程序时，我们需要的许多函数都可以在标准库中找到，它们都是组合程序所用的构件块。如果编写了一个需要重复使用的函数，也可以把它放到库中。

1.7 分别编译

大多数短的程序都可以包含在一个源文件中。然而，程序越长，编译的时间也就越长。因此，C/C++允许把一个程序放到多个文件中，分别进行编译。编译完所有的文件后，再将它们和库函数一并链接，以形成完整的执行代码。分别编译的好处是：如果改变一个文件的代码，不需要重新编译整个程序。在几乎所有程序上，这会节省大量的时间。C/C++编译器的用户手册都将包含编译多文件程序的指令。

1.8 理解.C和.CPP文件扩展

本书第一部分的程序当然是有效的C++程序，可以使用任何现代编译器进行编译。它们也是有效的C程序，能够使用C编译器进行编译。因此，如果要求编写C程序，本书第一部分显示的程序都可以作为例子。传统上，C程序使用.C作为文件扩展名，而C++程序使用.CPP作为文件扩展名。C++编译器使用这个文件扩展名来决定它正在编译什么类型的程序。这非常重要，因为编译器假定所有使用.C扩展名的程序是C程序，而使用.CPP的文件是C++程序。除非明确地标示出来，可以在第一部分的程序中使用这两种扩展名中的任意一种。然而，本书其余部分的程序将要求.CPP扩展名。

此外，尽管C是C++的子集，在这两种语言之间仍然存在细微的差别。有时，可能需要把C程序作为C程序（使用.C扩展名）来编译，当然，这种例子会标注出来。