C 标准库

本文包括大部分 C 标准库函数，但没有列出一些用途有限的函数以及某些可以简单的从其他函数合成的函数，也没有包含多字节和本地化函数。

标准库中的各个函数、类型以及宏分别在以下标准头文件中说明：

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

1 输入与输出<stdio.h>

头文件<stdio.h>定义了用于输入和输出的函数、类型和宏。最重要的类型是用于声明文件指针的 FILE。另外两个常用的类型是 size_t 和 fpos_t，size_t 是由运算符 sizeof 产生的无符号整类型；fpos_t 类型定义能够唯一说明文件中的每个位置的对象。由头部定义的最有用的宏是 EOF，其值代表文件的结尾。

1.1 文件操作

1.1.1 fopen

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
返回：成功为FILE指针，失败为NULL
```

打开以filename所指内容为名字的文件，返回与之关联的流。

mode决定打开的方式，可选值如下：

"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"	添加；打开或创建文本文件用于在文件末尾写
"rb"	打开二进制文件用于读
"wb"	创建二进制文件用于写，并删除已存在的内容(如果有的话)
"ab"	添加；打开或创建二进制文件用于在文件末尾写
"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+"	添加；打开或创建文本文件用于更新和在文件末尾写
"rb+"或"r+b"	打开二进制文件用于更新(即读和写)
"wb+"或"w+b"	创建二进制文件用于更新，并删除已存在的内容(如果有的话)
"ab+"或"a+b"	添加；打开或创建二进制文件用于更新和在文件末尾写

后六种方式允许对同一文件进行读和写，要注意的是，在写操作和读操作的交替过程中，必须调用fflush()或文件定位函数如fseek()、fsetpos()、rewind()等。

文件名filename的长度最大为FILENAME_MAX个字符，一次最多可打开FOPEN_MAX个文件(在<stdio.h>中定义)。

1.1.2 freopen

```
#include <stdio.h>
FILE *freopen(const char *filename, const char *mode,
FILE *stream);
返回：成功为stream，失败为NULL
```

以 mode 指定的方式打开文件 filename，并使该文件与流 stream 相关联。freopen()先尝试关闭与 stream 关联的文件，不管成功与否，都继续打开新文件。

该函数的主要用途是把系统定义的标准流 stdin、stdout、stderr 重定向到其他文件。

1.1.3 fflush

```
#include <stdio.h>
int fflush(FILE *stream);
返回：成功为0，失败返回EOF
```

对输出流(写打开)，fflush()用于将已写到缓冲区但尚未写出的全部数据都写到文件中；对输入流，其结果未定义。如果写过程中发生错误则返回 EOF，正常则返回 0。

fflush(NULL)用于刷新所有的输出流。

程序正常结束或缓冲区满时，缓冲区自动清仓。

1.1.4 fclose

```
#include <stdio.h>
int flcose(FILE *stream);
返回：成功为0，失败返回EOF
```

刷新 stream 的全部未写出数据，丢弃任何未读的缓冲区内的输入数据并释放自动分配的缓冲区，最后关闭流。

1.1.5 remove

```
#include <stdio.h>
int remove(const char *filename);
返回：成功为0，失败为非0值
```

删除文件filename。

1.1.6 rename

```
#include <stdio.h>
int rename(const char *oldfname, const char *newfname);
返回：成功为0，失败为非0值
```

把文件的名字从oldfname改为newfname。

1.1.7 tmpfile

```
#include <stdio.h>
FILE *tmpfile(void);
返回：成功为流指针，失败为NULL
```

以方式"wb+"创建一个临时文件，并返回该流的指针，该文件在被关闭或程序正常结束时被自动删除。

1.1.8 tmpnam

```
#include <stdio.h>
char *tmpnam(char s[L_tmpnam]);
```

返回：成功为非空指针，失败为NULL

若参数s为NULL(即调用tmpnam(NULL))，函数创建一个不同于现存文件名字的字符串，并返回一个指向一内部静态数组的指针。

若s非空，则函数将所创建的字符串存储在数组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。在程序的执行过程中，最多只能确保生成TMP_MAX个不同的名字。注意tmpnam函数只是用于创建一个名字，而不是创建一个文件。

1.1.9 setvbuf

```
#include <stdio.h>
int setvbuf(FILE *stream, char *buf, int mode, size_t size);
返回：成功返回0，失败返回非0
```

控制流stream的缓冲区，这要在读、写以及其他任何操作之前设置。

如果buf非空，则将buf指向的区域作为流的缓冲区，如果buf为NULL，函数将自行分配一个缓冲区。

size决定缓冲区的大小。

mode指定缓冲的处理方式，有如下值：

- _IOFBF，进行完全缓冲；
- _IOLBF，对文本文件表示行缓冲；
- _IOLNF，不设置缓冲。

1.1.10 setbuf

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

如果 buf 为 NULL，则关闭流 stream 的的缓冲区；否则 setbuf 函数等价于：

(void)setvbuf(stream, buf, _IOFBF, BUFSIZ)。

注意自定义缓冲区的尺寸必须为 BUFSIZ 个字节。

1.2 格式化输出

1.2.1 fprintf

```
#include <stdio.h>
int fprintf(FILE *stream, const char *format,…);
返回：成功为实际写出的字符数，出错返回负值
```

按照 format 说明的格式把变元表中变元内容进行转换，并写入 stream 指向的流。

格式化字符串由两种类型的对象组成：普通字符(它们被拷贝到输出流)与转换规格说明(它们决定变元的转换和输出格式)。每个转换规格说明均以字符%开头，以转换字符结束。如果%后面的字符不是转换字符，那么该行为是未定义的。

转换字符列表如下：

字符	说明
d, i	int；有符号十进制表示法
o	unsigned int；无符号八进制表示法（无前导 0）
x, X	unsigned int；无符号十六进制表示法（无前导 0X 和 0x），对 0x 用 abcdef，对 0X 用 ABCDEF
u	unsigned int；无符号十进制表示法
c	int；单个字符，转换为 unsigned char 类型后输出
s	char *；输出字符串直到'\0'或者达到精度指定的字符数
f	double；形如[-]mmm.ddd 的十进制浮点数表示法，d 的数目由精度确定。缺省精度为 6 位，精

	度为 0 时不输出小数点
e, E	double；形如[-]m.dddddde[+-]xx 或者[-]m.ddddddE[+-]xx 的十进制浮点数表示法，d 的数目由精度确定。缺省精度为 6 位，精度为 0 时不输出小数点
g G	double；当指数值小于-4 或大于等于精度时，采用%e 或%E 的格式；否则使用%f 的格式。尾部的 0 与小数点不打印
p	void *；输出指针值（具体表示与实现相关）
n	int *；到目前为止以此格式调用函数输出的字符的数目将被写入到相应变元中，不进行变元转换
%	不进行变元转换，输出%

在%与转换字符之间依次可以有下列标记：

标记	说明
-	指定被转换的变元在其字段内左对齐
+	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，那么在其前面附加一个空格
0	对于数值转换，在输出长度小于字段宽度时，加前导 0
#	指定其他输出格式，对于 o 格式，第一个数字必须为零；对于 x/X 格式，指定在输出的非 0 值前加 0x 或 0X；对于 e/E/f/g/G 格式，指定输出总有一个小数点；对于 g/GG 格式，还指定输出值后面无意义的 0 将被保留。
宽度[number]	一个指定最小字段宽的数。转换后的变元输出宽度至少要达到这个数值。如果变元的字符数小于此数值，那么在变元左/右边添加填充字符。填充字符通常为空格（设置了 0 标记则为 0）。
.	点号用于把字段宽和精度分开
精度[number]	对于字符串，说明输出字符的最大数目；对于 e/E/f 格式，说明输出的小数位数；对于 g/G 格式，说明输出的有效位数；对于整数，说明输出的最小位数（必要时可加前导 0）
h/l/L	长度修饰符，h 表示对应的变元按 short 或 unsigned short 类型输出；l 表示对应的变元按 long 或 unsigned long 类型输出；L 表示对应的变元按 long double 类型输出

在格式串中字段宽度和精度二者都可以用*来指定，此时该值可通过转换对应的变元来获得，这些变元必须是 int 类型。

1.2.2 printf

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

printf(...)等价于 fprintf(stdout, ...)。

1.2.3 sprintf

```
#include <stdio.h>
int sprintf(char *buf, const char *format, …);
返回：实际写到字符数组的字符数，不包括'\0'
```

与printf()基本相同，但输出写到字符数组buf而不是stdout中，并以'\0'结束。

注意，sprintf()不对buf进行边界检查，buf必须足够大，以便能装下输出结果。

1.2.4 snprintf

```
#include <stdio.h>
int snprintf(char *buf, size_t num, const char *format, …);
```

除了最多为num-1个字符被存放到buf指向的数组之外，snprintf()和sprintf()完全相同。数组以'\0'结束。

该函数不属于C89(C99增加的)，但应用广泛，所以将其包括了进来。

1.2.5 vprintf

1.2.6 vfprintf

1.2.7 vsprintf

1.2.8 vsnprintf

```
#include <stdarg.h>
```

```
#include <stdio.h>
int vprintf(char *format, va_list arg);
int vfprintf(FILE *stream, const char *format, va_list arg);
int vsprintf(char *buf, const char *format, va_list arg);
int vsnprintf(char *buf, size_t num, const char *format,
va_list arg);
```

这几个函数与对应的 printf()等价，但变元表由 arg 代替。参见第 7 节有关<stdarg.h>头文件的讨论。

vsnprintf 是 C99 增加的。

1.3 格式化输入

1.3.1 fscanf

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format, …);
返回：成功为实际被转换并赋值的输入项数目，
到达文件尾或变元被转换前出错为 EOF
```

在格式串 format 的控制下从流 stream 中读入字符，把转换后的值赋给后续各个变元，在此每一个变元都必须是一个指针。当格式串 format 结束时函数返回。

格式串 format 通常包含有用于指导输入转换的转换规格说明。格式串中可以包含：

- 空格或制表符，他们将被忽略；
- 普通字符(%除外)，与输入流中下一个非空白字符相匹配；
- 转换规格说明：由一个%、一个赋值屏蔽字符*(可选)、一个用于指定最大字段宽度的数(可选)、一个用于指定目标字段的字符 h/l/L(可选)、一个转换字符组成。

转换规格说明决定了输入字段的转换方式。通常把结果保存在由对应变元指向的变量中。然而，如果转换规格说明中包含有赋值屏蔽字符*，例如%*s，那么就跳过对应的输入字段，不进行赋值。输入字段是一个由非空白符组成的字符串，当遇到空白符或到达最大字段宽(如果有的话)时，对输入字段的读入结束。这意味着 scanf 函数可以跨越行的界限来读入其输入，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)。

转换字符列表如下：

字符	输入数据；变元类型
d	十进制整数；int *
i	整数；int *。该整数可以是以 0 开头的八进制数，也可以是以 0x/0X 开头的十六进制数
o	八进制数（可以带或不带前导 0）；unsigned int *
u	无符号十进制整数；unsigned int *
x	十六进制整数（可以带或不带前导 0x/0X)；unsigned int *
c	字符；char *。按照字段宽的大小把读入的字符保存在指定的数组中，不加入字符'\0'。字段宽的缺省值为 1。在这种情况下，不跳过空白符；如果要读入下一个非空白符，使用%1s（数字 1）
s	有非空白符组成的字符串（不包含引号）；char *。该变元指针指向一个字符数组，该字符数组有足够空间来保存该字符串以及在末尾添加的'\0'
e/f/g	浮点数；float *。float 浮点数的输入格式为：一个任选的正负号，一串可能包含小数点的数字和一个任选的指数字段。指数字段由字母 e/E 以及后跟的一个可能带正负号的整数组成
p	用 printf("%p")调用输出的指针值；void *
n	将到目前为止此调用所读的字符数写入变元；int *。不读入输入字符。不增加转换项目计数
[...]	用方括号括起来的字符集中的字符来匹配输入，以找到最长的非空字符串；char *。在末尾添加'\0'。格式[]...]表示字符集中包含字符]
[^...]	用不在方括号里的字符集中的字符来匹配输入，以找到最长的非空字符串；char *。在末尾添加'\0'。格式[]...]表示字符集中包含字符]
%	字面值%，不进行赋值

字段类型字符：

- 如果变元是指向 short 类型而不是 int 类型的指针，那么在 d/i/n/o/u/x 这几个转换字符前可以加上字符 h；
- 如果变元是指向 long 类型的指针，那么在 d/i/n/o/u/x 这几个转换字符前可以加上字符 l；
- 如果变元是指向 double 类型而不是 float 类型的指针，那么在 e/f/g 这几个转换字符前可以加上字符 l；
- 如果变元是指向 long double 类型的指针，那么在 e/f/g 前可以加上字符 L。

1.3.2 scanf

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

scanf(...)等价于 fscanf(stdin, ...)。

1.3.3 sscanf

```
#include <stdio.h>
int sscanf(const char *buf, const char *format, …);
```

与 scanf()基本相同，但 sscanf()从 buf 指向的数组中读，而不是 stdin。

1.4 字符输入输出函数

1.4.1 fgetc

```
#include <stdio.h>
int fgetc(FILE *stream);
```

以 unsigned char 类型返回输入流 stream 中下一个字符(转换为 int 类型)。如果到达文件末尾或发生错误，则返回 EOF。

1.4.2 fgets

```
#include <stdio.h>
char *fgets(char *str, int num, FILE *stream);
返回：成功返回 str，到达文件尾或发生错误返回 NULL
```

从流 stream 中读入最多 num-1 个字符到数组 str 中。当遇到换行符时，把换行符保留在 str 中，读入不再进行。数组 str 以'\0'结尾。

1.4.3 fputc

```
#include <stdio.h>
int fputc(int ch, FILE *stream);
返回：成功为所写的字符，出错为 EOF
```

把字符 ch(转换为 unsigned char 类型)输出到流 stream 中。

1.4.4 fputs

```
#include <stdio.h>
int fputs(const char *str, FILE *stream);
返回：成功返回非负值，失败返回 EOF
```

把字符串 str 输出到流 stream 中，不输出终止符'\0'。

1.4.5 getc

```
#include <stdio.h>
int getc(FILE *stream);
```

getc()与fgetc()等价。不同之处为：当getc函数被定义为宏时，它可能多次计算stream的值。

1.4.6 getchar

```
#include <stdio.h>
int getchar(void);
```

等价于getc(stdin)。

1.4.7 gets

```
#include <stdio.h>
char *gets(char *str);
返回：成功为str，到达文件尾或发生错误则为NULL
```

从stdin中读入下一个字符串到数组str中，并把读入的换行符替换为字符'\0'。

gets()可读入无限多字节，所以要保证str有足够的空间，防止溢出。

1.4.8 putc

```
#include <stdio.h>
int putc(int ch, FILE *stream);
```

putc()与fputc()等价。不同之处为：当putc函数被定义为宏时，它可能多次计算stream的值。

1.4.9 putchar

```
#include <stdio.h>
int putchar(int ch);
```

等价于 putc(ch, stdout)。

1.4.10 puts

```
#include <stdio.h>
int puts(const char *str);
返回：成功返回非负值，出错返回 EOF
```

把字符串 str 和一个换行符输出到 stdout。

1.4.11 ungetc

```
#include <stdio.h>
int ungetc(int ch, FILE *stream);
返回：成功时为 ch，出错为 EOF
```

把字符 ch(转换为 unsigned char 类型)写回到流 stream 中，下次对该流进行读操作时，将返回该字符。对每个流只保证能写回一个字符(有些实现支持回退多个字符)，且此字符不能是 EOF。

1.5 直接输入输出函数

1.5.1 fread

```
#include <stdio.h>
size_t fread(void *buf, size_t size, size_t count,
FILE *stream);
返回：实际读入的对象数
```

从流 stream 中读入最多 count 个长度为 size 个字节的对象，放到 buf 指向的数组中。

返回值可能小于指定读入数，原因可能是出错，也可能是到达文件尾。实际执行状态可用 feof()或 ferror()确定。

1.5.2 fwrite

```
#include <stdio.h>
size_t fwrite(const void *buf, size_t size, size_t count,
FILE *stream);
返回：实际输出的对象数
```

把 buf 指向的数组中 count 个长度为 size 的对象输出到流 stream 中，并返回被输出的对象数。如果发生错误，则返回一个小于 count 的值。

1.6 文件定位函数

1.6.1 fseek

```
#include <stdio.h>
int fseek(FILE *stream, long int offset, int origin);
返回：成功为0，出错为非0
```

对流 stream 相关的文件定位，随后的读写操作将从新位置开始。

对于二进制文件，此位置被定位在由 origin 开始的 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(文件结束处)。

对于文本流，offset 心须为 0，或者是由函数 ftell()返回的值(此时 origin 的值必须是 SEEK_SET)。

1.6.2 ftell

```
#include <stdio.h>
long int ftell(FILE *stream);
```

返回与流 stream 相关的文件的当前位置。出错时返回-1L。

1.6.3 rewind

```
#include <stdio.h>
void rewind(FILE *stream);
```

rewind(fp)等价于 fssek(fp, 0L, SEEK_SET)与 clearerr(fp)这两个函数顺序执行的效果，即把与流 stream 相关的文件的当前位置移到开始处，同时清除与该流相关的文件尾标志和错误标志。

1.6.4 fgetpos

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

```
返回：成功返回0，失败返回非0
```

把流stream的当前位置记录在*position中，供随后的fsetpos()调用时使用。

1.6.5 fsetpos

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *position);
返回：成功返回0，失败返回非0
```

把流stream的位置定位到*position中记录的位置。*position的值是之前调用fgetpos()记录下来的。

1.7 错误处理函数

当发生错误或到达文件末尾时，标准库中的许多函数将设置状态指示符。这些状态指示符可被显式地设置和测试。另外，(定义在<errno.h>中的)整数表达式errno可包含一个出错序号，该数将进一步给出最近一次出错的信息。

1.7.1 clearerr

```
#include <stdio.h>
void clearerr(FILE *stream);
```

清除与流stream相关的文件结束指示符和错误指示符。

1.7.2 feof

```
#include <stdio.h>
int feof(FILE *stream);
返回：到达文件尾时返回非0值，否则返回0
```

与流 stream 相关的文件结束指示符被设置时，函数返回一个非 0 值。

1.7.3 ferror

```
#include <stdio.h>
int ferror(FILE *stream);
返回：无错返回0，有错返回非0
```

与流 stream 相关的文件出错指示符被设置时，函数返回一个非 0 值。

1.7.4 perror

```
#include <stdio.h>
void perror(const char *str);
```

perror(s)用于输出字符串 s 以及与全局变量 errno 中的整数值相对应的出错信息，具体出错信息的内容依赖于实现。该函数的功能类似于：

```
fprintf(stderr, "%s: %s\n", s, "出错信息");
```

可参见第 3 节介绍的 strerror 函数。

2 字符类测试<ctype.h>

头文件<ctype.h>中说明了一些用于测试字符的函数。每个函数的变元均为int类型，变元的值必须是EOF或可用unsigned char类型表示的字符，函数的返回值为int类型。如果变元满足所指定的条件，那么函数返回非0值(表示真)；否则返回值为0(表示假)。这些函数包括2.1～2.11。

在7位ASCII字符集中，可打印字符是从0x20(' ')到0x7E('~')之间的字符；控制字符是从0(NUL)到0x1F(US)之间的字符和字符0x7F(DEL)。

2.1 isalnum

```
#include <ctype.h>
int sialnum(int ch);
```

变元为字母或数字时，函数返回非0值，否则返回0。

2.2 isalpha

```
#include <ctype.h>
int isalpha(int ch);
```

当变元为字母表中的字母时，函数返回非0值，否则返回0。各种语言的字母表互不相同，对于英语来说，字母表由大写和小写的字母A到Z组成。

2.3 iscntrl

```
#include <ctype.h>
int iscntrl(int ch);
```

当变元是控制字符时，函数返回非0，否则返回0。

2.4 isdigit

```
#include <ctype.h>
int isdigit(int ch);
```

当变元是十进制数字时，函数返回非0值，否则返回0。

2.5 isgraph

```
#include <ctype.h>
int isgraph(int ch);
```

如果变元为除空格之外的任何可打印字符，则函数返回非0值，否则返回0。

2.6 islower

```
#include <ctype.h>
int islower(int ch);
```

如果变元是小写字母，函数返回非0值，否则返回0。

2.7 isprint

```
#include <ctype.h>
```

```
int isprint(int ch);
```

如果变元是可打印字符(含空格)，则函数返回非0值，否则返回0。

2.8 ispunct

```
#include <ctype.h>
int ispunct(int ch);
```

如果变元是除空格、字母和数字外的可打印字符，则函数返回非0，否则返回0。

2.9 isspace

```
#include <ctype.h>
int isspace(int ch);
```

当变元为空白字符(包括空格、换页符、换行符、回车符、水平制表符和垂直制表符)时，函数返回非0，否则返回0。

2.10 isupper

```
#include <ctype.h>
int isupper(int ch);
```

如果变元为大写字母，函数返回非0，否则返回0。

2.11 isxdigit

```
#include <ctype.h>
int isxdigit(int ch);
```

当变元为十六进制数字时，函数返回非0，否则返回0。

2.12 tolower

```
#include <string.h>
int tolower(int ch);
```

当ch为大写字母时，返回其对应的小写字母；否则返回ch。

2.13 toupper

```
#include <string.h>
int toupper(int ch);
```

当ch为小写字母时，返回其对应的大写字母；否则返回ch。

3 字符串函数<string.h>

在头文件<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。只有函数memmove对重叠对象间的拷贝进行了定义，而其他函数都未定义。比较类函数将其变元视为unsigned char类型的数组。

3.1 strcpy

```
#include <string.h>
char *strcpy(char *str1, const char *str2);
```

把字符串 str2(包括'\0')拷贝到字符串 str1 当中，并返回 str1。

3.2 strncpy

```
#include <string.h>
char *strncpy(char *str1, const char *str2, size_t count);
```

把字符串 str2 中最多 count 个字符拷贝到字符串 str1 中，并返回 str1。如果 str2 中少于 count 个字符，那么就用'\0'来填充，直到满足 count 个字符为止。

3.3 strcat

```
#include <string.h>
char *strcat(char *str1, const char *str2);
```

把 str2(包括'\0')拷贝到 str1 的尾部(连接)，并返回 str1。其中终止原 str1 的'\0'被 str2 的第一个字符覆盖。

3.4 strncat

```
#include <string.h>
char *strncat(char *str1, const char *str2, size_t count);
```

把 str2 中最多 count 个字符连接到 str1 的尾部，并以'\0'终止 str1，返回 str1。其中终止原 str1 的'\0'被 str2 的第一个字符覆盖。

注意，最大拷贝字符数是 count+1。

3.5 strcmp

```
#include <string.h>
int strcmp(const char *str1, const char *str2);
```

按字典顺序比较两个字符串，返回整数值的意义如下：

- 小于 0，str1 小于 str2；
- 等于 0，str1 等于 str2；
- 大于 0，str1 大于 str2；

3.6 strncmp

```
#include <string.h>
int strncmp(const char *str1, const char *str2, size_t count);
```

同 strcmp，除了最多比较 count 个字符。根据比较结果返回的整数值如下：

- 小于 0，str1 小于 str2；
- 等于 0，str1 等于 str2；
- 大于 0，str1 大于 str2；

3.7 strchr

```
#include <string.h>
char *strchr(const char *str, int ch);
```

返回指向字符串 str 中字符 ch 第一次出现的位置的指针，如果 str 中不包含 ch，则返回 NULL。

3.8 strrchr

```
#include <string.h>
char *strrchr(const char *str, int ch);
```

返回指向字符串 str 中字符 ch 最后一次出现的位置的指针，如果 str 中不包含 ch，则返回 NULL。

3.9 strspn

```
#include <string.h>
size_t strspn(const char *str1, const char *str2);
```

返回字符串 str1 中由字符串 str2 中字符构成的第一个子串的长度。

3.10 strcspn

```
#include <string.h>
size_t strcspn(const char *str1, const char *str2);
```

返回字符串 str1 中由不在字符串 str2 中字符构成的第一个子串的长度。

3.11 strpbrk

```
#include <string.h>
char *strpbrk(const char *str1, const char *str2);
```

返回指向字符串 str2 中的任意字符第一次出现在字符串 str1 中的位置的指针；如果 str1 中没有与 str2 相同的字符，那么返回 NULL。

3.12 strstr

```
#include <string.h>
char *strstr(const char *str1, const char *str2);
```

返回指向字符串 str2 第一次出现在字符串 str1 中的位置的指针；如果 str1 中不包含 str2，则返回 NULL。

3.13 strlen

```
#include <string.h>
size_t strlen(const char *str);
```

返回字符串 str 的长度，'\0'不算在内。

3.14 strerror

```
#include <string.h>
```

```
char *strerror(int errnum);
```

返回指向与错误序号 errnum 对应的错误信息字符串的指针(错误信息的具体内容依赖于实现)。

3.15 strtok

```
#include <string.h>
char *strtok(char *str1, const char *str2);
```

在 str1 中搜索由 str2 中的分界符界定的单词。

对 strtok()的一系列调用将把字符串 str1 分成许多单词，这些单词以 str2 中的字符为分界符。第一次调用时 str1 非空，它搜索 str1，找出由非 str2 中的字符组成的第一个单词，将 str1 中的下一个字符替换为'\0'，并返回指向单词的指针。随后的每次 strtok()调用(参数 str1 用 NULL 代替)，均从前一次结束的位置之后开始，返回下一个由非 str2 中的字符组成的单词。当 str1 中没有这样的单词时返回 NULL。每次调用时字符串 str2 可以不同。

如：

```
char *p;
p = strtok("The summer soldier,the sunshine patriot", " ");
printf("%s", p);
do {
    p = strtok("\0", ", "); /* 此处 str2 是逗号和空格 */
    if (p)
        printf("|%s", p);
} while (p);
```

显示结果是：The | summer | soldier | the | sunshine | patriot

3.16 memcpy

```
#include <string.h>
void *memcpy(void *to, const void *from, size_t count);
```

把 from 中的 count 个字符拷贝到 to 中。并返回 to。

3.17 memmove

```
#include <string.h>
void *memmove(void *to, const void *from, size_t count);
```

功能与 memcpy 类似，不同之处在于，当发生对象重叠时，函数仍能正确执行。

3.18 memcmp

```
#include <string.h>
int memcmp(const void *buf1, const void *buf2, size_t count);
```

比较 buf1 和 buf2 的前 count 个字符，返回值与 strcmp 的返回值相同。

3.19 memchr

```
#include <string.h>
void *memchr(const void *buffer, int ch, size_t count);
```

返回指向 ch 在 buffer 中第一次出现的位置指针，如果在 buffer 的前 count 个字符当中找不到匹配，则返回 NULL。

3.20 memset

```
#include <string.h>
void *memset(void *buf, int ch, size_t count);
```

把 buf 中的前 count 个字符替换为 ch，并返回 buf。

4 数学函数<math.h>

头文件<math.h>中说明了数学函数和宏。

宏 EDOM 和 ERANGE(定义在头文件<errno.h>中)是两个非 0 整常量，用于引发各个数学函数的定义域错误和值域错误；HUGE_VAL 是一个 double 类型的正数。当变元取值在函数的定义域之外时，就会出现定义域错误。在发生定义域错误时，全局变量 errno 的值被置为 EDOM，函数的返回值视具体实现而定。如果函数的结果不能用 double 类型表示，那么就会发生值域错误。当结果上溢时，函数返回 HUGE_VAL 并带有正确的符号(正负号)，errno 的值被置为 ERANGE。当结果下溢时，函数返回 0，而 errno 是否被设置为 ERANGE 视具体实现而定。

4.1 sin

```
#include <math.h>
double sin(double arg);
```

返回 arg 的正弦值，arg 单位为弧度。

4.2 cos

```
#include <math.h>
double cos(double arg);
```

返回 arg 的余弦值，arg 单位为弧度。

4.3 tan

```
#include <math.h>
double tan(double arg);
```

返回 arg 的正切值，arg 单位为弧度。

4.4 asin

```
#include <math.h>
double asin(double arg);
```

返回 arg 的反正弦值 $\sin^{-1}(x)$，值域为[-pi/2,pi/2]，其中变元范围[-1,1]。

4.5 acos

```
#include <math.h>
```

```
double acos(double arg);
```

返回 arg 的反余弦值 $\cos^{-1}(x)$，值域为[0,pi]，其中变元范围[-1,1]。

4.6 atan

```
#include <math.h>
double atan(double arg);
```

返回 arg 的反正切值 $\tan^{-1}(x)$，值域为[-pi/2,pi/2]。

4.7 atan2

```
#include <math.h>
double atan2(double a, double b);
```

返回 a/b 的反正切值 $\tan^{-1}(a/b)$，值域为[-pi,pi]。

4.8 sinh

```
#include <math.h>
double sinh(double arg);
```

返回 arg 的双曲正弦值。

4.9 cosh

```
#include <math.h>
double cosh(double arg);
```

返回 arg 的双曲余弦值。

4.10 tanh

```
#include <math.h>
double tanh(double arg);
```

返回 arg 的双曲正切值。

4.11 exp

```
#include <math.h>
double exp(double arg);
```

返回幂函数 e^x。

4.12 log

```
#include <math.h>
double log(double arg);
```

返回自然对数 ln(x)，其中变元范围 arg > 0。

4.13 log10

```
#include <math.h>
double log10(double arg);
```

返回以 10 为底的对数 $\log_{10}(x)$，其中变元范围 arg > 0。

4.14 pow

```
#include <math.h>
double pow(double x, double y);
```

返回 x^y，如果 x=0 且 y<=0 或者如果 x<0 且 y 不是整数，那么产生定义域错误。

4.15 sqrt

```
#include <math.h>
double sqrt(double arg);
```

返回 arg 的平方根，其中变元范围 arg>=0。

4.16 ceil

```
#include <math.h>
double ceil(double arg);
```

返回不小于 arg 的最小整数。

4.17 floor

```
#include <math.h>
double floor(double arg);
```

返回不大于 arg 的最大整数。

4.18 fabs

```
#include <math.h>
double fabs(double arg);
```

返回 arg 的绝对值|x|。

4.19 ldexp

```
#include <math.h>
double ldexp(double num, int exp);
```

返回 num * 2^{exp}。

4.20 frexp

```
#include <math.h>
double frexp(double num, int *exp);
```

把num分成一个在[1/2,1)区间的真分数和一个2的幂数。将真分数返回，幂数保存在*exp中。如果num等于0，那么这两部分均为0。

4.21 modf

```
#include <math.h>
double modf(double num, double *i);
```

把num分成整数和小数两部分，两部分均与num有同样的正负号。函数返回小数部分，整数部分保存在*i中。

4.22 fmod

```
#include <math.h>
double fmod(double a, double b);
```

返回a/b的浮点余数，符号与a相同。如果b为0，那么结果由具体实现而定。

5 实用函数<stdlib.h>

在头文件<stdlib.h>中说明了用于数值转换、内存分配以及具有其他相似任务的函数。

5.1 atof

```
#include <stdlib.h>
double atof(const char *str);
```

把字符串 str 转换成 double 类型。等价于：strtod(str, (char**)NULL)。

5.2 atoi

```
#include <stdlib.h>
int atoi(const char *str);
```

把字符串 str 转换成 int 类型。等价于：(int)strtol(str, (char**)NULL, 10)。

5.3 atol

```
#include <stdlib.h>
long atol(const char *str);
```

把字符串 str 转换成 long 类型。等价于：strtol(str, (char**)NULL, 10)。

5.4 strtod

```
#include <stdlib.h>
double strtod(const char *start, char **end);
```

把字符串 start 的前缀转换成 double 类型。在转换中跳过 start 的前导空白符，然后逐个读入构成数的字符，任何非浮点数成分的字符都会终止上述过程。如果 end 不为 NULL，则把未转换部分的指针保存在*end 中。

如果结果上溢，返回带有适当符号的 HUGE_VAL，如果结果下溢，那么函数返回 0。在这两种情况下，errno 均被置为 ERANGE。

5.5 strtol

```
#include <stdlib.h>
long int strtol(const char *start, char **end, int radix);
```

把字符串 start 的前缀转换成 long 类型，在转换中跳过 start 的前导空白符。如果 end 不为 NULL，则把未转换部分的指针保存在*end 中。

如果 radix 的值在 2 到 36 间之间，那么转换按该基数进行；如果 radix 为 0，则基数为八进制、十进制、十六进制，以 0 为前导的是八进制，以 0x 或 0X 为前导的是十六进制。无论在哪种情况下，串中的字母是表示 10 到 radix-1 之间数字的字母。如果 radix 是 16，可以加上前导 0x 或 0X。

如果结果上溢，则依据结果的符号返回 LONG_MAX 或 LONG_MIN，置 errno 为 ERANGE。

5.6 strtoul

```
#include <stdlib.h>
unsigned long int strtoul(const char *start, char **end,
int radix);
```

与 strtol()类似，只是结果为 unsigned long 类型，溢出时值为 ULONG_MAX。

5.7 rand

```
#include <stdlib.h>
int rand(void);
```

产生一个 0 到 RAND_MAX 之间的伪随机整数。RAND_MAX 值至少为 32767。

5.8 srand

```
#include <stdlib.h>
void srand(unsigned int seed);
```

设置新的伪随机数序列的种子为 seed。种子的初值为 1。

5.9 calloc

```
#include <stdlib.h>
void *calloc(size_t num, size_t size);
```

为 num 个大小为 size 的对象组成的数组分配足够的内存，并返回指向所分配区域的第一个字节的指针；如果内存不足以满足要求，则返回 NULL。

分配的内存区域中的所有位被初始化为 0。

5.10 malloc

```
#include <stdlib.h>
void *malloc(size_t size);
```

为大小为 size 的对象分配足够的内存，并返回指向所分配区域的第一个字节的指针；如果内存不足以满足要求，则返回 NULL。

不对分配的内存区域进行初始化。

5.11 realloc

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
```

将ptr指向的内存区域的大小改为size个字节。如果新分配的内存比原内存大，那么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，那么新内存保持原内存区中前size字节的内容。函数返回指向新分配空间的指针。如果不能满足要求，则返回NULL，原ptr指向的内存区域保持不变。

如果ptr为NULL，则行为等价于malloc(size)。

如果size为0，则行为等价于free(ptr)。

5.12 free

```
#include <stdlib.h>
void free(void *ptr);
```

释放ptr指向的内存空间，若ptr为NULL，则什么也不做。ptr必须指向先前用动态分配函数malloc、realloc或calloc分配的空间。

5.13 abort

```
#include <stdlib.h>
void abort(void);
```

使程序非正常终止。其功能类似于raise(SIGABRT)。

5.14 exit

```
#include <stdlib.h>
void exit(int status);
```

使程序正常终止。atexit 函数以与注册相反的顺序被调用，所有打开的文件被刷新，所有打开的流被关闭。status 的值如何被返回依具体的实现而定，但用 0 表示正常终止，也可用值 EXIT_SUCCESS 和 EXIT_FAILURE。

5.15 atexit

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

注册在程序正常终止时所要调用的函数 func。如果成功注册，则函数返回 0 值，否则返回非 0 值。

5.16 system

```
#include <stdlib.h>
int system(const char *str);
```

把字符串 str 传送给执行环境。如果 str 为 NULL，那么在存在命令处理程序时，返回 0 值。如果 str 的值非 NULL，则返回值与具体的实现有关。

5.17 getenv

```
#include <stdlib.h>
char *getenv(const char *name);
```

返回与 name 相关的环境字符串。如果该字符串不存在，则返回 NULL。其细节与具体的实现有关。

5.18 bsearch

```
#include <stdlib.h>
void *bsearch(const void *key, const void *base, size_t n,
size_t size,    int (*compare)(const void *, const void *));
```

在 base[0]...base[n-1]之间查找与*key 匹配的项。size 指出每个元素占有的字节数。函数返回一个指向匹配项的指针，若不存在匹配则返回 NULL。

函数指针 compare 指向的函数把关键字 key 和数组元素比较，比较函数的形式为：

```
int func_name(const void *arg1, const void *arg2);
```

arg1 是 key 指针，arg2 是数组元素指针。

返回值必须如下：

- arg1 < arg2 时，返回值<0；
- arg1 == arg2 时，返回值==0；
- arg1 > arg2 时，返回值>0。

数组 base 必须按升序排列(与 compare 函数定义的大小次序一致)。

5.19 qsort

```
#include <stdlib.h>
void qsort(void *base, size_t n, size_t size, \
        int (*compare)(const void *, const void *));
```

对由 n 个大小为 size 的对象构成的数组 base 进行升序排序。

比较函数 compare 的形式如下：

```
int func_name(const void *arg1, const voie *arg2);
```

其返回值必须如下所示：

- arg1 < arg2，返回值<0；
- arg1 == arg2，返回值==0；
- arg1 > arg2，返回值>0。

5.20 abs

```
#include <stdlib.h>
int abs(int num);
```

返回 int 变元 num 的绝对值。

5.21 labs

```
#include <stdlib.h>
long labs(long int num);
```

返回 long 类型变元 num 的绝对值。

5.22 div

```
#include <stdlib.h>
div_t div(int numerator, int denominator);
```

返回 numerator/denominator 的商和余数，结果分别保存在结构类型 div_t 的两个 int 成员 quot 和 rem 中。

5.23 ldiv

```
#include <stdlib.h>
ldiv_t div(long int numerator, long int denominator);
```

返回 numerator/denominator 的商和余数，结果分别保存在结构类型 ldiv_t 的两个 long 成员 quot 和 rem 中。

6 诊断<assert.h>

6.1 assert

```
#include <assert.h>
void assert(int exp);
```

assert 宏用于为程序增加诊断功能。当 assert(exp)执行时，如果 exp 为 0，则在标准出错输出流 stderr 输出一条如下所示的信息：

Assertion failed: *expression*, file *filename*, line *nnn*

然后调用 abort 终止执行。其中的源文件名 *filename* 和行号 *nnn* 来自于预处理宏__FILE__和__LINE__。

如果<assert.h>被包含时定义了宏 NDEBUG，那么宏 assert 被忽略。

7 变长变元表<stdarg.h>

头文件<stdarg.h>中的说明提供了依次处理含有未知数目和类型的函数变元表的机制。

7.1 va_start

7.2 va_arg

7.3 va_end

```
#include <stdarg.h>
void va_start(va_list ap, lastarg);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

假设函数 f 含有可变数目的变元，lastarg 是它的最后一个有名参数，然后在 f 内说明一个类型为 va_list 的变量 ap，它将依次指向每个变元：

```
    va_list ap;
```

在访问任何未命名的变元前必须用 va_start 宏对 ap 进行初始化：

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

此后，宏 va_arg 的每次执行将产生一个与下一个未命名的变元有相同类型和值的值，它同时还修改 ap，以使下一次使用 va_arg 时返回下一个变元：

```
va_arg(ap, type);
```

当所有的变元处理完毕之后，f 返回之前，必须调用一次宏 va_end：

```
va_end(ap);
```

例子：函数 sum_series()的第一个参数是变元项数。

```
double sum_series(int num, … )
{
    double sum = 0.0, t;
    va_list ap;

    va_start(ap, num);
    for (; num; num--) {
        t = va_arg(ap, double);
        sum += t;
    }
    va_end(ap);
    return sum;
}
```

8 非局部跳转<setjmp.h>

头文件<setjmp.h>中的说明提供了一种避免通常的函数调用和返回顺序的途径，特别的，它允许立即从一个多层嵌套的函数调用中返回。

8.1 setjmp

```
#include <setjmp.h>
int setjmp(jmp_buf env);
```

setjmp()宏把当前状态信息保存到 env 中，供以后 longjmp()恢复状态信息时使用。如果是直接调用 setjmp()，那么返回值为 0；如果是由于调用 longjmp()而调用 setjmp()，那么返回值非 0。setjmp()只能在某些特定情况下调用，如在 if 语句、switch 语句及循环语句的条件测试部分以及一些简单的关系表达式中。

8.2 longjmp

```
#include <setjmp.h>
void longjmp(jmp_buf env, int val);
```

longjmp()用于恢复由最近一次调用 setjmp()时保存到 env 的状态信息。当它执行完时，程序就象 setjmp()刚刚执行完并返回非 0 值 val 那样继续执行。包含 setjmp()宏调用的函数一定不能已经终止。所有可访问的对象的值都与调用 longjmp()时相同，唯一的例外是，那些调用 setjmp()宏的函数中的非 volatile 自动变量如果在调用 setjmp()后有了改变，那么就变成未定义的。

9 信号处理<signal.h>

头文件<signal.h>中提供了一些用于处理程序运行期间所引发的异常条件的功能，如处理来源于外部的中断信号或程序执行期间出现的错误等事件。

9.1 signal

```
#include <signal.h>
void (*signal(int sig, void (*handler)(int)))(int);
```

signal()用于确定以后当信号 sig 出现时的处理方法。如果 handler 的值是 SIG_DFL，那么就采用实现定义的缺省行为；如果 handler 的值是 SIG_IGN，那么就忽略该信号；否则，调用 handler 所指向的函数(参数为信号类型)。有效的信号包括：

SIGABRT	异常终止，如调用 abort()。
SIGFPE	算术运算出错，如除数为 0 或溢出。
SIGILL	非法函数映象，如非法指令。
SIGINT	交互式信号，如中断。
SIGSEGV	非法访问存储器，如访问不存在的内存单元。
SIGTERM	发送给本程序的终止请求信号。

signal()返回信号 sig 原来的的 handler；如果出错，则返回 SIG_ERR。

当随后出现信号 sig 时，就中断正在执行的操作，转而执行信号处理函数(*handler)(sig)。如果从信号处理程序中返回，则从中断的位置继续执行。

信号的初始状态由实现定义。

9.2 raise

```
#include <signal.h>
int raise(int sig);
```

向程序发送信号 sig。如果发送不成功，就返回一个非 0 值。

10 日期与时间函数<time.h>

头文件<time.h>中说明了一些用于处理日期和时间的类型和函数。其中的一部分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相同。clock_t 和 time_t 是两个用于表示时间的算术类型，而 struct tm 则用于存放日历时间的各个成分。tm 的各个成员的用途及取值范围如下：

```
    int tm_sec; /* 秒，0～61 */
    int tm_min; /* 分，0～59 */
    int tm_hour; /* 时，0～23 */
    int tm_mday; /* 日，1～31 */
    int tm_mon; /* 月(从1月开始)，0～11 */
    int tm_year; /* 年(从1900年开始) */
    int tm_wday; /* 星期(从周日开始)，0～6 */
    int tm_yday; /* 天数(从1月1日开始)，0～365 */
    int tm_isdst; /* 夏令时标记 */
```

其中，tm_isdst 在使用夏令时时其值为正，在不使用夏令时时其值为0，如果该信息不能使用，其值为负。

10.1 clock

```
#include <time.h>
clock_t clock(void);
```

返回程序自开始执行到目前为止所占用的处理机时间。如果处理机时间不可使用，那么返回-1。clock()/CLOCKS_PER_SEC 是以秒为单位表示的时间。

10.2 time

```
#include <time.h>
time_t time(time_t *tp);
```

返回当前日历时间。如果日历时间不能使用，则返回-1。如果 tp 不为 NULL，那么同时把返回值赋给*tp。

10.3 difftime

```
#include <time.h>
double difftime(time_t time2, time_t time1);
```

返回 time2-time1 的值(以秒为单位)。

10.4 mktime

```
#include <time.h>
time_t mktime(struct tm *tp);
```

将结构*tp 中的当地时间转换为 time_t 类型的日历时间，并返回该时间。如果不能转换，则返回-1。

10.5 asctime

```
#include <time.h>
char *asctime(const struct tm *tp);
```

将结构*tp 中的时间转换成如下所示的字符串形式：

```
day month date hours:minutes:seconds year\n\0
```

如：

```
Fri Apr 15 15:14:13 2005\n\0
```

返回指向该字符串的指针。字符串存储在可被其他调用重写的静态对象中。

10.6 ctime

```
#include <time.h>
char *ctime(const time_t *tp);
```

将*tp 中的日历时间转换为当地时间的字符串，并返回指向该字符串指针。字符串存储在可被其他调用重写的静态对象中。等价于如下调用：

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

10.7 gmtime

```
#include <time.h>
struct tm *gmtime(const time_t *tp);
```

将*tp 中的日历时间转换成 struct tm 结构形式的国际标准时间(UTC)，并返回指向该结构的指针。如果转换失败，返回 NULL。结构内容存储在可被其他调用重写的静态对象中。

10.8 localtime

```
#include <time.h>
struct tm *localtime(const time_t *tp);
```

将*tp 中的日历时间转换成 struct tm 结构形式的本地时间，并返回指向该结构的指针。结构内容存储在可被其他调用重写的静态对象中。

10.9 strftime

```
#include <time.h>
size_t strftime(char *s, size_t smax, const char *fmt, \
        const struct tm *tp);
```

根据 fmt 的格式说明把结构*tp 中的日期与时间信息转换成指定的格式，并存储到 s 所指向的数组中，写到 s 中的字符数不能多于 smax。函数返回实际写到 s 中的字符数(不包括'\0')；如果产生的字符数多于 smax，则返回 0。

fmt 类似于 printf()中的格式说明，它由 0 个或多个转换规格说明与普通字符组成。普通字符原封不动的拷贝到 s 中，每个%c 按照下面所描述的格式用与当地环境相适应的值来替换。转换规格列表如下：

格式	说明
%a	一星期中各天的缩写名
%A	一星期中各天的全名
%b	缩写月份名
%B	月份全名
%c	当地时间和日期表示
%d	用整数表示的一个月中的第几天(01～31)
%H	用整数表示的时(24 小时制，00～23)
%I	用整数表示的时(12 小时制，01～12)
%j	用整数表示的一年中各天(001～366)
%m	用整数表示的月份(01～12)
%M	用整数表示的分(00～59)
%p	与 AM/PM 对应的当地表示方法
%S	用整数表示的秒(00～61)
%U	用整数表示一年中的星期数(00～53，将星期日看作为每周的第一天)
%w	用整数表示一周中的各天(0～6，星期日为 0)

%W	用整数表示一年中的星期数(00～53，将星期一看作为每周的第一天)
%x	当地日期表示
%X	当地时间表示
%y	不带公元的年(00～99)
%Y	完整年份表示
%Z	时区名字(可获得时)
%%	%本身

11 由实现定义的限制<limits.h>和<float.h>

头文件<limits.h>中定义了用于表示整类型大小的常量。以下所列的值是可接受的最小值，实际系统中可能有更大的值。

CHAR_BIT	8	char 类型的位数
CHAR_MAX	UCHAR_MAX 或 SCHAR_MAX	char 类型的最大值
CHAR_MIN	0 或 SCHAR_MIN	char 类型的最小值
INT_MAX	32767	int 类型的最大值
INT_MIN	-32767	int 类型的最小值
LONG_MAX	2147483647	long 的最大值
LONG_MIN	-2147483647	long 类型的最小值
SCHAR_MAX	+127	signed char 类型的最大值
SCHAR_MIN	-127	signed char 类型的最小值
SHRT_MAX	+32767	short 类型的最大值
SHRT_MIN	-32767	short 类型的最小值
UCHAR_MAX	255	unsigned char 类型的最大值
UINT_MAX	65535	unsigned int 类型的最大值
ULONG_MAX	4294967295	unsigned long 的最大值
USHRT_MAX	65535	unsigned short 的最大值

以下是<float.h>的一个子集，是与浮点算术运算相关的一些常量。给出的每个值代表相应量的一个最小取值。实际实现可以定义适当的值。

FLT_RADIX	2	指数表示的基数，如 2、16
FLT_ROUNDS		加法的浮点舍入规则
FLT_DIG	6	float 类型精度(小数位数)

FLT_EPSILON	1E-5	使“1.0 + x != 1.0”成立的最小 *x*
FLT_MANT_DIG		基数为FLT_RADIX 的尾数中的数字数
FLT_MAX	1E+37	最大浮点数
FLT_MAX_EXP		使 FLT_RADIX^{n-1} 可表示的最大 *n*
FLT_MIN	1E-37	最小的规范化浮点数
FLT_MIN_EXP		使 10^n 为规范化数的最小 *n*
DBL_DIG	10	double 类型精度(小数位数)
DBL_EPSILON	1E-9	使“1.0 + x != 1.0”成立的最小 *x*
DBL_MANT_DIG		基数为FLT_RADIX 的尾数中的数字数
DBL_MAX	1E+37	最大双精度浮点数
DBL_MAX_EXP		使 FLT_RADIX^{n-1} 可表示的最大 *n*
DBL_MIN	1E-37	最小的规范化双精度浮点数
DBL_MIN_EXP		使 10^n 为规范化数的最小 *n*