C 标准库 - 参考手册

C 语言是一种通用的、面向过程式的计算机程序设计语言。1972 年，为了移植与开发 UNIX 操作系统，丹尼斯·里奇在贝尔电话实验室设计开发了 C 语言。

C 语言是一种广泛使用的计算机语言，它与 Java 编程语言一样普及，二者在现代软件程序员之间都得到广泛使用。

C 标准库是一组 C 内置函数、常量和头文件，比如 <stdio.h>、<stdlib.h>、<math.h>，等等。这个标准库可以作为 C 程序员的参考手册。

C 标准库 - <math.h>

简介

math.h 头文件定义了各种数学函数和一个宏。在这个库中所有可用的功能都带有一个 **double** 类型的参数，且都返回 **double** 类型的结果。

库宏

下面是这个库中定义的唯一的一个宏：

序号	宏 & 描述
1	**HUGE_VAL** 当函数的结果不可以表示为浮点数时。如果是因为结果的幅度太大以致于无法表示，则函数会设置 errno 为 ERANGE 来表示范围错误，并返回一个由宏 HUGE_VAL 或者它的否定（- HUGE_VAL）命名的一个特定的很大的值。 如果结果的幅度太小，则会返回零值。在这种情况下，error 可能会被设置为 ERANGE，也有可能不会被设置为 ERANGE。

库函数

下面列出了头文件 math.h 中定义的函数：

序号	函数 & 描述
1	double acos(double x) 返回以弧度表示的 x 的反余弦。
2	double asin(double x) 返回以弧度表示的 x 的反正弦。

3	double atan(double x) 返回以弧度表示的 x 的反正切。
4	double atan2(double y, double x) 返回以弧度表示的 y/x 的反正切。y 和 x 的值的符号决定了正确的象限。
5	double cos(double x) 返回弧度角 x 的余弦。
6	double cosh(double x) 返回 x 的双曲余弦。
7	double sin(double x) 返回弧度角 x 的正弦。
8	double sinh(double x) 返回 x 的双曲正弦。
9	double tanh(double x) 返回 x 的双曲正切。
10	double exp(double x) 返回 e 的 x 次幂的值。
11	double frexp(double x, int *exponent) 把浮点数 x 分解成尾数和指数。返回值是尾数，并将指数存入 exponent 中。所得的值是 x = mantissa * 2 ^ exponent。
12	double ldexp(double x, int exponent) 返回 x 乘以 2 的 exponent 次幂。
13	double log(double x) 返回 x 的自然对数（基数为 e 的对数）。
14	double log10(double x) 返回 x 的常用对数（基数为 10 的对数）。
15	double modf(double x, double *integer) 返回值为小数部分（小数点后的部分），并设置 integer 为整数部分。

16	double pow(double x, double y) 返回 x 的 y 次幂。
17	double sqrt(double x) 返回 x 的平方根。
18	double ceil(double x) 返回大于或等于 x 的最小的整数值。
19	double fabs(double x) 返回 x 的绝对值。
20	double floor(double x) 返回小于或等于 x 的最大的整数值。
21	double fmod(double x, double y) 返回 x 除以 y 的余数。

C 标准库 - <stdlib.h>

简介

stdlib .h 头文件定义了四个变量类型、一些宏和各种通用工具函数。

库变量

下面是头文件 stdlib.h 中定义的变量类型：

序号	变量 & 描述
1	**size_t** 这是无符号整数类型，它是 **sizeof** 关键字的结果。
2	**wchar_t** 这是一个宽字符常量大小的整数类型。
3	**div_t** 这是 **div** 函数返回的结构。
4	**ldiv_t**

	这是 **ldiv** 函数返回的结构。

库宏

下面是头文件 stdlib.h 中定义的宏：

序号	宏 & 描述
1	**NULL** 这个宏是一个空指针常量的值。
2	**EXIT_FAILURE** 这是 exit 函数失败时要返回的值。
3	**EXIT_SUCCESS** 这是 exit 函数成功时要返回的值。
4	**RAND_MAX** 这个宏是 rand 函数返回的最大值。
5	**MB_CUR_MAX** 这个宏表示在多字节字符集中的最大字符数，不能大于 MB_LEN_MAX。

库函数

下面是头文件 stdlib.h 中定义的函数：

序号	函数 & 描述
1	double atof(const char *str) 把参数 *str* 所指向的字符串转换为一个浮点数（类型为 double 型）。
2	int atoi(const char *str) 把参数 *str* 所指向的字符串转换为一个整数（类型为 int 型）。
3	long int atol(const char *str) 把参数 *str* 所指向的字符串转换为一个长整数（类型为 long int 型）。
4	double strtod(const char *str, char **endptr) 把参数 *str* 所指向的字符串转换为一个浮点数（类型为 double 型）。

5	long int strtol(const char *str, char **endptr, int base) 把参数 *str* 所指向的字符串转换为一个长整数（类型为 long int 型）。
6	unsigned long int strtoul(const char *str, char **endptr, int base) 把参数 *str* 所指向的字符串转换为一个无符号长整数（类型为 unsigned long int 型）。
7	void *calloc(size_t nitems, size_t size) 分配所需的内存空间，并返回一个指向它的指针。
8	void free(void *ptr) 释放之前调用 *calloc*、*malloc* 或 *realloc* 所分配的内存空间。
9	void *malloc(size_t size) 分配所需的内存空间，并返回一个指向它的指针。
10	void *realloc(void *ptr, size_t size) 尝试重新调整之前调用 *malloc* 或 *calloc* 所分配的 ptr 所指向的内存块的大小。
11	void abort(void) 使一个异常程序终止。
12	int atexit(void (*func)(void)) 当程序正常终止时，调用指定的函数 **func**。
13	void exit(int status) 使程序正常终止。
14	char *getenv(const char *name) 搜索 name 所指向的环境字符串，并返回相关的值给字符串。
15	int system(const char *string) 由 string 指定的命令传给要被命令处理器执行的主机环境。
16	void *bsearch(const void *key, const void *base, size_t nitems, size_t size, int (*compar)(const void *, const void *)) 执行二分查找。
17	void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*)) 数组排序。

18	int abs(int x) 返回 x 的绝对值。
19	div_t div(int numer, int denom) 分子除以分母。
20	long int labs(long int x) 返回 x 的绝对值。
21	ldiv_t ldiv(long int numer, long int denom) 分子除以分母。
22	int rand(void) 返回一个范围在 0 到 *RAND_MAX* 之间的伪随机数。
23	void srand(unsigned int seed) 该函数播种由函数 **rand** 使用的随机数发生器。
24	int mblen(const char *str, size_t n) 返回参数 *str* 所指向的多字节字符的长度。
25	size_t mbstowcs(schar_t *pwcs, const char *str, size_t n) 把参数 *str* 所指向的多字节字符的字符串转换为参数 *pwcs* 所指向的数组。
26	int mbtowc(whcar_t *pwc, const char *str, size_t n) 检查参数 *str* 所指向的多字节字符。
27	size_t wcstombs(char *str, const wchar_t *pwcs, size_t n) 把数组 *pwcs* 中存储的编码转换为多字节字符，并把它们存储在字符串 *str* 中。
28	int wctomb(char *str, wchar_t wchar) 检查对应于参数 *wchar* 所给出的多字节字符的编码。

C 标准库 - <assert.h>

简介

C 标准库的 **assert.h** 头文件提供了一个名为 **assert** 的宏，它可用于验证程序做出的假设，并在假设为假时输出诊断消息。

已定义的宏 **assert** 指向另一个宏 **NDEBUG**，宏 **NDEBUG** 不是 <assert.h> 的一部分。如果已在引用 <assert.h> 的源文件中定义 NDEBUG 为宏名称，则 **assert** 宏的定义如下：

```
#define assert(ignore) ((void)0)
```

库宏

下面列出了头文件 assert.h 中定义的唯一的函数：

序号	函数 & 描述
1	void assert(int expression) 这实际上是一个宏，不是一个函数，可用于在 C 程序中添加诊断。

C 标准库 -<ctype.h>

简介

C 标准库的 **ctype.h** 头文件提供了一些函数，可用于测试和映射字符。

这些函数接受 **int** 作为参数，它的值必须是 EOF 或表示为一个无符号字符。

如果参数 c 满足描述的条件，则这些函数返回非零（true）。如果参数 c 不满足描述的条件，则这些函数返回零。

库函数

下面列出了头文件 ctype.h 中定义的函数：

序号	函数 & 描述
1	int isalnum(int c) 该函数检查所传的字符是否是字母和数字。
2	int isalpha(int c) 该函数检查所传的字符是否是字母。
3	int iscntrl(int c) 该函数检查所传的字符是否是控制字符。
4	int isdigit(int c)

	该函数检查所传的字符是否是十进制数字。
5	int isgraph(int c) 该函数检查所传的字符是否有图形表示法。
6	int islower(int c) 该函数检查所传的字符是否是小写字母。
7	int isprint(int c) 该函数检查所传的字符是否是可打印的。
8	int ispunct(int c) 该函数检查所传的字符是否是标点符号字符。
9	int isspace(int c) 该函数检查所传的字符是否是空白字符。
10	int isupper(int c) 该函数检查所传的字符是否是大写字母。
11	int isxdigit(int c) 该函数检查所传的字符是否是十六进制数字。

标准库还包含了两个转换函数，它们接受并返回一个 "int"

序号	函数 & 描述
1	int tolower(int c) 该函数把大写字母转换为小写字母。
2	int toupper(int c) 该函数把小写字母转换为大写字母。

字符类

序号	字符类 & 描述
1	**数字** 完整的数字集合 { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
2	**十六进制数字**

	集合 { 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f }
3	**小写字母** 集合 { a b c d e f g h i j k l m n o p q r s t u v w x y z }
4	**大写字母** 集合 {A B C D E F G H I J K L M N O P Q R S T U V W X Y Z }
5	**字母** 小写字母和大写字母的集合
6	**字母数字字符** 数字、小写字母和大写字母的集合
7	**标点符号字符** 集合 ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ` { | } ~
8	**图形字符** 字母数字字符和标点符号字符的集合
9	**空格字符** 制表符、换行符、垂直制表符、换页符、回车符、空格符的集合。
10	**可打印字符** 字母数字字符、标点符号字符和空格字符的集合。
11	**控制字符** 在 ASCII 编码中，这些字符的八进制代码是从 000 到 037，以及 177（DEL）。
12	**空白字符** 包括空格符和制表符。
13	**字母字符** 小写字母和大写字母的集合。

C 标准库 - <errno.h>

简介

C 标准库的 **errno.h** 头文件定义了整数变量 **errno**，它是通过系统调用设置的，在错误事件中的某些库函数表明了什么发生了错误。该宏扩展为类型为 int 的可更改的左值，因此它可以被一个程序读取和修改。

在程序启动时，**errno** 设置为零，C 标准库中的特定函数修改它的值为一些非零值以表示某些类型的错误。您也可以在适当的时候修改它的值或重置为零。

errno.h 头文件定义了一系列表示不同错误代码的宏，这些宏应扩展为类型为 **int** 的整数常量表达式。

库宏

下面列出了头文件 errno.h 中定义的宏：

序号	宏 & 描述
1	extern int errno 这是通过系统调用设置的宏，在错误事件中的某些库函数表明了什么发生了错误。
2	EDOM Domain Error 这个宏表示一个域错误，它在输入参数超出数学函数定义的域时发生，errno 被设置为 EDOM。
3	ERANGE Range Error 这个宏表示一个范围错误，它在输入参数超出数学函数定义的范围时发生，errno 被设置为 ERANGE。

C 标准库 - <float.h>

简介

C 标准库的 **float.h** 头文件包含了一组与浮点值相关的依赖于平台的常量。这些常量是由 ANSI C 提出的，这让程序更具有可移植性。

在讲解这些常量之前，最好先弄清楚浮点数是由下面四个元素组成的：

组件	组件描述
S	符号（+/-）
b	指数表示的基数，2 表示二进制，10 表示十进制，16 表示十六进制，等等...
e	指数，一个介于最小值 e_{min} 和最大值 e_{max} 之间的整数。

p	精度，基数 b 的有效位数

基于以上 4 个组成部分，一个浮点数的值如下：

floating-point = (S) p x b^e

或

floating-point = (+/-) precision x $base^{exponent}$

库宏

下面的值是特定实现的，且是通过 #define 指令来定义的，这些值都不得低于下边所给出的值。请注意，所有的实例 FLT 是指类型 float，DBL 是指类型 double，LDBL 是指类型 long double。

宏	描述
FLT_ROUNDS	定义浮点加法的舍入模式，它可以是下列任何一个值： -1 - 无法确定 0 - 趋向于零 1 - 去最近的值 2 - 趋向于正无穷 3 - 趋向于负无穷
FLT_RADIX 2	这个宏定义了指数表示的基数。基数 2 表示二进制，基数 10 表示十进制，基数 16 表示十六进制。
FLT_MANT_DIG	这些宏定义了 FLT_RADIX 基数中的位数。

DBL_MANT_DIG LDBL_MANT_DIG	
FLT_DIG 6 DBL_DIG 10 LDBL_DIG 10	这些宏定义了舍入后不会改变表示的十进制数字的最大值（基数 10）。
FLT_MIN_EXP DBL_MIN_EXP LDBL_MIN_EXP	这些宏定义了基数为 FLT_RADIX 时的指数的最小负整数值。
FLT_MIN_10_EXP -37 DBL_MIN_10_EXP -37 LDBL_MIN_10_EXP -37	这些宏定义了基数为 10 时的指数的最小负整数值。
FLT_MAX_EXP DBL_MAX_EXP LDBL_MAX_EXP	这些宏定义了基数为 FLT_RADIX 时的指数的最大整数值。
FLT_MAX_10_EXP +37 DBL_MAX_10_EXP +37 LDBL_MAX_10_EXP +37	这些宏定义了基数为 10 时的指数的最大整数值。
FLT_MAX 1E+37 DBL_MAX 1E+37 LDBL_MAX 1E+37	这些宏定义最大的有限浮点值。
FLT_EPSILON 1E-5 DBL_EPSILON 1E-9 LDBL_EPSILON 1E-9	这些宏定义了可表示的最小有效数字。
FLT_MIN 1E-37 DBL_MIN 1E-37 LDBL_MIN 1E-37	这些宏定义了最小的浮点值。

实例

下面的实例演示了 float.h 文件中定义的一些常量的使用。

```
#include <stdio.h>

#include <float.h>


int main()

{

   printf("The maximum value of float = %.10e\n", FLT_MAX);

   printf("The minimum value of float = %.10e\n", FLT_MIN);


   printf("The number of digits in the number = %.10e\n", FLT_MANT_DIG);

}
```

让我们编译和运行上面的程序，这将产生下列结果：

```
The maximum value of float = 3.4028234664e+38

The minimum value of float = 1.1754943508e-38

The number of digits in the number = 7.2996655210e-312
```

C 标准库 - <limits.h>

简介

limits.h 头文件决定了各种变量类型的各种属性。定义在该头文件中的宏限制了各种变量类型（比如 char、int 和 long）的值。

这些限制指定了变量不能存储任何超出这些限制的值，例如一个无符号可以存储的最大值是 255。

库宏

下面的值是特定实现的，且是通过 #define 指令来定义的，这些值都不得低于下边所给出的值。

宏	值	描述
CHAR_BIT	8	定义一个字节的比特数。
SCHAR_MIN	-128	定义一个有符号字符的最小值。
SCHAR_MAX	127	定义一个有符号字符的最大值。
UCHAR_MAX	255	定义一个无符号字符的最大值。
CHAR_MIN	0	定义类型 char 的最小值，如果 char 表示负值，则它的值等于 SCHAR_MIN，否则等于 0。
CHAR_MAX	127	定义类型 char 的最大值，如果 char 表示负值，则它的值等于 SCHAR_MAX，否则等于 UCHAR_MAX。
MB_LEN_MAX	1	定义多字节字符中的最大字节数。
SHRT_MIN	-32768	定义一个短整型的最小值。
SHRT_MAX	+32767	定义一个短整型的最大值。
USHRT_MAX	65535	定义一个无符号短整型的最大值。
INT_MIN	-32768	定义一个整型的最小值。
INT_MAX	+32767	定义一个整型的最大值。
UINT_MAX	65535	定义一个无符号整型的最大值。
LONG_MIN	-2147483648	定义一个长整型的最小值。
LONG_MAX	+2147483647	定义一个长整型的最大值。
ULONG_MAX	4294967295	定义一个无符号长整型的最大值。

实例

下面的实例演示了 limit.h 文件中定义的一些常量的使用。

```
#include <stdio.h>#include <limits.h>

int main(){



   printf("The number of bits in a byte %d\n", CHAR_BIT);



   printf("The minimum value of SIGNED CHAR = %d\n", SCHAR_MIN);

   printf("The maximum value of SIGNED CHAR = %d\n", SCHAR_MAX);

   printf("The maximum value of UNSIGNED CHAR = %d\n", UCHAR_MAX);



   printf("The minimum value of SHORT INT = %d\n", SHRT_MIN);

   printf("The maximum value of SHORT INT = %d\n", SHRT_MAX);



   printf("The minimum value of INT = %d\n", INT_MIN);

   printf("The maximum value of INT = %d\n", INT_MAX);



   printf("The minimum value of CHAR = %d\n", CHAR_MIN);

   printf("The maximum value of CHAR = %d\n", CHAR_MAX);



   printf("The minimum value of LONG = %ld\n", LONG_MIN);

   printf("The maximum value of LONG = %ld\n", LONG_MAX);



   return(0);}
```

让我们编译和运行上面的程序，这将产生下列结果：

```
The number of bits in a byte 8The minimum value of SIGNED CHAR = -128The maximum value of SIGNED CHAR = 127The max
imum value of UNSIGNED CHAR = 255The minimum value of SHORT INT = -32768The maximum value of SHORT INT = 32767The
minimum value of INT = -32768The maximum value of INT = 32767The minimum value of CHAR = -128The maximum value of
CHAR = 127The minimum value of LONG = -2147483648The maximum value of LONG = 2147483647
```

C 标准库 - <locale.h>

简介

locale.h 头文件定义了特定地域的设置，比如日期格式和货币符号。接下来我们将介绍一些宏，以及一个重要的结构 **struct lconv** 和两个重要的函数。

库宏

下面列出了头文件 locale.h 中定义的宏，这些宏将在下列的两个函数中使用：

序号	宏 & 描述
1	**LC_ALL** 设置下面的所有选项。
2	**LC_COLLATE** 影响 strcoll 和 strxfrm 函数。
3	**LC_CTYPE** 影响所有字符函数。
4	**LC_MONETARY** 影响 localeconv 函数提供的货币信息。
5	**LC_NUMERIC** 影响 localeconv 函数提供的小数点格式化和信息。
6	**LC_TIME** 影响 strftime 函数。

库函数

下面列出了头文件 locale.h 中定义的函数：

序号	函数 & 描述

1	char *setlocale(int category, const char *locale) 设置或读取地域化信息。
2	struct lconv *localeconv(void) 设置或读取地域化信息。

库结构

```
typedef struct {
   char *decimal_point;
   char *thousands_sep;
   char *grouping;
   char *int_curr_symbol;
   char *currency_symbol;
   char *mon_decimal_point;
   char *mon_thousands_sep;
   char *mon_grouping;
   char *positive_sign;
   char *negative_sign;
   char int_frac_digits;
   char frac_digits;
```

```
   char p_cs_precedes;

   char p_sep_by_space;

   char n_cs_precedes;

   char n_sep_by_space;

   char p_sign_posn;

   char n_sign_posn;

} lconv
```

以下是各字段的描述：

序号	字段 & 描述
1	**decimal_point** 用于非货币值的小数点字符。
2	**thousands_sep** 用于非货币值的千位分隔符。
3	**grouping** 一个表示非货币量中每组数字大小的字符串。每个字符代表一个整数值，每个整数指定当前组的位数。值为 0 意味着前一个值将应用于剩余的分组。
4	**int_curr_symbol** 国际货币符号使用的字符串。前三个字符是由 ISO 4217:1987 指定的，第四个字符用于分隔货币符号和货币量。
5	**currency_symbol** 用于货币的本地符号。
6	**mon_decimal_point** 用于货币值的小数点字符。

7	**mon_thousands_sep** 用于货币值的千位分隔符。
8	**mon_grouping** 一个表示货币值中每组数字大小的字符串。每个字符代表一个整数值，每个整数指定当前组的位数。值为 0 意味着前一个值将应用于剩余的分组。
9	**positive_sign** 用于正货币值的字符。
10	**negative_sign** 用于负货币值的字符。
11	**int_frac_digits** 国际货币值中小数点后要显示的位数。
12	**frac_digits** 货币值中小数点后要显示的位数。
13	**p_cs_precedes** 如果等于 1，则 currency_symbol 出现在正货币值之前。如果等于 0，则 currency_symbol 出现在正货币值之后。
14	**p_sep_by_space** 如果等于 1，则 currency_symbol 和正货币值之间使用空格分隔。如果等于 0，则 currency_symbol 和正货币值之间不使用空格分隔。
15	**n_cs_precedes** 如果等于 1，则 currency_symbol 出现在负货币值之前。如果等于 0，则 currency_symbol 出现在负货币值之后。
16	**n_sep_by_space** 如果等于 1，则 currency_symbol 和负货币值之间使用空格分隔。如果等于 0，则 currency_symbol 和负货币值之间不使用空格分隔。
17	**p_sign_posn** 表示正货币值中正号的位置。
18	**n_sign_posn**

	表示负货币值中负号的位置。

下面的值用于 **p_sign_posn** 和 **n_sign_posn**:

值	描述
0	封装值和 currency_symbol 的括号。
1	放置在值和 currency_symbol 之前的符号。
2	放置在值和 currency_symbol 之后的符号。
3	紧挨着放置在值和 currency_symbol 之前的符号。
4	紧挨着放置在值和 currency_symbol 之后的符号。

C 标准库 - <setjmp.h>

简介

setjmp.h 头文件定义了宏 **setjmp()**、函数 **longjmp()** 和变量类型 **jmp_buf**，该变量类型会绕过正常的函数调用和返回规则。

库变量

下面列出了头文件 setjmp.h 中定义的变量：

序号	变量 & 描述
1	**jmp_buf** 这是一个用于存储宏 **setjmp()** 和函数 **longjmp()** 相关信息的数组类型。

库宏

下面是这个库中定义的唯一的一个宏：

序号	宏 & 描述
1	int setjmp(jmp_buf environment) 这个宏把当前环境保存在变量 **environment** 中，以便函数 **longjmp()** 后续使用。如果这个宏直接从宏调用中返回，则它会返回零，但是如果它从 **longjmp()** 函数调用中返回，则它会返回一个非零值。

库函数

下面是头文件 setjmp.h 中定义的唯一的一个函数：

序号	函数 & 描述
1	void longjmp(jmp_buf environment, int value) 该函数恢复最近一次调用 **setjmp()** 宏时保存的环境，**jmp_buf** 参数的设置是由之前调用 setjmp() 生成的。

C 标准库 - <signal.h>

简介

signal.h 头文件定义了一个变量类型 **sig_atomic_t**、两个函数调用和一些宏来处理程序执行期间报告的不同信号。

库变量

下面是头文件 signal.h 中定义的变量类型：

序号	变量 & 描述
1	**sig_atomic_t** 这是 **int** 类型，在信号处理程序中作为变量使用。它是一个对象的整数类型，该对象可以作为一个原子实体访问，即使存在异步信号时，该对象可以作为一个原子实体访问。

库宏

下面是头文件 signal.h 中定义的宏，这些宏将在下列两个函数中使用。**SIG_** 宏与 signal 函数一起使用来定义信号的功能。

序号	宏 & 描述
1	**SIG_DFL** 默认的信号处理程序。
2	**SIG_ERR** 表示一个信号错误。
3	**SIG_IGN** 忽视信号。

SIG 宏用于表示以下各种条件的信号码：

序号	宏 & 描述
1	**SIGABRT** 程序异常终止。

2	**SIGFPE** 算术运算出错，如除数为 0 或溢出。
3	**SIGILL** 非法函数映象，如非法指令。
4	**SIGINT** 中断信号，如 ctrl-C。
5	**SIGSEGV** 非法访问存储器，如访问不存在的内存单元。
6	**SIGTERM** 发送给本程序的终止请求信号。

库函数

下面是头文件 signal.h 中定义的函数：

序号	函数 & 描述
1	void (*signal(int sig, void (*func)(int)))(int) 该函数设置一个函数来处理信号，即信号处理程序。
2	int raise(int sig) 该函数会促使生成信号 **sig**。sig 参数与 SIG 宏兼容。

C 标准库 - <stdarg.h>

简介

stdarg.h 头文件定义了一个变量类型 **va_list** 和三个宏，这三个宏可用于在参数个数未知（即参数个数可变）时获取函数中的参数。

可变参数的函数通在参数列表的末尾是使用省略号(,...)定义的。

库变量

下面是头文件 stdarg.h 中定义的变量类型：

序号	变量 & 描述

1	**va_list** 这是一个适用于 **va_start()、va_arg()** 和 **va_end()** 这三个宏存储信息的类型。

库宏

下面是头文件 stdarg.h 中定义的宏：

序号	宏 & 描述
1	void va_start(va_list ap, last_arg) 这个宏初始化 **ap** 变量，它与 **va_arg** 和 **va_end** 宏是一起使用的。**last_arg** 是最后一个传递给函数的已知的固定参数，即省略号之前的参数。
2	type va_arg(va_list ap, type) 这个宏检索函数参数列表中类型为 **type** 的下一个参数。
3	void va_end(va_list ap) 这个宏允许使用了 **va_start** 宏的带有可变参数的函数返回。如果在从函数返回之前没有调用 **va_end**，则结果为未定义。

C 标准库 - <stddef.h>

简介

stddef .h 头文件定义了各种变量类型和宏。这些定义中的大部分也出现在其它头文件中。

库变量

下面是头文件 stddef.h 中定义的变量类型：

序号	变量 & 描述
1	**ptrdiff_t** 这是有符号整数类型，它是两个指针相减的结果。
2	**size_t** 这是无符号整数类型，它是 **sizeof** 关键字的结果。
3	**wchar_t** 这是一个宽字符常量大小的整数类型。

库宏

下面是头文件 stddef.h 中定义的宏：

序号	宏 & 描述
1	NULL 这个宏是一个空指针常量的值。
2	offsetof(type, member-designator) 这会生成一个类型为 size_t 的整型常量，它是一个结构成员相对于结构开头的字节偏移量。成员是由 *member-designator* 给定的，结构的名称是在 *type* 中给定的。

C 标准库 - <stdio.h>

简介

stdio .h 头文件定义了三个变量类型、一些宏和各种函数来执行输入和输出。

库变量

下面是头文件 stdio.h 中定义的变量类型：

序号	变量 & 描述
1	**size_t** 这是无符号整数类型，它是 **sizeof** 关键字的结果。
2	**FILE** 这是一个适合存储文件流信息的对象类型。
3	**fpos_t** 这是一个适合存储文件中任何位置的对象类型。

库宏

下面是头文件 stdio.h 中定义的宏：

序号	宏 & 描述
1	**NULL** 这个宏是一个空指针常量的值。
2	**_IOFBF**、**_IOLBF** 和 **_IONBF**

	这些宏扩展了带有特定值的整型常量表达式，并适用于 **setvbuf** 函数的第三个参数。
3	**BUFSIZ** 这个宏是一个整数，该整数代表了 **setbuf** 函数使用的缓冲区大小。
4	**EOF** 这个宏是一个表示已经到达文件结束的负整数。
5	**FOPEN_MAX** 这个宏是一个整数，该整数代表了系统可以同时打开的文件数量。
6	**FILENAME_MAX** 这个宏是一个整数，该整数代表了字符数组可以存储的文件名的最大长度。如果实现没有任何限制，则该值应为推荐的最大值。
7	**L_tmpnam** 这个宏是一个整数，该整数代表了字符数组可以存储的由 tmpnam 函数创建的临时文件名的最大长度。
8	**SEEK_CUR**、**SEEK_END** 和 **SEEK_SET** 这些宏是在 **fseek** 函数中使用，用于在一个文件中定位不同的位置。
9	**TMP_MAX** 这个宏是 tmpnam 函数可生成的独特文件名的最大数量。
10	**stderr**、**stdin** 和 **stdout** 这些宏是指向 FILE 类型的指针，分别对应于标准错误、标准输入和标准输出流。

库函数

下面是头文件 stdio.h 中定义的函数：

> 为了更好地理解函数，请按照下面的序列学习这些函数，因为第一个函数中创建的文件会在后续的函数中使用到。

序号	函数 & 描述
1	int fclose(FILE *stream) 关闭流 stream。刷新所有的缓冲区。
2	void clearerr(FILE *stream) 清除给定流 stream 的文件结束和错误标识符。

3	int feof(FILE *stream) 测试给定流 stream 的文件结束标识符。
4	int ferror(FILE *stream) 测试给定流 stream 的错误标识符。
5	int fflush(FILE *stream) 刷新流 stream 的输出缓冲区。
6	int fgetpos(FILE *stream, fpos_t *pos) 获取流 stream 的当前文件位置，并把它写入到 pos。
7	FILE *fopen(const char *filename, const char *mode) 使用给定的模式 mode 打开 filename 所指向的文件。
8	size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream) 从给定流 stream 读取数据到 ptr 所指向的数组中。
9	FILE *freopen(const char *filename, const char *mode, FILE *stream) 把一个新的文件名 filename 与给定的打开的流 stream 关联，同时关闭流中的旧文件。
10	int fseek(FILE *stream, long int offset, int whence) 设置流 stream 的文件位置为给定的偏移 offset，参数 *offset* 意味着从给定的 *whence* 位置查找的字节数。
11	int fsetpos(FILE *stream, const fpos_t *pos) 设置给定流 stream 的文件位置为给定的位置。参数 *pos* 是由函数 fgetpos 给定的位置。
12	long int ftell(FILE *stream) 返回给定流 stream 的当前文件位置。
13	size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream) 把 ptr 所指向的数组中的数据写入到给定流 stream 中。
14	int remove(const char *filename) 删除给定的文件名 filename，以便它不再被访问。
15	int rename(const char *old_filename, const char *new_filename) 把 old_filename 所指向的文件名改为 new_filename。

16	void rewind(FILE *stream) 设置文件位置为给定流 stream 的文件的开头。
17	void setbuf(FILE *stream, char *buffer) 定义流 stream 应如何缓冲。
18	int setvbuf(FILE *stream, char *buffer, int mode, size_t size) 另一个定义流 stream 应如何缓冲的函数。
19	FILE *tmpfile(void) 以二进制更新模式(wb+)创建临时文件。
20	char *tmpnam(char *str) 生成并返回一个有效的临时文件名，该文件名之前是不存在的。
21	int fprintf(FILE *stream, const char *format, ...) 发送格式化输出到流 stream 中。
22	int printf(const char *format, ...) 发送格式化输出到标准输出 stdout。
23	int sprintf(char *str, const char *format, ...) 发送格式化输出到字符串。
24	int vfprintf(FILE *stream, const char *format, va_list arg) 使用参数列表发送格式化输出到流 stream 中。
25	int vprintf(const char *format, va_list arg) 使用参数列表发送格式化输出到标准输出 stdout。
26	int vsprintf(char *str, const char *format, va_list arg) 使用参数列表发送格式化输出到字符串。
27	int fscanf(FILE *stream, const char *format, ...) 从流 stream 读取格式化输入。
28	int scanf(const char *format, ...) 从标准输入 stdin 读取格式化输入。

29	int sscanf(const char *str, const char *format, ...) 从字符串读取格式化输入。
30	int fgetc(FILE *stream) 从指定的流 stream 获取下一个字符（一个无符号字符），并把位置标识符往前移动。
31	char *fgets(char *str, int n, FILE *stream) 从指定的流 stream 读取一行，并把它存储在 str 所指向的字符串内。当读取 **(n-1)** 个字符时，或者读取到换行符时，或者到达文件末尾时，它会停止，具体视情况而定。
32	int fputc(int char, FILE *stream) 把参数 char 指定的字符（一个无符号字符）写入到指定的流 stream 中，并把位置标识符往前移动。
33	int fputs(const char *str, FILE *stream) 把字符串写入到指定的流 stream 中，但不包括空字符。
34	int getc(FILE *stream) 从指定的流 stream 获取下一个字符（一个无符号字符），并把位置标识符往前移动。
35	int getchar(void) 从标准输入 stdin 获取一个字符（一个无符号字符）。
36	char *gets(char *str) 从标准输入 stdin 读取一行，并把它存储在 str 所指向的字符串中。当读取到换行符时，或者到达文件末尾时，它会停止，具体视情况而定。
37	int putc(int char, FILE *stream) 把参数 char 指定的字符（一个无符号字符）写入到指定的流 stream 中，并把位置标识符往前移动。
38	int putchar(int char) 把参数 char 指定的字符（一个无符号字符）写入到标准输出 stdout 中。
39	int puts(const char *str) 把一个字符串写入到标准输出 stdout，直到空字符，但不包括空字符。换行符会被追加到输出中。
40	int ungetc(int char, FILE *stream) 把字符 char（一个无符号字符）推入到指定的流 stream 中，以便它是下一个被读取到的字符。
41	void perror(const char *str)

	把一个描述性错误消息输出到标准错误 stderr。首先输出字符串 str，后跟一个冒号，然后是一个空格。

C 标准库 - <string.h>

简介

string .h 头文件定义了一个变量类型、一个宏和各种操作字符数组的函数。

库变量

下面是头文件 string.h 中定义的变量类型：

序号	变量 & 描述
1	**size_t** 这是无符号整数类型，它是 **sizeof** 关键字的结果。

库宏

下面是头文件 string.h 中定义的宏：

序号	宏 & 描述
1	**NULL** 这个宏是一个空指针常量的值。

库函数

下面是头文件 string.h 中定义的函数：

序号	函数 & 描述
1	void *memchr(const void *str, int c, size_t n) 在参数 *str* 所指向的字符串的前 n 个字节中搜索第一次出现字符 c（一个无符号字符）的位置。
2	int memcmp(const void *str1, const void *str2, size_t n) 把 *str1* 和 *str2* 的前 n 个字节进行比较。
3	void *memcpy(void *dest, const void *src, size_t n) 从 src 复制 n 个字符到 *dest*。
4	void *memmove(void *dest, const void *src, size_t n)

	另一个用于从 *src* 复制 n 个字符到 *dest* 的函数。
5	void *memset(void *str, int c, size_t n) 复制字符 c（一个无符号字符）到参数 *str* 所指向的字符串的前 n 个字符。
6	char *strcat(char *dest, const char *src) 把 *src* 所指向的字符串追加到 *dest* 所指向的字符串的结尾。
7	char *strncat(char *dest, const char *src, size_t n) 把 *src* 所指向的字符串追加到 *dest* 所指向的字符串的结尾，直到 n 字符长度为止。
8	char *strchr(const char *str, int c) 在参数 *str* 所指向的字符串中搜索第一次出现字符 c（一个无符号字符）的位置。
9	int strcmp(const char *str1, const char *str2) 把 *str1* 所指向的字符串和 *str2* 所指向的字符串进行比较。 返回 0 表示相等，非 0 表示不相等
10	int strncmp(const char *str1, const char *str2, size_t n) 把 *str1* 和 *str2* 进行比较，最多比较前 n 个字节。
11	int strcoll(const char *str1, const char *str2) 把 *str1* 和 *str2* 进行比较，结果取决于 LC_COLLATE 的位置设置。
12	char *strcpy(char *dest, const char *src) 把 *src* 所指向的字符串复制到 *dest*。
13	char *strncpy(char *dest, const char *src, size_t n) 把 *src* 所指向的字符串复制到 *dest*，最多复制 n 个字符。
14	size_t strcspn(const char *str1, const char *str2) 检索字符串 str1 开头连续有几个字符都不含字符串 str2 中的字符。
15	char *strerror(int errnum) 从内部数组中搜索错误号 errnum，并返回一个指向错误消息字符串的指针。
16	size_t strlen(const char *str) 计算字符串 str 的长度，直到空结束字符，但不包括空结束字符。

17	char *strpbrk(const char *str1, const char *str2) 检索字符串 *str1* 中第一个匹配字符串 *str2* 中字符的字符，不包含空结束字符。也就是说，依次检验字符串 str1 中的字符，当被检验字符在字符串 str2 中也包含时，则停止检验，并返回该字符位置。
18	char *strrchr(const char *str, int c) 在参数 *str* 所指向的字符串中搜索最后一次出现字符 c（一个无符号字符）的位置。
19	size_t strspn(const char *str1, const char *str2) 检索字符串 *str1* 中第一个不在字符串 *str2* 中出现的字符下标。
20	char *strstr(const char *haystack, const char *needle) 在字符串 *haystack* 中查找第一次出现字符串 *needle*（不包含空结束字符）的位置。
21	char *strtok(char *str, const char *delim) 分解字符串 *str* 为一组字符串，*delim* 为分隔符。
22	size_t strxfrm(char *dest, const char *src, size_t n) 根据程序当前的区域选项中的 LC_COLLATE 来转换字符串 **src** 的前 **n** 个字符，并把它们放置在字符串 **dest** 中。

C 标准库 - <time.h>

简介

time.h 头文件定义了四个变量类型、两个宏和各种操作日期和时间的函数。

库变量

下面是头文件 time.h 中定义的变量类型：

序号	变量 & 描述
1	**size_t** 是无符号整数类型，它是 **sizeof** 关键字的结果。
2	**clock_t** 这是一个适合存储处理器时间的类型。
3	**time_t is**

	这是一个适合存储日历时间类型。
4	**struct tm** 这是一个用来保存时间和日期的结构。

tm 结构的定义如下：

```
struct tm {

   int tm_sec;         /* 秒，范围从 0 到 59        */

   int tm_min;         /* 分，范围从 0 到 59        */

   int tm_hour;        /* 小时，范围从 0 到 23        */

   int tm_mday;        /* 一月中的第几天，范围从 1 到 31    */

   int tm_mon;         /* 月，范围从 0 到 11        */

   int tm_year;        /* 自 1900 年起的年数        */

   int tm_wday;        /* 一周中的第几天，范围从 0 到 6    */

   int tm_yday;        /* 一年中的第几天，范围从 0 到 365    */

   int tm_isdst;       /* 夏令时                */};
```

库宏

下面是头文件 time.h 中定义的宏：

序号	宏 & 描述
1	**NULL** 这个宏是一个空指针常量的值。
2	**CLOCKS_PER_SEC** 这个宏表示每秒的处理器时钟个数。

库函数

下面是头文件 time.h 中定义的函数：

序号	函数 & 描述
1	char *asctime(const struct tm *timeptr)

	返回一个指向字符串的指针，它代表了结构 timeptr 的日期和时间。
2	clock_t clock(void) 返回程序执行起（一般为程序的开头），处理器时钟所使用的时间。
3	char *ctime(const time_t *timer) 返回一个表示当地时间的字符串，当地时间是基于参数 timer。
4	double difftime(time_t time1, time_t time2) 返回 time1 和 time2 之间相差的秒数 (time1-time2)。
5	struct tm *gmtime(const time_t *timer) timer 的值被分解为 tm 结构，并用协调世界时（UTC）也被称为格林尼治标准时间（GMT）表示。
6	struct tm *localtime(const time_t *timer) timer 的值被分解为 tm 结构，并用本地时区表示。
7	time_t mktime(struct tm *timeptr) 把 timeptr 所指向的结构转换为一个依据本地时区的 time_t 值。
8	size_t strftime(char *str, size_t maxsize, const char *format, const struct tm *timeptr) 根据 format 中定义的格式化规则，格式化结构 timeptr 表示的时间，并把它存储在 str 中。
9	time_t time(time_t *timer) 计算当前日历时间，并把它编码成 time_t 格式。