目录：

附录 B：标准库 返回目录

本附录总结了 ANSI 标准定义的函数库。标准库不是 C 语言本身的构成部分，但是支持标准 C 的实现会提供该函数库中的函数声明、类型及宏定义。在这部分内容中，我们省略了一些使用比较受限的函数以及一些可以通过其他函数简单合成的函数，也省略了多字节字符的内容，同时，也不准备讨论与区域相关的一些属性，也就是与本地语言、国籍或文化相关的属性。

标准库中的函数、类型以及宏分别在下面的标准头文件中定义：

<assert.h>	<float.h>	<math.h>	<stdarg.h>	<stdlib.h>
<ctype.h>	<limits.h>	<setjmp.h>	<stddef.h>	<string.h>
<errno.h>	<local.h>	<signal.h>	<stdio.h>	<time.h>

可以通过下列方式访问头文件：

#include <头文件>

头文件的包含顺序是任意的，并可包含任意多次。头文件必须被包含在任何外部声明或定义之外，并且，必须在使用头文件中的任何声明之前包含头文件。头文件不一定是一个源文件。

以下划线开头的外部标识符保留给标准库使用，同时，其他所有以一个下划线和一个大写字母开头的标识符以及两个下划线开头的标识符也都保留给标准库使用。

B. 1 输入输出：<stdio.h> 返回目录

头文件<stdio.h>中定义的输入和输出函数、类型以及宏的数目几乎占整个标准库的三分之一。

流（stream）是与磁盘或其他外围设备关联的数据的源或目的地。尽管在某些系统中（如著名的 UNIX 系统中），文本流和二进制流是相同的，但标准库仍然提供了这两种类型的流。文本流是由文本行组成的序列，每行包含 0 个或多个字符，并以‘\n’结尾。在某些环境中，可能需要将文本流转换为其他表示形式（例如把‘\n’映射成回车符和换行符），或从其他表示形式转换为文本流。二进制流是由未经处理的字节构成的序列，这些字节记录着内部数据，并具有下列性质：如果在同一系统中写入二进制流，然后再读取该二进制流，则读出和写入的内容完全相同。

打开一个流，将把流与一个文件或设备连接起来，关闭流将断开这种连接。打开一个文件将返回一个指向 FILE 类型对象的指针，该指针记录了控制该流的所有必要信息。在不引起歧义的情况下，我们在下文中将不再区分“文件指针”和“流”。

程序开始执行是，stdin、stdout 和 stderr 这 3 个流已经处于打开状态。

B.1.1 文件操作 返回目录

下列函数用于处理与文件有关的操作。其中，类型 size_t 是由运算符 sizeof 生成的无符号整形。

```
FILE *fopen(const char *filename, const char *mode)
```

fopen 函数打开 filename 指定的文件，并返回一个与之相关联的流。如果打开失败，则返回 NULL。

访问模式 mode 可以为下列合法值之一：

“r”	打开文本文件用于读
“w”	创建文本文件用于写，并删除已存在的内容（如果有的话）
“a”	追加；打开或创建文本文件，并向文件末尾追加内容
“r+”	打开文本文件用于更新（即读和写）
“w+”	创建文本文件用于更新，并删除已存在的内容（如果有的话）
“a+”	追加；打开或创建文本文件用于更新，写文件时追加到文件末尾

后三种方式（更新方式）允许对同一个文件进行读和写。在读和写的交叉过程中，必须调用 fflush 函数或文件定位函数。如果在上述访问模式之后再加上 b,如“rb”或“w+b”等，则表示对二进制文件进行操作。文件名 filname 限定最多为 FILENAME_MAX 个字符。一次最多可打开 FOPEN_MAX 个文件。

```
FILE *freopen(const char *filename, const char *mode, FILE *stream)
```

freopen 函数以 mode 指定的模式打开 filename 指定的文件，并将该文件关联到 stream 指定的流。它返回 stream；若出错则返回 NULL。freopen 函数一般用于改变与 stdin、stdout 和 stderr 相关联的文件。

int fflush(FILE *stream)

对输出流来说，fflush 函数将已写到缓冲区但尚未写入文件的所有数据写到文件中。对输入流来说，其结果是未定义的。如果在写的过程中发生错误，则返回 EOF，否则返回 0。fflush(NULL)将清洗所有的输出流。

int fclose(FILE *stream)

fclose 函数将所有未写入的数据写入 stream 中，丢弃缓冲区中的所有未读输入数据，并释放自动分配的全部缓冲区，最后关闭流。若出错则返回 EOF，否则返回 0。

int remove(const char *filename) 返回目录

remove 函数删除 filename 指定的文件，这样，后续试图打开该文件的操作将失败。如果删除操作失败，则返回一个非 0 值。

int rename(const char *oldname, const char *newname)

rename 函数修改文件的名字。如果操作失败，则返回一个非 0 值。

FILE *tmpfile(void)

tmpfile 函数以模式“w+b”创建一个临时文件，该文件在被关闭或程序正常结束时将被自动删除。如果创建操作成功，该函数返回一个流；如果创建文件失败，则返回 NULL。

char *tmpnam(char s[L_tmpnam]) 返回目录

tmpnam(NULL)函数创建一个与现有文件名不同的字符串，并返回一个指向一内部静态数组的指针。tmpname(s)函数把创建的字符串保存到数组 s 中，并将它作为函数值返回。s 中至少要有 L_tmpnam 个字符的空间。tmpnam 函数在每次被调用时均生成不同的名字。在程序执行的过程中，最多只能确保生成 TMP_MAX 个不同的名字。注意，tmpnam 函数只能用于创建一个名字，而不能创建一个文件。

int setvbuf(FILE *stream, char *buf, int mode, size_t size)

setvbuf 函数控制流 stream 的缓冲。在执行读、写以及其他任何操作之前必须调用此函数。当 mode 的值为_IOFBF 时，将进行完全缓冲。当 mode 的值为_IOLBF 时，将对文本文件进行行缓冲，当 mode 的值为_IONBF 时，表示不设置缓冲。如果 buf 的值不是 NULL，则 setvbuf 函数将 buf 指向的区域作为流的缓冲区，否则将分配一个缓冲区。size 决定缓冲区的长度。如果 setvbuf 函数出错，则返回一个非 0 值。

void setbuf(FILE *stream, char *buf)

如果 buf 的值为 NULL，则关闭流 stream 的缓冲；否则 setbuf 函数等价于（void）setvbuf(stream, buf, _IOFBF, BUFSIZ)。

B．1．2 格式化输出 返回目录

printf 函数提供格式化输出转换。

int fprintf(FILE *stream, const char *format, …)

fprintf 函数按照 format 说明的格式对输出进行转换，并写到 stream 流中。返回值是实际写入的字符数。若出错则返回一个负值。

格式串由两种类型的对象组成：普通字符（将被复制到输出流中）与转换说明（分别决定下一后续参数的转换和打印）。每个转换说明均以字符%开头，以转换字符结束。在%与转换字符之间可以依次包含下列内容：

◆标志（可以以任意顺序出现），用于修改转换说明

-	指定被转换的参数在其字段内左对齐
+	指定在输出的数前面加上正负号
空格	如果第一个字符不是正负号，则在其前面加上空格
0	对于数值转换，当输出长度小于字段宽度时，添加前导 0 进行填充
#	指定另一种输出形式。如果为 O 转换，则第一个数字为零；如果为 x 或 X 转换，指定在输出的非 0 值前加 0x 或 0X；对于 e、E、f、g 或 G 转换，指定输出总包括一个小数点；对于 g 或 G 转换，值尾部无意义的 0 将被保留

◆一个数值，用于指定最小字段宽度。转换后的参数输出宽度至少要达到这个数值。如果参数的字符数小于此数值，则在参数左边（如果要求左对齐的话则为右边）填充一些字符。填充字符通常为空格，但是，如果设置了 0 填充标志，则填充字符为 0。

◆点号，用于分隔字段宽度的精度。

◆表示精度的数。对于字符串，它指定打印的字符的最大个数；对于 e、E 或 f 转换，它指定打印的小数点后的数字位数；对于 g 或 G 转换，它指定打印的有效数字位数；对于整型数，它指定打印的数字位数（必要时可加填充为 0 以达到要求的精度）。

◆长度修饰符 h、l 或 L。“h”表示将相应的参数按 short 或 unsigned short 类型输出。“l”表示将相应的参数按 long 或 unsigned long 类型输出；“L”表示将相应的参数按照 long double 类型输出。

宽度和精度中的一个或两个都可以用*指定，这种情况下，该值将通过转化下一个参数计算得到（下一个参数必须为 int 类型）。

表 B-1 中列出了这些转换字符及其意义。如果%后面的字符不是转换字符，则其行为没有定义。

表 B-1 printf 函数的转换字符

转换字符	参数类型；转换结果
d，i	int；有符号十进制表示
o	unsigned int；无符号八进制表示（无前导 0）
x， X	unsigned int；无符号十六进制表示（无前导 0x 和 0X）。如果是 0x，则使用 abcdef，如果是 0X，则使用 ABCDEF
u	int；无符号十进制表示
c	int；转换为 unsigned char 类型后为一个字符
s	char *；打印字符串中的字符，直到遇到‘\0’或者已打印了由精度指定的字符数
f	double；形式为[-]mmm.ddd 的十进制表示，其中，d 的数目由精度确定，默认精度为 6。精度为 0 时，不输出小数点
e，E	double；形式为[-]m.dddddd e±xx 或[-]m.dddddd E±xx 的十进制表示。d 的数目由精度确定，默认精度为 6。精度为 0 时不输出小数点
g，G	double；当指数小于-4 或大于等于精度是，采用%e 或%E 的格式，否则采用%f 的格式。尾部的 0 与小数点不打印
p	void *；打印指针值（具体表示方法与实现有关）
n	int *；到目前为止，此 printf 调用输出的字符的数目将被写入到相应参数中。不进行参数转换
%	不进行参数转换；打印一个符号%

```
int printf(const char *format, …)
```
返回目录

printf(…)函数等价于 fprintf(stdout, …)。

```
int sprintf(char *s, const char *format, …)
```

sprintf 函数与 printf 函数基本相同，但其输出将被写到字符串 s 中，并以‘\0’结束。s 必须足够大，以足够容纳输出结果。该函数返回实际输出的字符数，不包括‘\0’。

```
int vprintf(const char *format, va_list arg)
int vfprintf(FILE *stream, const char *format, va_list arg)
int vsprintf(char *s, const char *format, va_list arg)
```

vprintf、vfprintf、vsprintf 这三个函数分别于对应的 printf 函数等价，但它们用 arg 代替了可变参数表。arg 有宏 va_start 初始化，也可能由 va_arg 调用初始化。详细的信息参见 B.7 节中对<stdarg.h>头文件的讨论。

B．1．3 格式化输入

返回目录

scanf 函数处理格式化输入转换。

```
int fscanf(FILE *stream, const char *format, …)
```

fscanf 函数根据格式串 format 从流 stream 中读取输入，并把转换后的值赋给后续各个参数，其中的每个参数都必须是一个指针。当格式串 format 用完时，函数返回。如果到达文件的末尾或在转换输入前出错，该函数返回 EOF；否则，返回实际被转换并赋值的输入项的数目。

格式串 format 通常包括转换说明，它用于指导对输入进行解释。格式字符串中可以包含下列项目：

◆空格或制表符

◆普通字符（% 除外），它将与输入流中下一个非空白字符进行匹配

◆转换说明，由一个%、一个赋值屏蔽字符*（可选）、一个指定最大字段宽度的数（可选）、一个指定目标字段宽度的字符（h、l 或 L）（可选）以及一个转换字符组成。

转换说明决定了下一个输入字段的转换方式。通常结果将被保存在由对应参数指向的变量中。但是，如果转换说明中包含赋值屏蔽字符*，例如%*s，这将跳过对应的输入字段，并不进行赋值。输入字段是一个由非空白字符组成的字符串，当遇到下一个空白字符或达到最大字段宽度（如果有的话）时，对当前输入字段的读取结束。这意味着，scanf 函数可以跨越行的边界读取输入，因为换行符也是空白符（空白符包括空格、横向制表符、纵向制表符、换行符、回车符和换页符）。

换行字符说明了对输入字段的解释方式。对应的参数必须是指针。合法的转换字符**如表 B-2** 所示。

如果参数是指向 short 类型而非 int 类型的指针，则在转换字符 d、i、n、o、u 和 x 之前可以加上前缀 h。如果参数是指向 long 类型的指针，则在这几个转换字符前可以加上字母 l。如果参数是指向 double 类型而非 float 类型的指针，则在转换字符 e、f 和 g 前可以加上字母 l。如果参数是指向 long double 类型的指针，则在转换字符 e、f 和 g 前可以加上字母 L。

表 B-2 scanf 函数的转换字符

转换字符	输入数据；参数类型
d	十进制整型数；int *
i	整型数；int *。该整型数可以是八进制数（以 0 打头）或十六进制数（以 0x 或 0X 打头）
o	八进制整型数（可以带或不带前导 0）；int *
u	无符号十进制整型数；unsigned int *
x	十六进制整型数（可以带或不带前导 0x 或 0X）；int *
c	字符；char *，按照字段宽度的大小把读取的字符保存到指定的数组中，不增加字符‘\0’字段宽度的默认值为 1。在这种情况下，读取输入时将不跳过空白符，如果要读取下一个非空白字符，可以使用%ls
s	由空白符组成的字符串（不包含引号）；char *。它指向一个字符数组，该字符数组必须有足够的空间，以保存该字符串以及在尾部添加的‘\0’字符
e、f、g	浮点数；float *。float 类型浮点数的输入格式为：一个可选的正负号、一个可能包含小数点的数字串、一个可选的指数字段（字母 e 或 E 后跟一个可能带正负号的整型数）
p	printf（“%p”）函数调用打印的指针值：void *
n	将到目前为止该函数调用读取的字符数写入对应的参数中；int *。不读取输入字符。不增加已转换的项目计数
[…]	与方括号中的字符集合匹配的输入字符中最长的非空字符串；char *。末尾将添加字符‘\0’。
[…]	与方括号中的字符集合不匹配的输入字符中最长的非空字符串；char *。末尾将添加字符‘\0’。
%	表示“%”，不进行赋值

```
int scanf(const char *format, …)
```
返回目录

scanf(…)函数与 fscanf(stdin, …)相同。

```
int sscanf(const char *s, const char *format, …)
```

sscanf(s, …)函数与 scanf(…)等价，所不同的是，前者的输入字符来源于字符串 s。

B．1．4 字符输入/输出函数

返回目录

```
int fgetc(FILE *stream)
```

fgetc 函数返 stream 流的下一个字符，返回类型为 unsigned char（被转换为 int 类型）。如果到达文件末尾或发生错误，这返回 EOF。

char *fgets(char *s, int n, FILE *stream)
fgets 函数最多将 n-1 个字符读入到数组 s 中。当遇到换行符时，把换行符读入到数组 s 中，读取过程终止。数组 s 以‘\0’结尾。fgets 函数返回数组 s。如果到达文件的末尾或发生错误，则返回 NULL。

int fputc(int c, FILE *stream)
fputc 函数把字符 c（转换为 unsigned char 类型）输出到流 stream 中。它返回写入的字符，错误则返回 EOF。

int fputs(const char *s, FILE *stream)
fputs 函数把字符串 s（不包含字符‘\n’）输出到流 stream 中；它返回一个非负值，若出错返回 EOF。

int getc(FILE *stream)
getc 函数等价于 fgetc，所不同的是，当 getc 函数定义为宏时，它可能多次计算 stream 的值。

int getchar(void)
getchar 函数等价于 getc(stdin)。

char *gets(char *s)
gets 函数把下一个输入行读入到数组 s 中，并把末尾的换行字符替换为字符‘\0’。它返回数组 s，如果到达文件末尾或发生错误，这返回 NULL。

int putc(int c, FILE *stream)
putc 函数等价于 fputc，所不同的是，当 putc 函数定义为宏时，它可能多次计算 stream 的值。

int putchar(int c)
putchar(c)函数等价于 putc（c，stdout）。

int puts(const char *s)
puts 函数把字符串 s 和一个换行字符输出到 stdout 中。如果发生错误，则返回 EOF；否则返回一个非负值。

int ungetc(int c, FILE *stream)
ungetc 函数把 c（转换为 unsigned char 类型）写回到流 stream 中，下次对该流进行操作时，将返回该字符。对每个流只能写回一个字符，且此字符不能是 EOF。ungetc 函数返回被写回的字符；如果发生错误，则返回 EOF。

B．1．5 直接输入/输出函数 返回目录

size_t fread(void *ptr, size_t size, size_t nobj, FILE *stream)
fread 函数从流 stream 中读取最大 nobj 个长度为 size 的对象，并保存到 ptr 指向的数组中。它返回读取的对象数目，此返回值可能小于 nobj。必须通过函数 feof 和 ferror 获得结果执行状态。

size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *stream)
fwrite 函数从 ptr 指向的数组中读取 nobj 个长度为 size 的对象，并输出到流 stream 中。它返回输出的对象数目。如果发生错误，返回值会小于 nobj 的值。

B．1．6 文件定位 返回目录

int fseek(FILE *stream, long offset, int origin)
fseek 函数设置流 stream 的文件位置，后续的读写操作将从新位置开始。对于二进制文件，此位置被设置为从 origin 开始的第 offset 个字符处。origin 的值可能为 SEEK_SET（文件开始处）、SEEK_CUR（当前位置）或

SEEK_END（文件结束）。对于文本流，offset 必须设置为 0，或者由函数 ftell 返回的值（此时 origin 的值必须是 SEEK_SET）。fseek 函数在出错时返回一个 0 值。

long ftell(FILE *stream)
ftell 函数返回 stream 流的当前文件位置。出错时该函数返回 -1L。

void rewind(FILE *stream)
rewind（fp）函数等价于语句 fseek（fp， 0L，SEEK_SET）；clearerr（fp）的执行结果。

int fgetpos(FILE *stream, fpos_t *ptr)
fgetpos 函数把 stream 流的当前位置记录在*ptr 中，供随后的 fsetpos 函数调用使用。若出错者返回一个非 0 值。

int fsetpos(FILE *stream, const fpos_t *ptr)
fsetpos 函数将流 stream 的当前位置设置为 fgetpos 记录在*ptr 中的位置。若出错则返回一个非 0 值。

B．1．7 错误处理函数 返回目录

当发生错误或到达文件末尾时，标准库中的许多函数都会设置状态指示符。这些状态指示符可被显示地设置和测试。另外，整型表达式 errno（在<errno.h>中声明）可能包含一个错误编号，据此可以进一步了解最近一次出错的信息。

void clearerr(FILE *stream)
clearerr 函数清除与流 stream 相关的文件结束符和错误指示符。

int feof(IFLE *stream)
如果设置了与 stream 流相关的文件结束符，feof 函数将返回一个非 0 值。

void ferror(FILE *stream)
如果设置了与 stream 流相关的错误指示符，ferror 函数将返回一个非 0 值。

void perror(const char *s)
perror（s）函数打印字符串 s 以及与 errno 中整型值对应的错误信息，错误信息的具体内容与具体的实现相关。该函数的功能类似于执行下列语句：
fprintf（stderr， “%s：%s\n”，s，“error message”）
有关函数 strerror 的信息，参见 B.3 节中的介绍。

B．2 字符类别测试：<ctype.h> 返回目录

头文件<ctypte.h>中声明了一些测试字符的函数。每个函数的参数均为 int 类型，参数的值必须是 EOF 或可用 unsigned char 类型表示的字符，函数的返回值为 int 类型。如果参数 c 满足指定的条件，则函数返回非 0 值（表示真），否则返回 0（表示假）。这些函数包括：

isalnum(c)	函数 isalpha(c)或 isadigit(c)为真
isalpha(c)	函数 isupper(c)或 islower(c)为真
iscntrl(c)	c 为控制字符
isdigit(c)	c 为十进制数字
isgraph(c)	c 为除空格的可打印字符
islower(c)	c 是小写字母
isprint(c)	c 是包括空格的可打印字符
ispunct(c)	c 是出空格，字母和数字外的可打印字符
isspace(c)	c 是空格，换页符，换行符，回车符，横向制表符或纵向制表符
isupper(c)	c 是大写字母
isxdigit(c)	c 是十六进制数字

在 7 位 ASCII 字符集中，可打印字符是 0x20（‘ ’）到 0x7E（‘~’）之间的字符；控制字符是从 0（NULL）到 0x1F（US）之间的字符以及字符 0x7F（DEL）。

```
int tolower(int c)      将 c 转换为小写字母
int toupper(int c)      将 c 转换为大写字母
```

如果 c 是大写字母，则 tolower（c）返回相应的小写字母，否则返回 c。如果 c 是小写字母，则 toupper（c）返回相应的大写字母，否则返回 c。

B. 3 字符串函数：<string.h>

返回目录

头文件<string.h>中定义了两组字符串函数。第一组函数的名字以 str 开头；第二组函数的名字以 mem 开头。除函数 memmove 外，其他函数都没有定义重叠对象间的复制行为。比较函数将把参数作为 unsigned char 类型的数组看待。

在下表中，变量 s 和 t 的类型为 char *；cs 和 ct 的类型为 const char *；n 的类型为 size_t；c 的类型为 int（将被转换为 char 类型）。

char *strcpy(s, ct)	将字符串 ct（包括'\0'）复制到字符串 s 中，并返回 s
char *strncpy(s, ct, n)	将字符串 ct 中最多 n 个字符复制到字符串 s 中，并返回 s。如果 ct 中少于 n 个字符，则用'\0'填充
char *strcat(s, ct)	将字符串 ct 连接到字符串 s 的尾部，并返回 s
char *strncat(s, ct, n)	将字符串 ct 中最多 n 个字符连接到字符串 s 的尾部，并以'\0'结束；函数返回 s
int strcmp(cs, ct)	比较字符串 cs 和 ct；当 cs<ct 时，返回一个负数；当 cs==ct 时，返回 0；当 cs>ct 时，返回正数
int strncmp(cs, ct, n)	比较字符串 cs 中至多前 n 个字符与字符串 ct 相比较；当 cs<ct 时，返回一个负数；当 cs==ct 时，返回 0；当 cs>ct 时，返回正数
char *strchr(cs, c)	返回指向字符 c 在字符串 cs 中第一次出现的位置的指针；如果 cs 中不包含 c，则返回 NULL
char *strrchr(cs, c)	返回指向字符 c 在字符串 cs 中最后一次出现的位置的指针；如果 cs 中不包含 c，则返回 NULL
size_t strspn(cs, ct)	返回字符串 cs 中包含 ct 中的字符的前缀长度
size_t strcspn(cs, ct)	返回字符串 cs 中不包含 ct 中的字符的前缀长度
char *strpbrk(cs, ct)	返回一个指针，它指向字符串 ct 中的任意字符第一次出现在字符串 cs 中的位置；如果 cs 中没有与 ct 相同的字符，则返回 NULL
char *strstr(cs, ct)	返回一个指针，他指向字符串 ct 中第一次出现在字符串 cs 中的位置；如果 cs 中不包含字符串 ct，则返回 NULL
size_t strlen(cs)	返回字符串 cs 的长度
char *strerror(n)	返回一个指针，它指向与错误编号 n 对应的错误信息字符串（错误信息的集体内容与具体实现相关
char *strtok(s, ct)	strtok 函数在 s 中搜索由 ct 中的字符界定记号。详细信息参见下面讨论

对 strtok（s，ct）进行一系列调用，可以把字符串 s 分成许多记号，这些记号以 ct 中的字符为分界符。第一次调用时，s 为非空。它搜索 s，找到不包含 ct 中字符的第一个记号，将 s 中的下一个字符替换为‘\0’，并返回指向记号的指针。随后，每次调用 strtok 函数时（由 s 的值是否为 NULL 指示），均返回下一个不包含 ct 中字符的记号。当 s 中没有这样的记号时，返回 NULL。每次调用时字符串 ct 可以不同。

以 mem 开头的函数按照字符数组的方式操作对象，其主要目的是提供一个高效的函数接口。在下表列出的函数中，s 和 t 的类型均为 void *，cs 和 ct 的类型均为 const void *，n 的类型为 size_t， c 的类型均为 int（将被转换为 unsigned char 类型）。

void *memcpy(s, ct, n)	将字符串 ct 中的 n 个字符拷贝到 s 中，并返回 s
void *memmove(s, ct, n)	该函数的功能与 memcpy 相似，所不同是，当对象重叠时，该函数仍能正确执行
void memcmp(cs, ct, n)	将 cs 的前 n 个字符与 ct 进行比较，其返回值与 strcmp 的返回值相同

void *memchar(cs, c, n)	返回一个指针，它指向 c 在 cs 中第一次出现的位置。如果在 cs 的前 n 个字符中找不到匹配，则返回 NULL
void *memset(s, c, n)	将 s 中的前 n 个字符替换为 c，并返回 s

B. 4 数学函数：<math.h>

返回目录

头文件<math.h>中声明了一些数学函数和宏。

宏 EDOM 和 ERANGE（在头文件<error.h>中声明）是两个非 0 整型常量，用于指示函数的定义域错误和值域错误；HUGE_VAL 是一个 double 类型的正数。当参数位于函数定义的作用域之外时，就会出现定义域错误。在发生定义域错误是，全局变量 errno 的值将被设置为 EDOM，函数的返回值与具体实现相关。如果函数的结果不能用 double 类型表示，则会发生值域错误。当结果上溢时，函数返回 HUGE_VAL，并带有正确的正负号，errno 的值将被设置为 ERANGE。当结果下溢时，函数返回 0，而 errno 是否设置为 ERANGE 要视具体的实现而定。

在下表中，x 和 y 的类型为 double，n 的类型为 int，所有函数的返回值的类型均为 double。三角函数的角度用弧度表示。

sin(x)	x 的正弦值
cos(x)	x 的余弦值
tan(x)	x 的正切值
asin(x)	值域为[-π/2, π/2]，其中 x∈[-1, 1]
acos(x)	值域为[0, π]，其中 x∈[-1, 1]
atan(x)	值域为[-π/2, π/2]
atan2(y, x)	值域为[-π, π]
sinh(x)	x 的双曲正弦值
cosh(x)	x 的双曲余弦值
tanh(x)	x 的双曲正切值
exp(x)	幂函数
log(x)	自然对数 ln（x），其中 x〉0
log10(x)	以 10 为底的对数 log10（x），其中 x〉0
pow(x, y)	x 的 y 次方，如果 x=0 且 y〈=0，或者 x〈0 且 y 不是整数，都产生定义域错误
sqrt(x)	x 的平方根，其中 x〉=0
ceil(x)	不小于 x 的最小整数，其中 x 的类型为 double
floor(x)	不大于 x 的最大整数，其中 x 的类型为 double
fabs(x)	x 的绝对值
ldexp(x, n)	计算 x*（2 的 n 方）的值
frexp(x, int *exp)	
modf(x, double *ip)	
fmod(x, y)	求 x/y 的浮点余数，符号与 x 相同。如果 y 为 0，则结果与具体的实现相关

B. 5 实用函数：<stdio.h>

返回目录

头文件<stdio.h>中声明了一些执行数值转换、内存分配以及其他类似工作的函数。

double atof(const char *s)

atof 函数将字符串 s 转换为 double 类型。该函数等价于 strtod（s, （char**）NULL）。

int atoi(const char *s)

atoi 函数将字符串 s 转换为 int 类型。该函数等价于（int）strtol（s, （char**）NULL, 10）。

long atol(const char *s)

atol 函数将字符串 s 转换为 long 类型。该函数等价于 strtol（s, （char**）NULL, 10）。

double strtod(const char *s, char **endp)

strtod 函数将字符串 s 的前缀转换为 double 类型，并在转换时跳过 s 的前导空白符。除非 endp 为 NULL，否则该函数将把指向 s 中未转换部分（s 的后缀部分）的指针保存在*endp 中。如果结果上溢，则函数返回带有适当符号的 HUGE_VAL；如果结果下溢，则返回 0。在这两种情况下，errno 都将被设置为 ERANGE。

long strtol(const char *s, char **endp, int base)

strtol 函数将字符串 s 的前缀转换为 long 类型，并在转换时跳过 s 的前导空白符。除非 endp 为 NULL，否则该函数将把指向 s 中未转换部分（s 的后缀部分）的指针保存在*endp 中。如果 base 的取值在 2~36 之间，则假定输入时以该数为基地的；如果 base 的值为 0，则基底为八进制、十进制或十六进制。以 0 为前缀的是八进制，以 0x 或 0X 为前缀的是十六进制。无论在哪种情况下，字母均表示 10~base-1 之间的数字。如果 base 值是 16，则可以加前导 0x 或 0X。如果结果上溢，则函数根据结果的符号返回 LONG_MAX 或 LONG_MIN，同时将 errno 的值设置为 ERANGE。

unsigned long strtoul(const char *s, char **endp, int base)

strtoul 函数的功能与 strtol 函数相同，但其结果为 unsigned long 类型，错误值为 ULONG_MAX。

int rand(void) 返回目录

rand 函数产生一个 0~RAND_MAX 之间的伪随机整数。RAND_MAX 的取值至少为 32767。

void srand(unsigned int seed)

srand 函数将 seed 作为生成新的随机数序列的种子。种子数 seed 的初值为 1。

void *calloc(size_t nobj, size_t size)

calloc 函数为由 nobj 个长度为 size 的对象组成的数组分配内存，并返回指向分配区域的指针；若无法满足要求，则返回 NULL。该空间的初始长度为 0 字节。

void *malloc(size_t size)

malloc 函数为长度为 size 的对象分配内存，并返回指向分配区域的指针；若无法满足要求，则返回 NULL。该函数不对分配的内存区域进行初始化。

void *realloc(void *p, size_t size) 返回目录

realloc 函数将 p 指向的对象的长度修改为 size 个字节。如果新分配的内存比原内存大，则原内存的内容保持不变，增加的空间不进行初始化。如果新分配的内存比原内存小，则新分配内存单元不被初始化。realloc 函数返回指向新分配空间的指针；若无法满足要求，则返回 NULL，在这种情况下，原指针 p 指向的单元内容保持不变。

void free(void *p)

free 函数释放 p 指向的内存空间。当 p 的值为 NULL 时，该函数不执行任何操作。p 必须指向先前使用动态分配函数 malloc、realloc 或 calloc 分配的空间。

void abort(void)

abort 函数使程序非正常终止。其功能与 raise（SIGABRT）类似。

void exit(int status) 返回目录

exit 函数使程序正常终止。atexit 函数的调用顺序与登记的顺序相反，这种情况下，所有已打开的文件缓冲区将被清洗，所有已打开的流将被关闭，控制也将返回给环境。status 的值如何返回各环境要视具体的实现而定，但 0 值表示终止成功。也可能是用值 EXIT_SUCCESS 和 EXIT_FAILURE 作为返回值。atexit 函数登记函数 fcn，该函数将在程序正常终止时被调用。若果登记失败，则返回非 0 值。

int atexit(void (*fcn)(void))

atexit 函数登记函数 fcn，该函数将在程序正常终止时被调用。如果登记失败，则返回非 0 值。

int system(const char *s)

system 函数将字符串 s 传递给执行环境。如果 s 的值为 NULL，并且有命令处理程序，则这个函数返回非 0 值。如果 s 的值不是 NULL，则返回值与具体的实现有关。

```
char *getenv(const char *name)
```
getenv 函数返回与 name 有关的环境字符串。如果该字符串不存在，则返回 NULL。其细节与具体的实现有关。

```
void bserch(const void key, const void base, size_t n,
            size_t size, int (*cmp)(const void *, const void *))
```
bsearch 函数在 base[0]…base[n-1]之间查找与*key 匹配的项。在函数 cmp 中，如果第一个参数（查找关键字）小于第二个参数（表项），它必须返回一个负值；如果第一个参数等于第二个参数，它必须返回 0；如果第一个参数大于第二个参数，它必须返回一个正值。数组 base 中的项必须按升序排列。bsearch 函数返回一个指针，它指向一个匹配项，如果不存在匹配项，则返回 NULL。

```
void qsort(void base, size_t n, size_t size, int (*cmp)(const void , const void *))
```
qsort 函数对 base[0]…base[n-1]数组中的对象进行升序排列，数组中每一个对象的长度为 size。比较函数 cmp 与 bsearch 函数中的描述相同。

```
int abs(int n)
```
返回目录

abs 函数返回 int 类型参数 n 的绝对值。

```
long labs(long n)
```
labs 函数返回 long 类型参数 n 的绝对值。

```
div_t div(int num, int denom)
```
div 函数计算 num/denom 的商和余数，并把结果分别保存在结构类型 div_t 的两个 int 类型的成员 quot 和 rem 中。

```
ldiv_t ldiv(long num, long denom)
```
返回目录

ldiv 函数计算 num/denom 的商和余数，并把结果分别保存在结构类型 ldiv_t 的两个 long 类型的成员 quot 和 rem 中。

B. 6 诊断<assert.h>

assert 函数宏用于为程序增加诊断功能。其形式如下：

```
void assert（int 表达式）
```

如果执行语句

```
assert（表达式）
```

时，表达式的值为 0，则 assert 宏将在 stderr 中打印一条消息，比如：

```
Assertion failed：表达式：表达式，file 源文件名，line 行号
```

打印消息后，该宏将调用 abort 终止程序的执行。其中的源文件名和行号来自预处理器宏__FILE__及__LINE__。

如果定义了宏 NDEBUG，同时又包含了头文件<assert.h>，则 assert 宏将被忽略。

B. 7 可变参数表:<stdarg.h>

返回目录

头文件<stdarg.h>提供了遍历未知数目和类型的函数参数表的功能。

假定函数 f 带有可变数目的实际参数，lastarg 是它的最后一个命名形式参数。那么，在函数 f 内声明一个类型为 va_list 的变量 ap，它将依次指向每一个实际参数；

```
va_list ap;
```

在访问任何未命名的参数前，必须用 va_start 宏初始化 ap 一次。

```
va_start (va_list ap, lastarg) ;
```

此后，每次执行宏 va_arg 都将产生一个与下一个未命名的参数具有相同的数值的值，它同时还修改 ap，以使下一次执行 va_arg 时返回下一个参数：

```
类型 va_arg (va-list ap) ;
```

在所有的参数处理完毕后，且在退出函数 f 之前，必须调用宏 va_end 一次，如下所示：

```
void va_end(va_list ap);
```

B. 8 非局部跳转：<setjmp.h> 返回目录

头文件<setjmp.h>中的声明提供了一种不同于通常的函数调用和返回顺序的方式，特别是，它允许立即从一个深层嵌套的函数调用中返回。

```
int setjmp(jmp_buf env)
```

setjmp 宏将状态信息保存到 env 中，供 longjmp 使用。如果直接调用 setjmp，则返回值为 0；如果是在 longjmp 中调用 setjmp，则返回值为非 0。setjmp 只是用于某些上下文中，如用于 if 语句，switch 语句、循环语句的条件测试中及一些简单的关系表达式中。例如：

```
if (setjmp(env) == 0)
    /* 直接调用 setjmp 时，转移到这里 */
else
    /* 调用 longjmp 时，转移到这里 */
```

```
void longjmp(jmp_buf env, int val)
```

longjmp 通过最近一次调用 setjmp 时保存到 env 中的信息恢复状态，同时，程序重新恢复执行，其状态等同于 setjmp 宏调用刚刚执行完并返回非 0 值 val。包含 setjmp 宏调用的函数的执行必须还没有终止。除下列情况外，可访问对象的值同调用 longjmp 时的值相同；在调用 setjmp 宏后，如果调用 setjmp 宏的函数中的非 volatile 自动变量改变了，则它们将变成未定义状态。

B. 9 信号：<signal.h> 返回目录

头文件<signal.h>提供了一些处理程序运行期间引发的各种异常条件的功能，比如来源于外部的中断信号或程序执行错误引起的中断信号。

```
void (*signal(int sig, void (*handler)(int)))(int)
```

signal 决定了如何处理后续的信号。如果 handler 的值是 SIG_DFL，则采用由实现定义的默认行为；如果 handler 的值是 SIG_IGN，则忽略该信号；否则，调用 handler 指向的函数（以信号作为参数）。有效的信号包括：

```
SIGABRT         异常终止，例如由 abort 引起的终止
SIGFPE          算数运算出错，如除数为 0 或溢出
SIGILL          非法函数映像，如非法指令
SIGINT          用于交互式目的信号，如中断
SIGSEGV         非法存储器访问，如访问不存在的内存单元
SIGTERM         发送给程序的终止请求
```

对于特定的信号，signal 将返回 handler 的前一个值；如果出现错误，则返回值 SIG_ERR。

当随后碰到信号 sig 时，该函数将恢复为默认行为，随后调用信号处理程序，就好像由(*handler)(sig)调用的一样。信号处理程序返回后，程序将从信号发生的位置重新开始执行。

信号的初始状态由具体的实现定义。

```
int raise(int sig)
```

raise 向程序发送信号 sig。如果发送不成功，则返回一个非 0 值。

B. 10 日期与时间函数：<time.h> 返回目录

头文件<time.h>中声明了一些处理日期与时间的类型和函数。其中的一些函数用于处理当地时间，因为时区等原因，当地时间与日历时间可能不同。clock_t 和 time_t 是两个表示时间的算术类型，struct tm 用于保存日历时间的各个构成部分。结构 tm 中各个成员的用途及取值范围如下所示：

```
int tm_sec;             从当前分钟开始经过的秒数（0，61）
int tm_min;             从当前小时开始经过的分钟数（0，59）
int tm_hour;            从午夜开始经过的小时数（0，22）
int tm_mday;            当月的天数（1，31）
int tm_mon;             从 1 月起经过的月数（0，11）
int tm_year;            从 1990 年起经过的年数
```

```
    int tm_wday;            从星期天起经过的天数（0，6）
    int tm_yday;            从 1 月 1 日起经过的天数（0，365）
    int tm_isdst;           夏令时标记
```

使用夏令时时，tm_isdst 的值为正，否则为 0。如果该信息无效，则其值为负。

clock_t clock(void)

clock 函数返回程序开始执行后占用的处理器时间。如果无法获得处理器时间，则返回值为-1。clock()/CLOCKS_PER_SEC 是以秒为单位表示的时间。

time_t time(time_t *tp) 返回目录

time 函数返回当前日历时间。如果无法获取日历时间，则返回值为-1。如果 tp 不是 NULL，则同时将返回值赋给*tp。

double difftime(time_t time2, time_t time1)

difftime 函数返回 time2-time1 的值（以秒为单位）。

time_t mktime(struct tm *tp)

mktime 函数将结构*tp 中的当地时间转换为与 time 表示方式相同的日历时间。结构中各成员的值位于上面所示范围内。mktime 函数返回转换后得到的日历时间；如果该时间不能表示，返回-1。

下面 4 个函数返回指向可别其他函数调用覆盖的静态对象的指针。

char *asctime(const struct tm *tp)

asctime 函数将结构*tp 中的时间转换为下列所示的字符串形式：

```
Sun Jan  3 15:14:13 1988\n\0
```

char *ctime(const time_t *tp) 返回目录

ctime 函数将结构*tp 中的日历时间转换为当地时间。它等价于下列函数调用：

```
asctime (localtime (tp) )
```

struct tm *gmtime(const time_t *tp)

gmtime 函数将*tp 中的日历时间转换为协调世界时(UTC)。如果无法获得 UTC，则该函数返回 NULL。函数名字 gmtime 有一定的历史意义。

struct tm *localtime(const time_t *tp)

localtime 函数将结构*tp 中的日历时间转换为当地时间。

size_t strftime(char *s, size_t smax, const char *fmt, const struct tm *tp)

strftime 函数根据 fmt 中的格式把结构*tp 中的日期与时间信息转换为指定的格式，并存储到 s 中，其中 fmt 类似于 printf 函数中的格式说明。普通字符（包括终结符’\0’）将复制到 s 中。每个%c 将按照下面描述的格式替换为与本地环境相适应的值。最多 smax 个字符写到 s 中。strftime 函数返回实际写到 s 中的字符数（不包含字符’\0’）；如果字符数多于 smax，该函数将返回值 0。

fmt 的转换说明及其含义如下所示：

```
%a          一星期中各天的缩写名
%A          一星期中各天的全名
%b          缩写的月份名
%B          月份的全名
%c          当地时间和日期表示
%d          一个月中的某一天（0 - 31）
%H          小时（24 小时表示）（00 - 23）
%I          小时（12 小时表示）（00 - 12）
%j          一年中的各天（001 - 366）
%m          月份（00 - 12）
```

%M	分钟（00 - 59）
%p	与 AM 与 PM 相对应的当地时间等价表示方法
%S	秒（00 - 61）
%U	一年中的星期序号（00 - 53，将星期日看作是每周的第一天）
%w	一周中的各天（0 - 6，星期日为 0）
%W	一年中的星期序号（00 - 53， 将星期一看作是每周的第一天）
%x	当地日期表示
%X	当地时间表示
%y	不带世纪数目的年份（00 - 99）
%y	带世纪数目的年份
%Z	时区名（如果有的话）
%%	%本身

B. 11 与具体实现相关的限制：<limits.h>和<folat.h>

返回目录

头文件<limits.h>定义了一些表示整型大小的常量。以下所列的值是 **NI LabWindows/CVI6.0** 系统中使用的值。

```
#define CHAR_BIT    8                                  char 类型的位数

#define UCHAR_MAX   0xff                          unsigned char 类型的最大值
#define USHRT_MAX   0xffff                       unsigned short 类型的最大值
#define UINT_MAX    0xffffffff                     unsigned int 类型的最大值
#define ULONG_MAX   0xffffffffL                   unsigned long 类型的最大值

#define CHAR_MAX    SCHAR_MAX                           char 类型的最大值
#define SCHAR_MAX   0x7f                         signed char 类型的最大值
#define SHRT_MAX    0x7fff                       signed short 类型的最大值
#define INT_MAX     0x7fffffff                     signed int 类型的最大值
#define LONG_MAX    0x7fffffffL                   singed long 类型的最大值

#define CHAR_MIN    SCHAR_MIN                           char 类型最小值
#define SCHAR_MIN   (-SCHAR_MAX-1)                singed char 类型最小值
#define SHRT_MIN    (-SHRT_MAX-1)                signed short 类型最小值
#define INT_MIN     (-INT_MAX-1)                        int 类型最小值
#define LONG_MIN    (-LONG_MAX-1)                      long 类型最小值
```

下表列出的名字是<float.h>的一个子集，它们是与浮点算数运算相关的一些常量。以下所列的值是 **NI LabWindows/CVI6.0** 系统中使用的值。

```
#define FLT_ROUNDS          1                          加法的浮点舍入模式
#define FLT_RADIX           2                       指数表示的基数,例如 2、16

#define FLT_DIG             6                             表示精度的十进制数字
#define FLT_EPSILON         _FloatEps.val          最小的数 x，x 满足 1.0+x ≠1.0
#define FLT_MANT_DIG        24                  尾数中的数（以 FLT_RADIX 为基数）
#define FLT_MAX             _FloatMax.val                       最大的浮点数
#define FLT_MAX_EXP         128 最大的数 n，n 满足：FLT_RADIX’ n-1 仍是可表示的
#define FLT_MIN             _FloatMin.val                   最小的规格化浮点数
#define FLT_MIN_EXP         (-125)最小的数 n，n 满足：10‘n 是一个规格化数

#define DBL_DIG             15                            表示精度的十进制数
#define DBL_EPSILON         _DoubleEps.val         最小的数 x，x 满足 1.0+x ≠1.0
```

```
#define DBL_MANT_DIG            53
#define DBL_MAX                 _DoubleMax.val
#define DBL_MAX_10_EXP          308
#define DBL_MAX_EXP             1024
#define DBL_MIN                 _DoubleMin.val
#define DBL_MIN_10_EXP          (-307)
#define DBL_MIN_EXP             (-1021)

#define LDBL_MANT_DIG           DBL_MANT_DIG
#define LDBL_EPSILON            DBL_EPSILON
#define LDBL_DIG                DBL_DIG
#define LDBL_MIN_EXP            DBL_MIN_EXP
#define LDBL_MIN                DBL_MIN
#define LDBL_MIN_10_EXP         DBL_MIN_10_EXP
#define LDBL_MAX_EXP            DBL_MAX_EXP
#define LDBL_MAX                DBL_MAX
#define LDBL_MAX_10_EXP         DBL_MAX_10_EXP
```

结束

返回目录