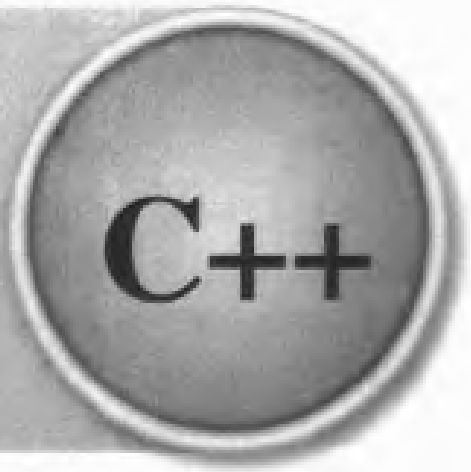
The Complete Reference

参考大全



第三部分

标准函数库

C++定义了两种类型的库。第一种库是标准函数库，这个库由具有多种用途的独立函数组成，这些函数不属于任何类。函数库是从C继承而来的。第二种库是面向对象的类库。本书第三部分提供了标准函数库的参考资料，第四部分描述了类库。

标准函数库分为以下几类：

- 输入/输出函数
- 字符串和字符处理函数
- 数学函数
- 时间、日期和地点函数
- 动态分配函数
- 杂项函数
- 宽字符函数

最后一类函数于1995年添加到标准C语言中，后来又被合并到C++中。这类函数提供了与几个库函数具有相同功效的宽字符（wchar_t）。坦率地说，宽字符库的使用非常有限，但是C++提供了操作宽字符的较好环境，本书将在第31章给出简要描述。

C99往C函数库中添加了一些新元素，其中的几个，例如，支持复杂运算、通用类型的数学函数的宏，与C++中已有的功能重复。还有一些新功能可能在将来被合并到C++中。无论如何，C99添加的库元素与C++是不兼容的，所以本书不讨论C99的标准C库。

最后要说明的是：所有编译器提供的函数都比标准C/C++所定义的函数要多，通常情况下，这些附加函数用来为操作系统接口和其他与环境有关的操作提供服务。具体情况应查阅相应的编译器文档。

第25章　基于C的输入/输出函数

本章介绍基于C的输入/输出（I/O）函数，这些函数也被标准C++所支持。虽然人们在编写新代码时通常愿意采用C++的面向对象的I/O系统，但是当你认为在一个C++程序中使用C的I/O函数更合适时，也没有任何理由拒绝使用。本章介绍的函数最早是由ANSI C标准指定的，它们通常被称为ANSI C I/O系统。

与基于C的I/O函数相关联的头文件称为<cstdio>（一个C程序必须使用头文件stdio.h）。这个头文件定义了几种被文件系统使用的宏和类型，其中最重要的类型是FILE，该类型用于声明一个文件指针。除此之外，还有另外两种类型，它们是size_t和fpos_t。size_t类型（通常是一些无符号整型）定义一个对象，该对象能够存放操作系统所允许的最大文件；fpos_t类型也定义一个对象，该对象可以存放惟一指定文件中每一个位置的所有信息。头文件中定义的最常用的宏是EOF，它是一个指示文件结束的值。

许多I/O函数可以在出现错误时设置内置的全局变量errno。当出现错误时，程序可以检查这个变量以获得关于该错误的更多信息。errno的值是与实现相关的。

要想了解基于C的I/O系统的概况，可参阅第一部分中的第8章和第9章。

注意：本章讲述基于字符的I/O函数。这些函数最早是为标准C和C++定义的，到目前为止，它们是使用最广泛的函数。在1995年，几种宽字符（wchar_t）函数被添加进来，本书将在第31章对其进行简要介绍。

25.1　clearerr函数

```
#include <cstdio>
void clearerr(FILE *stream);
```

clearerr( )函数重置（即设置为0）与stream所指的流相关的错误标记。此外，文件结束指示符也被重置。

每个流的错误标记最初是通过对fopen( )的成功调用而设置的。文件错误可能由多种原因产生，其中有许多是与系统相关的。错误的性质可以通过调用perror( )来决定，该函数可以显示发生了什么错误（参见perror( )）。

与clearerr( )相关的函数有feof( )，ferror( )和perror( )。

25.2　fclose函数

```
#include <cstdio>
int fclose(FILE *stream);
```

fclose( )函数关闭与stream相关的文件并刷新相应的缓冲区。执行fclose( )之后，stream不再与文件相连接，所有被自动分配的缓冲区都将释放。

如果fclose( )执行成功，则返回0；否则，返回EOF。如果试图关闭一个已经被关闭的文件，那么将产生一个错误。另外，在关闭文件之前删除此文件也将产生一个错误，因为这将导致缺乏足够的自由磁盘空间。

与fclose( )相关的函数有fopen( )，freopen( )和fflush( )。

25.3 feof函数

```
#include <cstdio>
int feof(FILE *stream);
```

feof( )函数用来检查文件位置指示符以确定是否到达与stream相连的文件的末尾。如果文件位置指示符位于文件尾，该函数返回一个非0值；否则，返回0。

一旦到达文件尾，后来进行的读取操作将返回EOF，直到调用rewind( )或用fseek( )移动文件位置指示符为止。

因为文件结束标记也是一个有效的二进制整型数，所以feof( )函数在处理二进制文件时格外有用。例如，若要确定是否到达了一个二进制文件的末尾，必须明确地调用feof( )，而不是简单地测试getc( )的返回值。

与feof( )相关的函数有clearerr( )，ferror( )，perror( )，putc( )和getc( )。

25.4 ferror函数

```
#include <cstdio>
int ferror(FILE *stream);
```

ferror( )函数检查给定的stream的文件错误。如果没有错误，返回0；否则，返回一个非0值。

为了准确地确定错误性质，可使用perror( )函数。

与perror( )相关的函数有clearerr( )，feof( )和perror( )。

25.5 fflush函数

```
#include <cstdio>
int fflush(FILE *stream);
```

如果stream与一个以写入方式打开的文件相连，调用fflush( )将使输出缓冲区的内容被真正写入到该文件，而且文件仍保持打开状态。

如果函数的返回值为0，则说明写入成功；如果返回值为EOF，则表示发生了写错误。

如果程序正常结束或者缓冲区被写满，则所有缓冲区会被自动刷新。

与fflush( )相关的函数有fclose( )，fopen( )，fread( )，fwrite( )，getc( )和putc( )。

25.6 fgetc函数

```
#include <cstdio>
int fgetc(FILE *stream);
```

fgetc( )函数从输入stream中返回当前位置的下一个字符并且给文件的位置指示符加1。该字符被读作一个被转换成整数的unsigned char类型。

如果到达了文件尾，fgetc( )返回EOF。然而，因为EOF是一个有效的整型值，所以当处理二进制文件时，必须利用feof( )检查文件结束与否。如果fgetc( )遇到了一个错误，也将返回EOF。若要处理二进制文件，必须利用ferror( )检查文件错误。

与fgetc( )相关的函数有fputc( )，getc( )，putc( )和fopen( )。

25.7 fgetpos函数

```
#include <cstdic>
int fgetpos(FILE *stream, fpos_t *position);
```

fgetpos( )函数把文件位置指示符指定的当前值存储在position所指的对象中。position所指的对象必须是fpos_t类型。该对象中存储的值只有在以后调用fsetpos( )时才有用。

如果发生错误，fgetpos( )返回一个非0值，否则将返回0。

与fgetpos( )相关的函数有fsetpos( )，fseek( )和ftell( )。

25.8 fgets函数

```
#include <cstdio>
char *fgets(char *str, int num, FILE *stream);
```

fgets( )函数从stream中读取num-1个字符并将这些字符放入str所指的字符数组。无论遇到换行符或EOF，还是遇到指定的限定符，函数将停止读取字符。该函数在读完字符之后，立即在读出的最后一个字符后放一个Null。另外，该函数将保留一个换行符并将其作为str所指的数组的一部分。

如果读取成功，fgets( )返回str；如果失败，则返回一个空指针。如果出现读取错误，str所指的数组内容将是不确定的。因为无论是出现一个错误，还是到达文件尾，函数都将返回一个空指针，所以应该使用feof( )或ferror( )确定究竟发生了哪一种情况。

与fgets( )相关的函数有fputs( )，fgetc( )，gets( )和puts( )。

25.9 fopen函数

```
#include <cstdio>
FILE *fopen(const char *fname, const char *mode);
```

fopen( )函数打开一个文件（该文件名由fname函数指定）并返回与它相连的流。可以用于该文件的操作类型由mode值定义。表25.1列出了mode的合法值。文件名必须是由操作系统定义的有效文件名组成的字符串，而且还可以包含环境所支持的路径说明。

如果fopen( )成功打开了指定的文件，则返回一个FILE指针；如果没能打开文件，则返回一个空指针。

就像表中列出的那样，一个文件既能够以文本模式打开，也能够以二进制模式打开。如果是按文本模式打开，有可能发生一些字符转换。例如，换行可以被转换为回车/换行序列。如果是按二进制模式打开，则不会发生这种转换。下面的代码片断说明了打开一个文件的正确

方法：

```
FILE *fp;

if ((fp = fopen("test", "w"))==NULL) {
  printf("Cannot open file.\n");
  exit(1);
}
```

该方法可以在试图往文件中写入内容之前对文件进行打开操作时检查任何错误，例如写保护错误和磁盘已满错误。

如果使用fopen( )打开一个输出文件，那么以这个文件名命名的先前的文件内容将被清除并形成一个新文件。如果不存在这个名字的文件，函数将创建一个文件。

表25.1 fopen( )函数中模式（mode）参数的合法值

模式	含义
"r"	打开要读取的文本文件
"w"	创建一个要写入的文本文件
"a"	把内容添加到文本文件
"rb"	打开要读取的二进制文件
"wb"	创建要写入的二进制文件
"ab"	把内容添加到二进制文件
"r+"	打开要读取/写入的文本文件
"w+"	创建要读取/写入的文本文件
"a+"	打开要读取/写入的文本文件
"rb+" 或 "r+b"	打开要读取/写入的二进制文件
"wb+" 或 "w+b"	创建要读取/写入的二进制文件
"ab+" 或 "a+b"	打开要读取/写入的二进制文件

如果是为了进行读取操作打开一个文件，则要求这个文件必须存在。如果文件不存在，函数将返回一个错误。如果想往文件尾添加信息，必须使用模式“a”。如果文件不存在，就创建它。

当访问一个以读/写方式打开的文件时，不能在输出操作之后跟着执行输入操作，除非在两个操作之间执行fflush( )，fseek( )，fsetpos( )或rewind( )。另外，如果在两个操作之间没有执行这些函数，也不能在输入操作之后跟着执行输出操作，除非在输入过程中到达了文件尾。也就是说，在文件末尾处可以直接在输入操作后执行输出操作。

与fopen( )函数相关的函数有fclose( )，fread( )，fwrite( )，putc( )和getc( )。

25.10 fprintf函数

```
#include <cstdio>
int fprintf(FILE *stream, const char *format, ...);
```

fprintf( )函数把由format字符串指定的参数列表组成的参数值输出到stream所指的流，其返回值为实际打印的字符数。如果发生错误，则返回一个负数。

这里可以有0个参数，也可以有几个参数，其最大值与使用的系统有关。

控制字符串格式的操作和指令与printf( )的相同，详细内容参见printf( )。

与 fprintf( )相关的函数有 printf( )和 fscanf( )。

25.11　fputc 函数

```
#include <cstdio>
int fputc(int ch, FILE *stream);
```

fputc( )函数按当前文件位置将字符 ch 写入指定流并给文件位置指示符加 1。尽管由于历史原因将 ch 声明为 int，但它被 fputc( )转换为 unsigned char。因为所有字符参数在调用时才被改为整型，所以通常看到的是将字符值用做参数。如果使用整型参数，高位字节将被丢弃。

fputc( )的返回值是写入的字符值。如果出现了错误，则返回 EOF。对为二进制操作打开的文件来说，EOF 可以是有效字符，而且需要用函数 ferror( )确定是否真的发生了错误。

与 fputc( )相关的函数有 fgetc( )，fopen( )，fprintf( )，fread( )和 fwrite( )。

25.12　fputs 函数

```
#include <cstdio>
int fputs(const char *str, FILE *stream);
```

fputs( )函数把 str 所指的字符串内容写到指定的流，而且不写入 null 结束符。

如果运行成功，fputs( )函数返回一个非负数；如果失败，则返回 EOF。如果以文本模式打开流，有可能进行一些字符转换。这意味着字符串和文件之间的映射有可能不是一对一的。然而，如果流是以二进制模式打开的，那么将不会发生字符转换，而且字符串和文件之间的映射将是一对一的。

与 fputs( )相关的函数有 fgets( )，gets( )，puts( )，fprintf( )和 fscanf( )。

25.13　fread 函数

```
#include <cstdio>
size_t fread(void *buf, size_t size, size_t count,
             FILE *stream);
```

fread( )函数从 stream 所指的流中读取 count 个对象（每个对象的大小为 size）并将这些对象放入 buf 所指的数组中，然后将文件位置指示符向前移动，移动个数为被读取的字符个数。

fread( )函数返回实际读取的项数。如果读取的项数少于调用所要求的个数，则要么出现一个错误，要么到达了文件末尾。要想确定究竟出现了哪一种情况，必须使用 feof( )或 ferror( )。如果流是针对文本操作被打开的，或许会发生一些字符转换，例如：将回车 / 换行序列转换为新行。

与 fread( )相关的函数有 fwrite( )，fopen( )，fscanf( )，fgetc( )和 getc( )。

25.14　freopen 函数

```
#include <cstdio>
FILE *freopen(const char *fname, const char *mode,
              FILE *stream);
```

freopen( )函数把一个已有的流与一个不同的文件相联。新的文件名由 fname 指定，访问模式由 mode 指定，重新分配的流由 stream 指定。mode 采用与 fopen( )相同的格式，详细内容参见 fopen( )。

当调用时，freopen( )首先关闭与 stream 相关联的当前文件。然而，如果试图关闭文件的操作未能成功，freopen( )函数仍继续打开其他文件。

如果运行成功，freopen( )函数返回一个指向 stream 的指针；否则，返回一个空指针。

freopen( )函数的主要用途是将系统定义的流（stdin，stdout 和 stderr）重新定向到其他的文件。

与 freopen( )相关的函数有 fopen( )和 fclose( )。

25.15 fscanf 函数

```
#include <cstdio>
int fscanf(FILE *stream, const char *format, ...);
```

fscanf( )函数很像 scanf( )，但它是从 stream 指定的流而不是从 stdin 读取信息。详细内容参见 scanf( )。

fscanf( )函数返回实际被赋值的参数个数，该数不包括跳过的字段。返回值为 EOF 意味着在第一次赋值前出现了故障。

与 fscanf( )相关的函数有 scanf( )和 fprintf( )。

25.16 fseek 函数

```
#include <cstdio>
int fseek(FILE *stream, long offset, int origin);
```

fseek( )函数根据 offset 和 origin 的值设置与流相关联的文件位置指示符。它的用途是支持可以随机访问的 I/O 操作。offset 是从 origin 开始查找的字节数，origin 的值必须是下面所列的宏（在<cstdio>中定义）中的一种。

名字	含义
SEEK_SET	从文件开始处查找
SEEK_CUR	从当前位置查找
SEEK_END	从文件尾查找

如果返回值为 0，则说明 fseek( )运行成功；若返回值不为 0，则说明运行失败。

可以使用 fseek( )把位置指示符移到文件中的任何地方，甚至可以超过文件尾。然而，如果试图把位置指示符移到文件开始之前，则会引发一个错误。

fseek( )函数还将清除与指定流相关联的文件结束标记。此外，它还使得以前作用于同一个流的 ungetc( )失效（参见 ungetc( )）。

与 fseek( )相关的函数有 ftell( )，rewind( )，fopen( )，fgetpos( )和 fsetpos( )。

25.17　fsetpos 函数

```
#include <cstdio>
int fsetpos(FILE *stream, const fpos_t *position);
```

fsetpos( )函数把文件位置指示符移到 position 所指的对象指定的位置。这个值以前已通过调用 fgetpos( )获得。在执行 fsetpos( )之后，文件结束指示符被重新设置。另外，任何先前对 ungetc( )的调用将失效。

如果 fsetpos( )运行失败，将返回一个非 0 值；如果运行成功，则返回 0。

与 fsetpos( )相关的函数有 fgetpos( )，fseek( )和 ftell( )。

25.18　ftell 函数

```
#include <cstdio>
long ftell(FILE *stream);
```

ftell( )函数返回 stream 指定的文件位置指示符的当前值。如果是二进制流，返回值是从文件开始计算的字节数；如果是文本流，该返回值没有意义，除非是作为 fseek( )的参数，这是因为有可能发生影响文件表观大小的字符转换，例如：回车 / 换行被替换为新行。

当出现错误时，ftell( )函数返回 -1。

与 ftell( )相关的函数有 fseek( )和 fgetpos( )。

25.19　fwrite 函数

```
#include <cstdio>
size_t fwrite(const void *buf, size_t size,
              size_t count, FILE *stream);
```

fwrite( )函数把 count 个对象从 buf 指定的字符数组写到 stream 指定的流，其中某个对象的大小为 size 个字节。然后函数将文件位置指示符前移，移动量为写入的字符数。

如果调用成功，fwrite( )函数返回实际写入的对象个数，该返回值等于要求的个数；如果写入的对象数少于要求的个数，将引发一个错误。

与 fwrite( )相关的函数有 fread( )，fscanf( )，getc( )和 fgetc( )。

25.20　getc 函数

```
#include <cstdio>
int getc(FILE *stream);
```

getc( )函数返回输入流的下一个字符并给文件位置指示符加 1。该字符被读取为 unsigned char 类型，该类型被转换为一个整型。

如果到达文件尾，getc( )返回 EOF。然而，由于 EOF 是一个有效的整型值，因此当处理二进制文件时，必须使用 feof( )检查文件结束字符。如果 getc( )遇到一个错误，也将返回 EOF。如果是处理二进制文件，必须使用 ferror( )检查文件错误。

函数 getc( )和 fgetc( )一样，而且在大多数实现中只是将 getc( )定义为如下所示的宏。

```
#define getc(fp) fgetc(fp)
```

上面的语句将用fgetc( )函数代替getc( )宏。

与getc( )相关的函数有fputc( )，fgetc( )，putc( )和fopen( )。

25.21 getchar函数

```
#include <cstdio>
int getchar(void);
```

getchar( )函数返回stdin中的下一个字符。该字符被读取为unsigned char类型，该类型被转换为一个整型。

如果到达文件尾，getchar( )返回EOF。如果getchar( )遇到一个错误，也会返回EOF。getchar( )函数经常作为一个宏实现。

与getchar( )相关的函数有fputc( )，fgetc( )，putc( )和fopen( )。

25.22 gets函数

```
#include <cstdio>
char *gets(char *str);
```

gets( )函数从stdin中读取字符并将这些字符放入str指定的字符数组。在遇到一个换行符和一个EOF之前，函数将一直读取字符。换行符不作为字符串的一部分，相反，它将被转换为一个null以便终止字符串。

如果运行成功，gets( )返回str；如果失败，则返回一个空指针。如果出现读取错误，那么str所指的数组内容将是不确定的。因为无论是出现一个错误还是到达了文件尾，函数都将返回一个空指针，所以应该使用feof( )或ferror( )确定究竟出现了哪一种情况。

因为无法限制gets( )将要读取的字符数，因此str所指的数组有可能会越界，所以gets( )具有一定的危险性。

与gets( )相关的函数有fputs( )，fgetc( )，fgets( )和puts( )。

25.23 perror函数

```
#include <cstdio>
void perror(const char *str);
```

perror( )函数把全局变量errno的值映射为一个字符串并将该字符串写到stderr。如果str的值不为空，函数将先将其写出，然后加上一个分号，再后给出实现定义的错误信息。

25.24 printf函数

```
#include <cstdio>
int printf(const char *format, ...);
```

printf( )函数把一些参数写到stdout，这些参数由format所指的字符串指定的参数列表组成。

format所指的字符串由两类元素组成。一类由将在屏幕上打印的字符组成，另一类包含定义参数显示方式的格式说明符。格式说明符以百分号（%）开始，后跟格式码。参数的个数必

须与格式说明符的个数相同，格式说明符的顺序也要与参数的顺序相匹配。例如，下面的printf( )调用将显示 "Hi c 10 there!"。

```
printf("Hi %c %d %s", 'c', 10, "there!");
```

如果没有足够的参数与格式说明符相匹配，输出将是不确定的；如果参数的个数多于格式说明符的个数，多余的参数将被丢弃。表 25.2 列出了格式说明符。

printf( )函数返回实际打印的字符数。如果返回一个负值，说明发生了一个错误。

格式码可以包含一些指定域宽、精度和左对齐标记的修饰符。放置在%和格式码之间的一个整数可以作为最小域宽说明符。它可以用空格或0填充输出以确保达到某个最小长度。如果字符串或数字大于最小长度，则将全部打印。默认情况下，输出时使用空格填充。如果希望用0填充，则应在域宽说明符前面放一个0。例如，%05d 将用0对小于5位数的数字进行填充，从而使总长度达到5。

精度修饰符的具体含义依赖于被修饰的格式码。为了添加精度修饰符，可以在域宽说明符后面放一个小数点，后面加上精度值。对于e、E和f格式，精度修饰符确定打印的小数位数。例如，%10.4f将至少显示10个字符，其中有4个小数位。当将精度修饰符应用于g或G格式码时，可以决定显示的有意义的最大数字位数。当将其应用于整型数时，精度修饰符指定将被显示的最小数字位数。如果需要，还会添加前导零。

当把精度修饰符应用于字符串时，跟在句点后面的数字将指定最大域长。例如，%5.7s将显示一个字符串，该字符串至少为5个字符长，而且不能超过7个字符。如果字符串的长度大于最大域宽，其尾部的字符将被删掉。

表 25.2　printf( )函数的格式说明符

代码	格式
%c	字符
%d	有符号的十进制整型数
%i	有符号的十进制整型数
%e	科学计数法（e小写）
%E	科学计数法（E大写）
%f	十进制浮点数
%g	使用%e或%f中较短的一种（如果使用%e，应是小写e）
%G	使用%E或%f中较短的一种（如果使用%E，应是大写E）
%o	无符号八进制数
%s	字符串
%u	无符号十进制整型数
%x	无符号十六进制数（小写字母）
%X	无符号十六进制数（大写字母）
%p	显示一个指针
%n	相关参数是一个指向整型数的指针，该整数值是到目前为止所写的字符数
%%	打印一个%号

默认情况下，所有输出都是右对齐的：如果域宽大于被打印的数据，数据将被放置在域的右边缘。你可以在%后面加一个负号，从而强制性地使显示的数据左对齐。例如，%-10.2f将在10个字符的域中左对齐一个具有两个小数位的浮点数。另外，还有两个格式修饰符允许printf( )显

示短整型数和长整型数，它们可以被用于 d、i、o、u 和 x 类型说明符。修饰符 l 告诉 printf( )将跟着一个长整型数据，例如，%ld 是指显示一个长整型数；修饰符 h 告诉 printf( )显示一个短整型数，因此，%hu 是指数据是无符号的短整型数。

如果使用的是支持 1995 年添加的宽字符功能的现代编译器，则可以将修饰符 l 和说明符 c 一起使用以指示一个 wchar_t 类型的宽字符。还可以将修饰符 l 和格式命令 s 一起使用以指示一个宽字符串。

修饰符 L 可以作为浮点指令符 e、f 和 g 的前缀，它也说明后跟一个 long double 型的数据。

%n 命令把遇到 %n 时已被写出的字符个数放到一个整型变量中，其指针在参数列表中指定。例如，下面的代码片断在显示 This is a test 文本行之后显示数字 14：

```
int i;

printf("This is a test%n", &i);
printf("%d", i);
```

你可以对说明符 n 使用修饰符 l 或 h，从而说明相应的参数是指向一个长整型数，还是指向一个短整型数。

当与某些 printf( ) 格式码一同使用时，# 具有一个特殊的含义。如果在 g、G、f、e 或 E 代码前加一个 #，可以确保显示一个小数点（即使没有小数位）。如果在格式码 x 或 X 之前加一个 #，则在打印十六进制数时在该数前加上前缀 0x。如果在 o 前加上一个 #，则在打印八进制数时在该数前加上前缀 0。# 不能用于其他的格式说明符。

我们可以通过printf( )的参数而不是通过常量提供最小域宽和精度说明符。为了达到这个目的，可以将 * 用做占位符。当扫描格式字符串时，printf( )将按照参数的出现顺序把每一个 * 与相应的参数进行匹配。

与 printf( )相关的函数有 scanf( )和 fprintf( )。

25.25 putc 函数

```
#include <cstdio>
int putc(int ch, FILE *stream);
```

putc( )函数把包含在具有最少有效字节的ch 中的字符写到stream指定的输出流。因为字符参数在调用时将被提升为整型，所以可以把字符值用做 putc( )的参数。

如果运行成功，putc( )函数将返回写出的字符；如果出现错误，则返回 EOF。如果输出流以二进制模式打开，EOF 是一个有效的 ch 值。这意味着，必须利用 ferror( )确定是否发生了错误。

与 putc( )相关的函数有 fgetc( )，fputc( )，getchar( )和 putchar( )。

25.26 putchar 函数

```
#include <cstdio>
int putchar(int ch);
```

putchar( )函数把包含在具有最少有效字节的 ch 中的字符写到 stdout。这个函数的功能与 putc(ch, stdout)相当。因为字符参数在调用时将被提升为整型，所以可以把字符值用做

putchar( )的参数。

如果运行成功，putchar( )函数将返回写出的字符；如果出现错误，则返回 EOF。

与 putchar( )相关的函数是 putc( )。

25.27　puts 函数

```
#include <cstdio>
int puts(const char *str);
```

puts( )函数把 str 指定的字符串写到标准的输出设备，而且把空终结符转换为换行符。

如果运行成功，puts( )函数返回一个非负值；否则，返回一个 EOF。

与 puts( )相关的函数是 putc( )，gets( )和 printf( )。

25.28　remove 函数

```
#include <cstdio>
int remove(const char *fname);
```

remove( )函数删除 fname 指定的文件。如果文件被成功删除，函数返回 0；如果出现错误，则返回一个非 0 值。

与 remove( )相关的函数是 rename( )。

25.29　rename 函数

```
#include <cstdio>
int rename(const char *oldfname, const char *newfname);
```

rename( )函数把 oldfname 指定的文件名改为 newfname 指定的文件名。newfname 不必与现有的目录项相匹配。

如果运行成功，rename( )函数返回 0；如果出现错误，则返回一个非 0 值。

与 rename( )相关的函数是 remove( )。

25.30　rewind 函数

```
#include <cstdio>
void rewind(FILE *stream);
```

rewind( )函数把文件位置指示符移到指定流的起始处。此外，该函数还清除文件结束符和与 stream 相关联的错误标记。这个函数没有返回值。

与 rewind( )函数相关的函数是 fseek( )。

25.31　scanf 函数

```
#include <cstdio>
int scanf(const char *format, ...);
```

scanf( )函数是一个具有多种用途的输入例程，它从 stdin 流读取数据并把数据存储在参数列表指定的变量中。该函数可以读取所有的内置数据类型并将它们自动转换为适当的内部格式。

format指定的控制字符串由以下3类字符组成：

格式说明符
空白字符
非空白字符

输入格式说明符由一个%号开始，它告诉scanf( )下一个要读取的数据类型。表25.3列出了格式说明符。例如，%s读取一个字符串，而%d读取一个整型数。格式字符串按照从左至右的顺序读取，格式说明符按顺序与组成参数列表的参数相匹配。

表25.3 scanf( )的格式说明符

代码	含义
%c	读取单个字符
%d	读取一个十进制整型数
%i	读取一个整型数
%e	读取一个浮点数
%f	读取一个浮点数
%g	读取一个浮点数
%o	读取一个八进制数
%s	读取一个字符串
%x	读取一个十六进制数
%p	读取一个指针
%n	接收一个整型值，该值等于到目前为止所读取的字符个数
%u	读取一个无符号整型数
%[ ]	扫描一组字符
%%	读取一个百分号

为了读取一个长整型数，应在格式说明符前放置一个l；为了读取一个短整型数，应在格式说明符前放置一个h。这些修饰符可以与格式码d，i，o，u和x一同使用。

默认情况下，f，e和g说明符通知scanf( )把数据赋给一个float。如果在这些说明符之前加上一个l，scanf( )将把数据赋给一个double。我们可以用L告诉scanf( )接收数据的变量是一个long double型。

如果正在使用支持1995年添加的宽字符功能的现代编译器，则可以将修饰符l和格式码c一同使用以表示一个指向wchar_t类型的宽字符的指针。你也将修饰符l和格式码s一同使用以表示一个指向宽字符串的指针。l还可以用来把一个扫描集（scanset）修改为表示宽字符。

格式字符串中的空白字符将使scanf( )跳过输入流中的一个或多个空白字符。一个空白字符既可以是一个空格、一个列表字符，也可以是一个换行符。从本质上讲，控制字符串中的一个空白字符将使scanf( )读取（但不能存储）任意多个（包括0个）空白字符，直到遇到第一个非空白字符。

格式字符串中的非空白字符将使scanf( )读取并删除一个匹配字符，例如%d。%d将使scanf( )先读取一个整型数，然后读取并删除一个逗号，最后读取另一个整型数。如果没有发现指定的字符，scanf( )将终止执行。

所有通过scanf( )接收值的变量必须根据它们的地址传递。这意味着所有的参数必须是指针。

输入的数据项必须被空格、制表符或换行符分隔，像逗号和分号之类的标点符号不算做分

隔符。这意味着下面的代码行将接受 10 20 这样的输入，但不能接受 10,20 这样的输入：

```
scanf("%d%d", &r, &c);
```

放置在 % 之后和格式码之前的星号（*）将读取指定类型的数据，但禁止对其赋值。因此，假定输入为 10/20，下面的指令将把 10 放入 x，把 20 放入 y：

```
scanf("%d%*c%d", &x, &y);
```

格式指令可以指定一个最大的域宽修饰符，该修饰符是一个放置在 % 和格式码之间的整数，在读取数据时，它将限制所有域的字符个数。例如，如果希望读取到 address 中的字符不能超过 20 个，可编写下面的代码：

```
scanf("%20s", address);
```

如果输入流大于20个字符，下一次调用输入时将从此次调用停止的地方开始。如果遇到一个空白字符，输入可能在到达最大域宽之前终止。如果是这种情况，scanf( ) 将移到下一个域。

尽管空格、制表符和换行符被用做域的分隔符，但当读取单一字符时，这些字符将像所有其他字符一样被读取。例如，对于输入流 x y，下面的代码将把字符 x 读入 a，把空格读入 b，把字符 y 读入 c。

```
scanf("%c%c%c", &a, &b, &c);
```

注意：控制字符串中的所有其他字符（包括空格、制表符和换行符）将被用于与输入流中的字符进行匹配并被丢弃。也就是说，所有与其匹配的字符将被丢弃。例如，假定输入流是 10t20，下面的代码将把 10 放入 x，把 20 放入 y，而因为 t 包含在控制字符串中，所以 t 将被丢弃：

```
scanf("%dt%d", &x, &y);
```

scanf( ) 的另一个功能称为扫描集（scanset）。一个扫描集定义一个字符集，该字符集将被 scanf( ) 所读取并被分配给相应的字符数组。可以通过把想要扫描的字符放入方括号中来定义一个扫描集，此外还要在方括号之前加上一个百分号。例如，下面的扫描集告诉 scanf( ) 只读取字符 A、B 和 C：

```
%[ ABC]
```

当使用一个扫描集时，scanf( ) 继续读取字符并将其放入相应的字符数组，直到遇到一个不在扫描集中的字符为止。此外，相应的变量必须是一个指向一个字符数组的指针。数组将根据 scanf( ) 的返回值包含一个由所读取的字符组成、并以 null 结束的字符串。

如果集合中的第一个字符是 ^，则可以指定一个倒置（inverted）集。当第一个字符是 ^ 时，它告诉 scanf( ) 接受所有没有被扫描集定义的字符。

对于许多实现来说，可以用连字符（-）指定一个范围。例如，下面的代码告诉 scanf( ) 接受 A ~ Z 的所有字符。

```
%[ A-Z]
```

还要记住的一点是：扫描集是大小写敏感的。因此，如果既想扫描大写字符，又想扫描小写字符，则必须对其分别予以指定。

scanf( ) 函数将返回被成功赋值的域的个数。这个值不包括那些只被读取而未被赋值（由于

使用了*修饰符而禁止赋值)的域。如果在第一个域被赋值之前发生了错误，函数将返回EOF。

与scanf()相关的函数有printf()和fscanf()。

25.32 setbuf函数

```
#include <cstdio>
void setbuf(FILE *stream, char *buf);
```

setbuf()函数既可以指定stream将要使用的缓冲区(如果buf为空)，也可以关闭缓冲区。如果指定了一个程序员定义的缓冲区，那它的长度必须为BUFSIZ。BUFSIZ在<cstdio>中定义。

setbuf()函数没有返回值。

与setbuf()相关的函数有fopen()，fclose()和setvbuf()。

25.33 setvbuf函数

```
#include <cstdio>
int setvbuf(FILE *stream, char *buf, int mode, size_t size);
```

setvbuf()函数允许编程者为特定的流指定缓冲区及其大小和模式。buf所指的字符数组被用做I/O操作的流式缓冲区，缓冲区的大小由size设置，mode确定处理缓冲区的方式。如果buf为空，setvbuf()将分配自己的缓冲区。

mode的有效值为_IOFBF、_IONBF和_IOLBF，这些值在<cstdio>中定义。当mode被设置为_IOFBF时，将采用全缓冲方式；如果mode为_IOLBF，将采用行缓冲方式。也就是说，每次往输出流写出一个换行符时，缓冲区都将被刷新。而对输入流来说，在读到换行符之前，输入都将被缓冲；如果mode为_IONBF，则不采取任何缓冲。

如果运行成功，setvbuf()函数返回0；如果失败，返回一个非0值。

与setvbuf()相关的函数有setbuf()。

25.34 sprintf函数

```
#include <cstdio>
int sprintf(char *buf, const char *format, ...);
```

除了把输出放入buf指定的数组而不是写到控制台之外，sprintf()函数与printf()是一样的。详细情况参见printf()。

该函数的返回值等于实际放入数组的字符数。

与sprintf()相关的函数有printf()和fsprintf()。

25.35 sscanf函数

```
#include <cstdio>

int sscanf(const char *buf, const char *format, ...);
```

除了从buf指定的数组而不是从stdin读取字符外，sscanf()函数的功能与scanf()相同。详细内容参见scanf()。

函数的返回值等于实际被赋值的变量值。该值不包括使用格式指令修饰符*跳过的域。返

回值为 0 表示没有给任何域赋值，返回值为 EOF 表示在第一次赋值之前发生了一个错误。

与 sscanf( )相关的函数有 scanf( )和 fscanf( )。

25.36　tmpfile 函数

```
#include <cstdio>
FILE *tmpfile(void);
```

tmpfile( )函数打开一个用于更新的临时文件并返回一个指向该流的指针。该函数将自动地使用惟一的文件名以避免和现有的文件发生冲突。

如果运行失败，tmpfile( )函数返回一个空指针；否则，返回一个指向相应的流的指针。

当关闭文件或终止程序时，tmpfile( )创建的临时文件被自动删除。

与 tmpfile( )相关的函数有 tmpnam( )。

25.37　tmpnam 函数

```
#include <cstdio>
char *tmpnam(char *name);
```

tmpnam( )函数生成一个惟一的文件名并将其存入 name 指定的数组，该数组至少是 L_tmpnam 个字符长（L_tmpnam 在<cstdio>中定义）。tmpnam( )函数的主要用途是生成一个临时文件名，该文件名不同于当前磁盘目录中所有其他的文件名。

该函数最多可以被调用 TMP_MAX 次。TMP_MAX 在<cstdio>中定义，它最少应该为 25。每次调用 tmpnam( )时，它将生成一个新的临时文件名。

如果运行成功，函数将返回一个指向 name 的指针；否则，将返回一个空指针。如果 name 为空，临时文件名将被存放在 tmpnam( )所特有的一个静态数组中，该数组在下一次调用此函数时被重写。

与 tmpnam( )相关的函数有 tmpfile( )。

25.38　ungetc 函数

```
#include <cstdio>
int ungetc(int ch, FILE *stream);
```

ungetc( )函数把由 ch 的低位字节指定的字符返回到输入流 stream 中，这个字符将被下一个读取 stream 的操作获得。通过调用 fflush( )，fseek( )或 rewind( )，可以撤销一个 ungetc( )操作并丢弃相应的字符。

这里可以保证能够反推一个字符，然而，有些实现可以接受更多的反推操作。

不能将 ungetc( )用于 EOF。

通过调用 ungetc( )，可以清除与指定流相关联的文件结束标记。在所有反推的字符被读取之前，不能定义下一个文本流的文件位置指示符的值，此时它将与第一次调用 ungetc( )时的情况相同。对于二进制流，每次调用 ungetc( )后将使文件位置指示符减去 1。

如果运行成功，函数返回值为 ch；如果失败，则返回 EOF。

与 ungetc( )相关的函数有 getc( )。

25.39 vprintf，vfprintf和vsprintf函数

```
#include <cstdarg>
#include <cstdio>
int vprintf(char *format, va_list arg_ ptr);
int vfprintf(FILE *stream, const char *format,
             va_list arg_ ptr);
int vsprintf(char *buf, const char *format,
             va_list arg_ ptr);
```

除了参数列表被一个指向参数列表的指针代替之外，函数vprintf( )，vfprintf( )和vsprintf( )的功能分别与printf( )，fprintf( )和sprintf( )的功能相当。这个指针的类型必须是va_list，该类型在头文件<cstdarg>（C的头文件是stdarg.h）中定义。

与这些函数相关的函数有va_arg( )，va_start( )和va_end( )。