全国高技术重点图书

C++ IOSTREAM
面向对象I/O程序设计

[美] Cameron Hughes • Thomas Hamilton • Tracey Hughes 著
尤晓东 潘旭燕 吴卫华 忻 超 等 译
冀惠刚 顾铁成 等 审校

WILEY
PHEI
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL:http://www.phei.co.cn

内 容 提 要

本书详细、系统地介绍了使用 C++ IOSTREAM 进行面向对象的输入输出程序设计的方法与实例。书中介绍了 C 和 C++ 输入输出流的发展历史、面向对象的流处理和 IOSTREAM 类层次体系、用户定义对象的析取和插入、操纵符、IOSTREAM 类的扩展、对多媒体设备和通信端口的访问等内容。

本书的读者对象为计算机软件开发人员。

书　　名：C++IOSTREAM 面向对象 I/O 程序设计
著　　者：〔美〕Cameron Hughes　Thomas Hamilton　Tracey Hughes
译　　者：尤晓东　潘旭燕　吴卫华　忻　超等
审 校 者：冀惠刚　顾铁成等
责任编辑：胡毓坚
印 刷 者：北京市顺义县天竺颖华印刷厂印刷
装 订 者：三河市赵华装订厂
出版发行：电子工业出版社出版、发行
北京市海淀区万寿路 173 信箱　邮编 100036　发行部电话（010）68214070
URL:http://www.phei.co.cn
经　　销：各地新华书店经销
开　　本：787 × 1092 1/16　印张：18.5　字数：450 千字
版　　次：1997 年 5 月第一版　1997 年 5 月第一次印刷
印　　数：4000 册
书　　号：ISBN 7-5053-3831-5
TP.1644
定　　价：35.00 元
著作权合同登记号　图字：01-96-1006

译 者 序

在当前这个信息社会中，计算机的应用已经非常广泛。在计算机应用中，计算机软件的质量非常重要，而编写输入输出例程和输入输出设备驱动程序是所有程序员都要进行的重要工作。输入输出处理实际上在所有的软件中都起到至关重要的作用，若没有用于驱动输入输出设备的例程，计算机将毫无作用。由于输入输出程序设计是大多数程序的基本需要，寻找编程的效率、可重用性和适用性是相当有意义的。

为了帮助我国计算机程序设计人员更好地进行输入输出程序设计工作，提高我国计算机的开发和应用水平，我们翻译出版了这本《C++ IOSTREAM 面向对象 I/O 程序设计》。参加本书翻译工作的有尤晓东、潘旭燕、吴卫华、忻超、赵海鸿、李东亚、王立梅、徐茜、黄为等同志，冀惠刚、顾铁成等对本书进行了认真的审校。

由于时间仓促，翻译过程中难免出现错误，欢迎广大读者指正。

译 者

1996 于北京

《全国高技术重点图书》
出版指导委员会名单

主　任：朱丽兰

副主任：刘　杲　　卢鸣谷

委　员：（以姓氏笔划为序）

王大中	王为珍	牛田佳	王守武	刘　仁
刘　杲	卢鸣谷	叶培大	朱丽兰	孙宝寅
师昌绪	任新民	杨牧之	杨嘉墀	陈芳允
陈能宽	罗见龙	周炳琨	欧阳莲	张钰珍
张效祥	赵忠贤	顾孝诚	谈德颜	龚　刚
梁祥丰				

总干事：罗见龙　　梁祥丰

计算机技术领域

主任委员：张效祥

委　员：（以姓氏笔划为序）

王鼎兴	刘帧权	刘锦德	李三立	何成武
徐培忠	梁祥丰	董韫美		

序　言

编写I/O例程和I/O设备编程是所有程序员都要进行的重要工作。I/O处理实际上在所有的软件中都起到至关重要的作用。若没有I/O设备和用于驱动I/O设备的例程，计算机将毫无作用。由于I/O编程是大多数大型或小型程序的基本内容，因此，努力提高编程的效率、可重用性和适用性是相当有意义的。

例程和程序都必须是高效率的。一些系统性能都是单独由I/O性能所测定的。目前在网络、GUI或有着大量I/O的多媒体应用程序中，效率更是一种最根本的需要。I/O例程必须具有可重用性。为降低软件开发成本，需要可重用的构件。当今新技术推广的步伐已不允许程序员重复设计每个部分。通常，一旦I/O功能被测试和调试后，将放入库中，并且，如果设计适当的话，将被反复使用。最后，I/O编程必须适合于所采用的设备及数据。例如，使用缓冲技术对于一些设备能够提高效率，但对另一些设备可能造成死机等问题。在VGA图形适配器上采用的技术可能就不适用于IBM 8514。

使用C++ IOSTREAM进行面向对象的编程(正如本书书名的含义所示)，主要集中于讨论以下两点：

- 面向对象的输入/输出编程技术
- C++ IOSTREAM

当C++ IOSTREAM和面向对象的编程技术结合使用时，就可以很容易地实现高效率、可重用性和适用性。本书的读者应对C++语言具有基本理解，并初步具有C++编程技术。未涉足过C++ IOSTREAM倒是无关紧要的。

本书将I/O编程引入一个面向对象的世界。本书致力于从系统程序员的角度和应用程序程序员的角度来讨论问题。从通信端口的编程到设备环境及表示空间，对这些I/O设备和技术都将进行深入的探讨。

我们引进了面向对象的流(object-oriented stream)这个概念，它不仅仅只带来了编程时的简便。IOSTREAM类函数族的基本操作都将在本书中加以介绍。书中给出的所有例子都被测试通过，这些IOSTREAM类的例子都是在Windows和OS/2环境下运行的。对于怎样将IOSTREAM类连接到新的多媒体操作系统下，本书也给予了简要提示。在第一章到第六章中所给出的大多数例子都可以经过微小的修改之后，在任何环境之下运行。我们在编写本书时，尽可能地和最新的ANSI C++方案接近。对于ANSI方案目前并未明确规定的地方，我们采用的都是IOSTREAM类最流行的实现。

C++是C语言的进化，它在C丰富表达力的基础上增加了面向对象的功能。IOSTREAM类是标准流概念的进化，它向标准I/O库中增加了对象功能。IOSTREAM可以扩展并专门化，以便封装为新的多媒体流。目前，有关C++编程的这一重要方面的资料极其缺乏，希望本书能填补这一缺憾。

我们并不讳言，本书中所提供的例子或许存在错误，也不保证它们能够适合各种特殊应

用程序的需要。读者不应依赖于它们来解决特定的问题，以免其不正确的解决方案导致对人身的伤害或财产的损失。作者和出版商都不承担由于使用书中程序所造成的直接或间接损失。

致　谢

我们要特别感谢 Carol、Belinda，我们的著作代理人 Bernard David、Tim Ryan 和 Bob Aronds。

目　录

第一章　简介：流的历史和概念

当我们的智慧之眼能够洞察距离越来越小、速度越来越快的事物时，我们发现这自然界和我们观察周围明显可见的事物所看到的并不一样，以至于没有哪一种根据我们宏观经历所形成的模型能始终为“真”了。

ERWIN SHRODINGER——因果学和波力学

计算机工业中许多优秀的思想都是逐渐进化的结果，是许多人及组织的努力工作及智慧的结果。流(stream)的概念也不例外。输入/输出流是一些例程、数据结构和技术的集合，它使程序员以一种高效通用的方式来处理输入输出。

流的概念由来已久。其起源可以追溯到第二代计算机语言(如汇编语言)到第三代语言(诸如 FORTRAN、COBOL、BASIC、C 和 Pascal)的进化过程中。第三代语言曾经试图寻找一组通用的命令或函数，以处理连接到系统中的所有的 I/O 设备(参见图 1.1)。例如在 Pascal 语言中，write(Filename，Output)函数用于发送输出到控制台、打印机或存储文件中，这就和汇编语言解决 I/O 问题所采用的方式有所区别。第二代汇编语言的程序员在发送输出到控制台、打印机或存储设备时，需要端口地址、内存配置、中断位置和硬件设备名，并掌握字节交换、数据转换以及快速内存传输等技术。

流的概念在操作系统从第二代到第四代进化的过程中前进了一大步。从设计包括 LIOCS(逻辑 I/O 控制系统)的 IBM 的 DOS/VSE 操作系统时所取得的收获，到最终被包含在 UNIX 操作系统中的管道、过滤器和文本转换概念，数据流的概念和 I/O 流诞生了。

在业界中，C 语言、标准 C 库的 I/O 例程和 UNIX 操作系统构成了输入输出流概念的大多数实际可见的应用程序。UNIX 于 1970 年由 Bell 实验室的 Ken Thompson 和 Dennis Ritchie 开发完成。C 和 C++也是由 Bell 实验室推出的。这些语言都最终设计成能在 UNIX 环境下运行。流和 UNIX 操作系统的思想是紧密结合的。所有的 I/O 应导向文件或来自文件，甚至设备都是一种特殊类型的文件，这些思想是 UNIX 环境的主题内容。

流的概念和设备无关性(device independence)这一概念紧密相关。当程序或例程具备设备无关性时，如果所访问的设备存在变化，它们只要作少量修改或根本不需要修改就能够运行。例如，在设计例程时是将输出发向控制台，后来决定将输出发送到打印机，采用设备无关性思想的例程中的程序逻辑将不需要任何修改。

这样在编程时，程序员不必关心所访问的特定设备的细节。设备无关性被视为程序可移植性和可重用性的终极目标。表 1.1 中将与设备无关和设备相关加以对比。我们将在后面详细讨论设备无关的概念。这会儿可以这么说，在讨论 I/O 时，程序的设备无关性越强，就越接近流的概念。

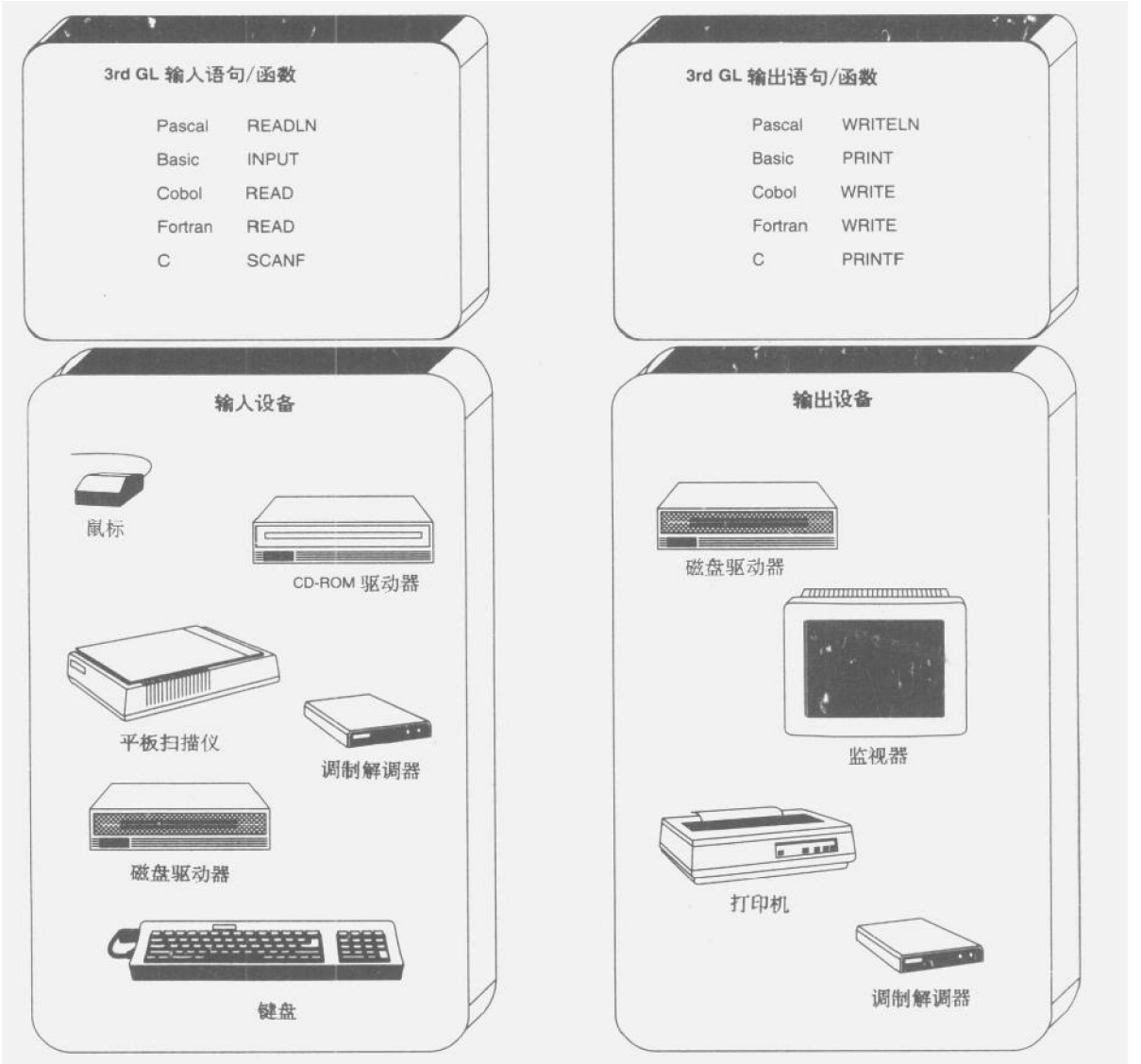

图 1.1　第三代输入输出语句、函数及它们所寻址的设备

表 1.1　设备无关和设备相关概念比较

设备相关	设备无关
特定硬件	可在一类硬件间移植
指定地址	使用文件句柄和重定向
需要指定分辨率	能够缩放分辨率
指定有关硬件构件的基本知识	访问硬件高级系统 API
程序代码和指定设备名相关	程序代码和 IOSTREAM 相关

使用流的概念，程序员将流入流出系统的数据看作是一个字节流。理想情况下，不能只

将流看作流入流出特定设备，而应看作能够接收和发送字节的通用设备(参见图1.2)。尽管通用I/O数据流是一种相当有效的概念，但对于特定的任务仍不一定最有效。无论如何，现代的任何I/O库若不支持流的概念，就不能称为是完整的。

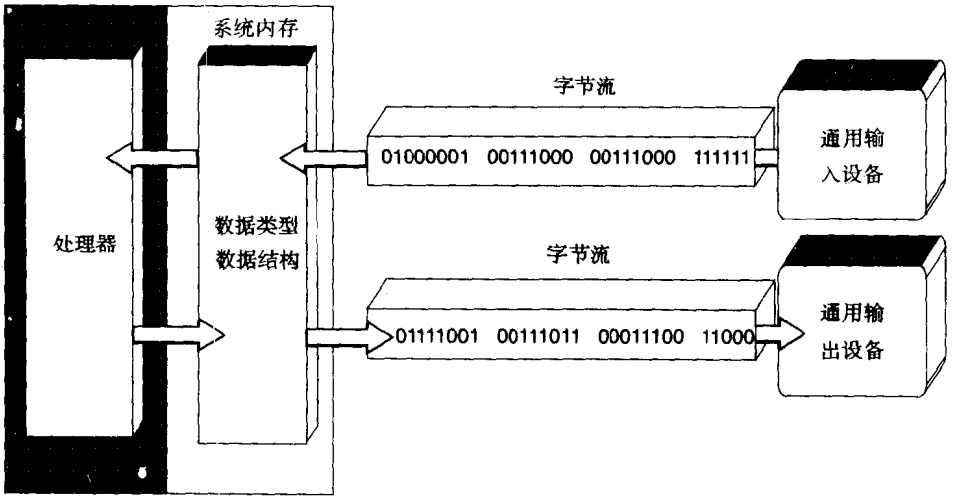

图1.2　与计算机相连的输入输出设备的通用字节流

1.1　标准C IOSTREAM

C++程序员应熟悉C语言的stdio库，原因至少有四点：

(1)许多C++程序和C程序是同时存在的。C程序需要调用stdio。

(2)许多包含经过测试和调试的代码的C库可以连接到C++应用程序中。

(3)有的C I/O例程必须转化成为C++ IOSTREAM。

(4)stdio库是C++ I/O功能的一个子集。

C++ I/O流可以和C stdio函数结合起来使用。C标准库函数中主要的I/O函数是在头文件stdio.h中说明的。这些函数的完整形式可以在C标准库中找到。stdio函数几乎构成标准C库函数的三分之一。下面我们将对C stdio库函数的基本组成进行简要的介绍。

1.1.1　stdio数据结构和头文件

FILE结构(或对象)应当被看作黑匣子。它的内部结构通常包含文件描述和缓冲信息。EOF是由几个函数返回的一个负整数常量表达式，以表明出现了错误或文件结束。

全程变量errno不包含在stdio库中，但也可被一些stdio库函数设置，以表明出现了错误。

1.1.2　预分配流

有三种预分配流：stdio、stdout和stderr，一些实现还包括stdprn和stdaux。在程序的主函数被调用前，这些流就被打开。流stdio、stdout和stderr都是指向FILE数据结构的指针。它们

分别对应于标准输入、标准输出和标准错误。键盘作为输入设备，控制台用于标准输出和错误。许多函数隐式使用这些流，但它们每个都具有完整的显式版本。在 C stdio 库中，不需要打开或关闭这些流。

1.1.3 文件访问函数

文件访问函数允许打开、关闭、重新打开和刷新与某个流有关的文件。

原型：int fclose(FILE * stream);

说明：fclose()函数关闭与 stream 有关的文件。任何缓冲的输入都被废弃，任何缓冲的输出都在文件关闭前被写出。fclose()函数在调用成功时返回 0；失败时返回 EOF。

原型：int fflush(FILE * stream);

说明：fflush()函数把所有其缓冲输出流中待完成的输出写到它的目标设备中。调用 fflush(void(0))可刷新所有打开的输出流。

原型：FILE * fopen(const char * filename,const char * mode);

说明：fopen()函数可打开由 filename 字符串指定了名字的文件。mode 字符串指定了文件流的访问模式。fopen()函数可返回指向控制此流的 FILE 对象的指针，在调用失败时返回空指针。在一些实现形式中，在标准 ANSI 模式字符之后可以加上另外的访问模式信息。另一些实现形式中，会提供另外一个函数，如_fsopen，用于允许以共享模式打开文件。表 1.2 是 ASNI 标准文件打开模式的列表。

表 1.2 ANSI 标准文件打开模式及其意义

模式	意义
r	打开一个文本文件用于只读
w	缩短文件长度为 0 或建立一个可写的文本文件
a	添加：打开或建立一个文本文件并从 EOF 处开始写
rb	打开一个二进制文件用于只读
wb	缩短文件长度为 0 或建立一个可写的二进制文件
ab	添加：打开或建立一个二进制文件并从 EOF 处开始写
r+	打开一个文本文件用于修改(读和写)
w+	缩短文件长度为 0 或建立一个可修改的文本文件
a+	添加：打开或建立一个文本文件并从 EOF 处开始修改
r+b 或 rb+	打开一个二进制文件用于修改(读和写)
w+b 或 rw+	缩短文件长度为 0 或建立一个可修改的二进制文件
a+b 或 ab+	添加：打开或建立一个二进制文件并从 EOF 处开始修改

原型：FILE * freopen(const char * filename,const char * mode,FILE * stream);

说明：函数 freopen()打开由 filename 指定了文件名的文件。mode 字符串指定文件流的访问模式。此函数然后关闭提供的流，并使一个文件与之相关。freopen()函数调用成功时返回提供的流，否则返回空指针。

1.1.4 缓冲函数

缓冲模式有三种：_IOFBUF、_IOLBUF 和_IONBUF，分别用于全缓冲、行缓冲和不缓冲。全缓冲是指当流中的数据存满之后，把它们作为一个数据块传送到操作系统或从操作系统

传入。行缓冲是指把存入的数据作为一个数据块保存，直至一个换行符被输入到流中。不缓冲是指源和目的之间只要有可能就传送数据。缺省的流缓冲由实现定义。下面两个函数，setbuf()和 setvbuf()可用于覆盖这些缺省缓冲模式。

原型：int setvbuf(FILE * stream,char * s,int mode,size_t size);

说明：setvbuf()函数用于重新定义或指定程序员定义的流所用的缓冲区。如果 s 是一个空指针，setvbuf()将分配自己的缓冲区；否则将使用提供的缓冲区，并认为其是合法的。如果提供了一个缓冲区，其大小被传递给第四个参数 size。缓冲模式由第三个参数决定，可以是_IOFBUF、_IOLBUF 或_IONBUF。此函数必须在一个流已和一个文件连接之后，而且未对流进行其他操作之前使用。如果函数 setvbuf()调用成功，将返回 0 值。如果 mode 为非法值或函数调用失败，将返回非 0 值。

原型：void setbuf(FILE * stream,char * s);

说明：setbuf()函数是 setvbuf()函数的速写，包括两种操作方式。当指定的缓冲区为空指针时，_IONBUF 作为缓冲模式。否则缓冲模式为_IOFBUF，缓冲区的大小由宏 BUFSIZE 定义（至少为 256）。

1.1.5　直接输入/输出函数

直接输入和输出函数类似于 UNIX 系统调用。实际上，当用文件描述符替换 FILE * 时，它们的接口是完全相同的。

原型：size_t fread(void * ptr,size_t size,size_t count, FILE * stream);

说明：fread()函数从 * stream 指定的流中读取 count 个指定长度的数据字段，并把它们放到 ptr 指向的数组中。fread()函数将返回成功读取的字段的个数；如果发现了 EOF 或出现读错误，返回值可能小于 count；如果 size 或 count 为 0，fread()将返回 0 及 ptrim 值。流的状态将保持不变。

原型：size_t fwrite(const void * ptr,size_t size,size_t count,FILE * stream);

说明：fwrite()函数把提供的数组 ptr 中的 count 个 size 长度的数据字段写到指定的流中。fwrite()函数返回写到流中字段的个数。如果出现写错误，返回值将小于 count。

1.1.6　获取及定位文件指针

指针表示的是从文件的开始处到文件中要读或要写下一个字符的位置的字节数。除 rewind()函数之外，下面的函数调用成功时，将重新设置流的出错状态。

原型：int fgetpos(FILE * stream,fpos_t * pos);

说明：fgetpos()函数将把指定了流的文件指示器的当前值存储到由 pos 指向的对象中。这一数值将被 fsetpos()使用。fgetpos()函数在调用成功时返回 0，否则返回非 0 值。出现错误时，fgetpos()函数在全程变量 errno 中存储一个由具体实现定义的值。

原型：int fseek(FILE * stream,long int offset,int whence);

说明：fseek()函数根据最后一个参数 whence 给定的字节偏移量，重新定义流的文件定位指示器。二进制流可以使用任何偏移量值，但文本流则应使用前一次调用 ftell()函数得到的值。fseek()函数在调用成功时返回 0，否则返回非 0 值。

原型：int fsetpos(FILE * stream,const fpos_t * pos);

说明：fsetpos()函数根据对同一流调用fgetpos()函数得到的数值来重新定位文件指示器。数值存储在由pos指向的对象中。调用fsetpos()函数成功时返回0，否则返回非0值。出现错误时，fsetpos()函数在全程变量errno中存储一个由实现定义的正值。

原型：long int ftell(FILE * stream);

说明：ftell()函数返回流的文件位置指示器的当前值。如果是二进制流，返回值是从文件开始处计算的字节数；如果是文本流，返回值只能被fseek()使用。ftell()函数在调用失败时，将返回-1L，并在errno中存储一个预定义的正值。

原型：void rewind(FILE * stream);

说明：rewind()函数设置流的文件指示器指针到文件的开始处。rewind()函数没有返回值，也不复位流的错误状态。

1.1.7 字符输入和输出

有几个函数可以完成文本和字符的输入和输出。它们都是逐个字符地读或写文本流。它们的实现形式已被优化以便尽可能快地传输数据，它们可以看作fgetc()和fputc()函数的增强版本。

原型：int fgetc(FILE * stream);

说明：fgetc()函数从由stream指定的输入流中获取下一个字符。函数得到的是转化为int的无符号char，如果stream是在文件的结束处或出现读错误时，fgetc()函数将返回EOF。

原型：char * fgets(char * s,int n,FILE * stream);

说明：fgets()函数从由stream指定的流中获取字符串，并存储到由s提供的缓冲区中，字符的个数最多为n-1。读入并保留换行字符。字符串都附加有一个空字符。函数调用成功时，返回提供的缓冲区的指针，否则返回一个空指针。

原型：int fputc(int c,FILE * stream);

说明：fputc()函数把单个字符(需要从整数转换为无符号的char)写入到由stream指定的输出流中，位置由文件指示器指定。如果文件无法定位文件指示器或流的访问模式为添加时，单个字符将添加到流中。fputc()函数在调用成功时返回写入的字符，否则返回EOF。

原型：int fputs(const char * s,FILE * stream);

说明：fputs()函数向stream指定的输出中写入字符串s。终止空字符不被发送。fputs()函数在调用成功时返回非0值，否则返回EOF。

原型：int getc(FILE * stream);

说明：宏getc()是fgetc()的一种实现形式，务必注意指定的参数不能具有任何副作用，因为流参数可能会被多次求值。例如：

```
int char=0;
/* Get and count all pending characters for standard input */
while (getc (++char ? stdin:stdin) ! = EOF);
```

如果有10个剩余的字符，char的数值将不一定为10。

getc()函数在调用成功时返回stream的下一个字符。当流到达文件的末尾或发生错误时，返回EOF。

原型：int getchar(void);

说明：getchar()函数和标准输入中的fgetc()是等价的。此函数是预分配的标准输入流中的隐式输入调用之一。此函数返回stdin的下一个字符，若流到达文件的末尾或发生错误时，返回EOF。

原型：char * gets(char * s);

说明：gets()函数从stdin中读取字符，放入由s指向的字节数组中，直至读到换行或EOF。换行字符被废弃，并在数组中最后一个字符之后放置空字符。gets()函数调用成功时返回s。如果出现错误或没有读到字符，返回空指针。

原型：int putc(int c,FILE * stream);

说明：putc()函数被实现为一个宏。它把c发送到FILE * stream。它与fputc()本质上是等价的。两个参数都应没有副作用，因为stream可能会被多次求值。putc()函数在调用成功时返回写入的字符，否则返回EOF。

原型：int puts(const char * s);

说明：puts()函数把由s指定了的字符串写到stdout，并把\N加到输出中。终结空字符不写到stdout。puts()函数在调用成功时返回非负值，出现写错误时返回EOF。

原型：int ungetc(int c,FILE * stream);

说明：ungetc()函数用于将由c指定的字符退回到由stream指定的输入流中。它保证只有一个字符被成功退回(因为它可能是未缓冲的或缓冲区已满)。ungetc()函数在调用成功时返回转换后被退回的字符，否则返回EOF。

1.1.8　格式化函数

stdio的格式化输入输出函数可作为在系统间传输数据的一种手段。数据的二进制表示形式和平台是相关的，而文本形式是通用的。把二进制数据转换为文本格式的标准方法，消除了平台所带来的特殊性。

stdlib库中有一些转换例程，用于整数和浮点数值与字符串之间的转换。但是，使用诸如atof()的函数来构造一个输出字符串并不像使用一行代码来指定一行输出一样直观。

1.1.9　可变参量

一些语言具有允许可变数目参量的特定函数。C语言提供了一种手段，利用它可以将任何函数定义成都具有这种能力，这种手段要通过stdarg库完成。C语言定义ellipse声明，用于通知编译器某函数将接受可变数目的参量。ellipse和va_list对象(指向char的指针)可以看作等价的，只是后者需要程序员恰当地建立可变参数块。

1.1.10　格式指定符

格式字符串都包含普通字符和格式指定符。指定符的格式是：

% [flags][width][.prec][h|l]type_specifier

格式字符串是一种缩微语言，其中标志、宽度和精度字符用于调整输出。尺寸指定符([h|l])和类型指定符用于描述参数是如何转换的。只有类型指定符是必须有的。

特别要注意的是，输入与输出的格式指定符是不同的，虽然它们看起来非常相似。输出数据是按值传递，而输入参数是通过地址接收。程序员应保证每一个格式指定符都有目标参