新核库函数手册_公开

Version 0.01

2021 年 2 月

修正记录

版本	日期	描述
V0.01	2021-2-04	

目录

ABS

概要

#include <stdlib.h>

int abs (int j)

说明

abs() 函数将返回 j 的绝对值。

示例

```
#include <stdio.h>
#include <stdlib.h>
void
main (void)
{
int a = -5;
printf(“The absolute value of %d is %d\n”, a, abs(a));
}
```

另请参见

labs() 和 fabs()

返回值

j 的绝对值。

ACOS

概要

#include <math.h>

double acos (double f)

说明

acos() 函数实现 cos() 的逆运算；即，向它传递一个介于范围 -1 至 +1 的值，它将返回以弧度表示的角度，其余弦等于该值。

示例

```
#include <math.h>
#include <stdio.h>
/* Print acos() values for -1 to 1 in degrees. */
void
main (void)
{
float i, a;
for(i = -1.0; i < 1.0 ; i += 0.1) {
a = acos(i)*180.0/3.141592;
printf("acos(%f) = %f degrees\n", i, a);
}
}
```

另请参见

sin()、 cos()、 tan()、 asin()、 atan() 和 atan2()

返回值

以弧度表示的角度，范围为 0 至 π。

ASCTIME

概要

#include <time.h>

char * asctime (struct tm * t)

说明

asctime() 函数接受 struct tm 结构形式的时间 （函数的参数指向该结构），并返回一个描述当前日期和时间的 26 字符的字符串，其格式为：

Sun Sep 16 01:03:52 1973\n\0

请注意字符串末尾的换行符。字符串中每个字段的宽度都是固定的。以下示例将获取当前时间，使用 localtime() 将其转换为 struct tm，然后它将其转换为 ASCII，并进行打印。 time() 函数需要由用户提供 （详情请参见 time()）。

示例

```
#include <stdio.h>
#include <time.h>
void
main (void)
{
time_t clock;
struct tm * tp;
time(&clock);
tp = localtime(&clock);
printf("%s", asctime(tp));
}
```

另请参见

ctime()、 gmtime()、 localtime() 和 time()

返回值

指向字符串的指针。

注

示例要求用户提供 time() 程序，因为它无法由编译器提供。更多详细信息，请参见 time()

ASIN

概要

#include <math.h>

double asin (double f)

说明

asin() 函数实现 sin() 的逆运算；即，向它传递一个介于范围 -1 至 +1 的值，它将返回以弧度表示的角度，其正弦等于该值。

示例

```
#include <math.h>
#include <stdio.h>
void
main (void)
{
float i, a;
for(i = -1.0; i < 1.0 ; i += 0.1) {
a = asin(i)*180.0/3.141592;
printf("asin(%f) = %f degrees\n", i, a);
}
}
```

另请参见

sin()、 cos()、 tan()、 acos()、 atan() 和 atan2()

返回值

以弧度表示的角度，介于范围 -π/2 至 π/2。

ASSERT

概要

#include <assert.h>

void assert (int e)

说明

该宏用于调试目的；基本使用方法是将断言自由地放置在代码中的这种位置：代码的正确操作取决于某些特定条件在初始时为真。 assert() 程序可用于在运行时确保某个假设为真。例如，以下语句断言 tp 不等于 NULL：

assert(tp);

如果在运行时表达式的求值结果为假，程序将中止并发出一条消息，指示断言的源文件和行号，以及用作其参数的表达式。由于篇幅有限，无法更全面地介绍 assert() 的用法，但它与验证程序正确性的方法是密切相关的。

assert() 宏依赖于函数 _fassert() 的实现。默认情况下，它使用 printf() 打印信息。应对该程序进行检查，以确保它满足应用程序的需求。将该函数的源文件包含到项目中（即使并未修改它），然后重新编译。 _fassert()函数不包含在任何库文件中。

示例

```
#include <assert.h>
void
ptrfunc (struct xyz * tp)
{
assert(tp != 0);
}
```

注

用户需要实现底层程序 _fassert(...)。

ATAN

概要

#include <math.h>

double atan (double x)

说明

该函数返回其参数的反正切；即，它返回范围 -π/2 至 π/2 之间的角度 “e”。

示例

```
#include <stdio.h>
#include <math.h>
void
main (void)
{
printf(“atan(%f) is %f\n”, 1.5, atan(1.5));
}
```

另请参见

sin()、 cos()、 tan()、 asin()、 acos() 和 atan2()

返回值

其参数的反正切。

ATAN2

概要

#include <math.h>

double atan2 (double y , double x)

说明

该函数返回 y/x 的反正切。

示例

```
#include <stdio.h>
#include <math.h>
void
main (void)
{
printf(“atan2(%f, %f) is %f\n”, 10.0, -10.0, atan2(10.0, -10.0));
}
```

另请参见

sin()、 cos()、 tan()、 asin()、 acos() 和 atan()

返回值

y/x 的反正切。

ATOF

概要

#include <stdlib.h>

double atof (const char * s)

说明

atof() 函数将扫描传递给它的字符串，并略过前导空格。然后，它会将数字的 ASCII 表示形式转换为双精度值。数字可以采用十进制、正常浮点数或科学记数法表示。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
char buf[80];
double i;
gets(buf);
i = atof(buf);
printf("Read %s: converted to %f\n", buf, i);
}
```

另请参见

atoi()、 atol() 和 strtod()

返回值

双精度浮点数。如果未在字符串中找到任何数字，则返回 0.0。

ATOI

概要

#include <stdlib.h>

int atoi (const char * s)

说明

atoi() 函数将扫描传递给它的字符串，略过前导空格并读取可选的符号。然后，它会将十进制数字的 ASCII 表示形式转换为整型。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
char buf[80];
int i;
gets(buf);
i = atoi(buf);
printf("Read %s: converted to %d\n", buf, i);
}
```

另请参见

xtoi()、 atof() 和 atol()

返回值

有符号整型。如果未在字符串中找到任何数字，则返回 0。

ATOL

概要

#include <stdlib.h>

long atol (const char * s)

说明

atol() 函数将扫描传递给它的字符串，并略过前导空格。然后，它会将十进制数字的 ASCII 表示形式转换为一个长整型值。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
char buf[80];
long i;
gets(buf);
i = atol(buf);
printf("Read %s: converted to %ld\n", buf, i);
}
```

另请参见

atoi() 和 atof()

返回值

长整型值。如果未在字符串中找到任何数字，则返回 0。

BSEARCH

概要

#include <stdlib.h>

void * bsearch (const void * key, void * base, size_t n_memb,

size_t size, int (*compar)(const void *, const void *))

说明

bsearch() 函数用于在有序数组中搜索与某个特定键匹配的元素。它使用二叉树搜索算法，并调用 compar 指向的函数来比较数组中的元素。

示例

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
struct value {
char name[10];
int value;
} values[] = {
{ "foobar", 66 };
{ "casbar", 87 };
{ "crossbar", 105 };
};
int
val_cmp (const void * p1, const void * p2)
{
return strcmp(((const struct value *)p1)->name,
((const struct value *)p2)->name);
}
void
main (void)
{
int i = sizeof(values)/sizeof(struct value);
struct value * vp;
qsort(values, i, sizeof values[0], val_cmp);
vp = bsearch(“fred”, values, i, sizeof values[0], val_cmp);
if(!vp)
printf(“Item ’fred’ was not found\n”);
else
printf(“Item ’fred’ has value %d\n”, vp->value);
}
```

另请参见

qsort()

返回值

指向匹配数组元素的指针 （如果存在多个匹配元素，则可能返回其中任意一个）。如果找不到匹配项，则返回空值。

注

比较函数必须具有正确的原型。

CEIL

概要

#include <math.h>

double ceil (double f)

说明

该程序返回不小于 f 的最小整数。

示例

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
void
main (void)
{
double j;
j = 2.345 * rand()
printf(“The ceiling of %f is %f\n”, j, ceil(j));









}
```

CGETS

概要

#include <conio.h>

char * cgets (char * s)

说明

cgets() 函数将从控制台读取一行输入并放入作为参数传递的缓冲区中。它通过重复调用 getche() 来实现该功能。在读取字符时，将会对它们进行缓冲，并且使用退格键可删除先前键入的字符，使用 ctrl-U 可删除目前为止键入的整行内容。其他字符会被放入缓冲区中，使用回车或换行 （换行符）可终止该函数。收集到的字符串使用空值终止。

示例

```
#include <conio.h>
#include <string.h>
char buffer[80];
void
main (void)
{
for(;;) {
cgets(buffer);
if(strcmp(buffer, “exit” == 0)
break;
cputs(“Type ’exit’ to finish\n”);
}
}
```

另请参见

getch()、 getche()、 putch() 和 cputs()

返回值

返回值是作为唯一参数传递的字符。

CLRWDT

概要

#include <zc.h>

CLRWDT();

说明

该宏用于清零器件的内部看门狗定时器。

示例

```
#include <zc.h>
void
main (void)
{
CLRWDT();
}
```

COS

概要

#include <math.h>

double cos (double f)

说明

该函数将产生其参数的余弦值，它是一个以弧度表示的角度。余弦值通过展开多项式级数逼近来计算。

示例

```
#include <math.h>
#include <stdio.h>
#define C 3.141592/180.0
void
main (void)
{
double i;
for(i = 0 ; i <= 180.0 ; i += 10)
printf("sin(%3.0f) = %f, cos = %f\n", i, sin(i*C), cos(i*C));
}
```

另请参见

sin()、 tan()、 asin()、 acos()、 atan() 和 atan2()

返回值

介于范围 -1 至 +1 的双精度值。

COSH、 SINH 和 TANH

概要

#include <math.h>

double cosh (double f)

double sinh (double f)

double tanh (double f)

说明

这些函数实现三角函数 cos()、 sin() 和 tan() 的双曲对等形式。

示例

```
#include <stdio.h>
#include <math.h>
void
main (void)
{
printf(“%f\n”, cosh(1.5));
printf(“%f\n”, sinh(1.5));
printf(“%f\n”, tanh(1.5));
}
```

返回值

函数 cosh() 返回双曲余弦值。

函数 sinh() 返回双曲正弦值。

函数 tanh() 返回双曲正切值。

CPUTS

概要

#include <conio.h>

void cputs (const char * s)

说明

cputs() 函数会将其参数字符串写入控制台，并在字符串中每个换行符之前输出回车符。它会重复调用 putch()。在主机系统上， cputs() 与 puts() 的区别在于它将直接写入控制台，而不是使用文件 I/O。在嵌入式系统中， cputs() 和 puts() 是等效的。

示例

```
#include <conio.h>
#include <string.h>
char buffer[80];
void
main (void)
{
for(;;) {
cgets(buffer);
if(strcmp(buffer, “exit” == 0)
break;
cputs(“Type ’exit’ to finish\n”;
}
}
```

另请参见

cputs()、 puts() 和 putch()

CTIME

概要

#include <time.h>

char * ctime (time_t * t)

说明

ctime() 函数会将其参数指向的以秒表示的时间转换为 asctime() 所描述的相同格式的字符串。因此，示例程序将打印当前时间和日期。

示例

```
#include <stdio.h>
#include <time.h>
void
main (void)
{
time_t clock;
time(&clock);
printf("%s", ctime(&clock));
}
```

另请参见

gmtime()、 localtime()、 asctime() 和 time()

返回值

指向字符串的指针。

注

示例要求用户提供 time() 程序，因为它无法由编译器提供。更多详细信息，请参见 time()。

DIH、DIL 和 EIH、EIL

概要

#include <zc.h>

void eih (void)

void eil (void)

void dih (void)

void dil(void)

说明

dih() 、dil() 和 eih()、eil() 程序分别用于禁止高、低中断和重新允许高、低中断。它们都以宏的形式实现。以下示例显示了在某个长整型变量 （该变量在中断期间会被修改）的访问代码周围使用 dih() 、dil() 和 eih()、eil()。在不进行这种处理的情况下，如果在访问计数值的连续字之间发生了中断，将有可能返回不正确的值。

永远不要在中断函数中调用 eih()、eil() 宏，在中断函数中也不需要调用 dih() 、dil()。

示例

```
#include <zc.h>
long count;
void
interrupt tick (void)
{
count++;
}
long
getticks (void)
{
long val; /* Disable interrupts around access
to count, to ensure consistency.*/
dih();
val = count;
eih();
return val;
}
```

DIV

概要

#include <stdlib.h>

div_t div (int numer, int denom)

说明

div() 函数计算将分子除以分母得到商和余数。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
div_t x;
x = div(12345, 66);
printf("quotient = %d, remainder = %d\n", x.quot, x.rem);
}
```

另请参见

udiv()、 ldiv() 和 uldiv()

返回值

在 div_t 结构中返回商和余数。

EVAL_POLY

概要

#include <math.h>

double eval_poly (double x, const double * d, int n)

说明

eval_poly() 函数在 x 点对多项式进行求值，其系数包含在数组 d 中，例如：

y = x*x*d2 + x*d1 + d0

多项式的阶数通过 n 传递。

示例

```
#include <stdio.h>
#include <math.h>
void
main (void)
{
double x, y;
double d[3] = {1.1, 3.5, 2.7};
x = 2.2;
y = eval_poly(x, d, 2);
printf(“The polynomial evaluated at %f is %f\n”, x, y);
}
```

返回值

多项式在 x 点进行求值得到的双精度值。

EXP

概要

#include <math.h>

double exp (double f)

说明

exp() 程序返回其参数的指数函数；即， e 的 f 次幂。

示例

```
#include <math.h>
#include <stdio.h>
void
main (void)
{
double f;
for(f = 0.0 ; f <= 5 ; f += 1.0)
printf("e to %1.0f = %f\n", f, exp(f));
}
```

另请参见

log()、 log10() 和 pow()

FABS

概要

#include <math.h>

double fabs (double f)

说明

该程序返回其双精度型参数的绝对值。

示例

```
#include <stdio.h>
#include <math.h>
void
main (void)
{
printf("%f %f\n", fabs(1.5), fabs(-1.5));
}
```

另请参见

abs() 和 labs()

FLOOR

概要

#include <math.h>

double floor (double f)

说明

该程序返回不大于 f 的最大整数。

示例

```
#include <stdio.h>
#include <math.h>
void
main (void)
{
printf("%f\n", floor( 1.5 ));
printf("%f\n", floor( -1.5));
}
```

FMOD

概要

#include <math.h>

double fmod (double x, double y)

说明

函数 fmod 以浮点数的形式返回 x/y 的余数。

示例

```
#include <math.h>
void
main (void)
{
double rem, x;
x = 12.34;
rem = fmod(x, 2.1);
}
```

返回值

x/y 的浮点型余数。

FREXP

概要

#include <math.h>

double frexp (double f, int * p)

说明

frexp() 函数将浮点数分成归一化小数部分和 2 的整数次幂。整数存储到 p 指向的 int 对象中。它的返回值 x 处于区间 (0.5, 1.0) 中或为零， f 等于 x 乘以 2 的整数次幂 （整数存储在 *p 中）。如果 f 为零，则结果的两个部分均为零。

示例

```
#include <math.h>
#include <stdio.h>
void
main (void)
{
double f;
int i;
f = frexp(23456.34, &i);
printf("23456.34 = %f * 2^%d\n", f, i);
}
```

另请参见

ldexp()

FTOA

概要

#include <stdlib.h>

char * ftoa (float f, int * status)

说明

函数 ftoa 将 f 的内容转换为一个字符串，字符串存储到之后返回的缓冲区中。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
char * buf;
float input = 12.34;
int status;
buf = ftoa(input, &status);
printf("The buffer holds %s\n", buf);
}
```

另请参见

strtol()、 itoa()、 utoa() 和 ultoa()

返回值

该程序返回对写入结果的缓冲区的引用。

GETCH

概要

#include <conio.h>

char getch (void)

说明

getch() 函数以空桩的形式提供，它可以在每个项目需要时完成。通常，该函数将从与 stdin 关联的外设中读取一个数据字节，并返回其值。

示例

```
#include <conio.h>
char result;
void
main (void)
{
result = getch();
}
```

另请参见

getche() 和 getchar()

GETCHE

概要

#include <conio.h>

char getche (void)

说明

getche() 函数以空桩的形式提供，它可以在每个项目需要时完成。通常，该函数将从与 stdin 关联的外设中读取一个数据字节，并返回其值。与 getch() 不同，它会回显接收到的这个字符。

示例

```
#include <conio.h>
char result;
void
main (void)
{
result = getche();
}
```

另请参见

getch() 和 getchar()

GETCHAR

概要

#include <stdio.h>

int getchar (void)

说明

getchar() 程序通常从 stdin 进行读取，但它以调用 getche() 的形式实现。

示例

```
#include <stdio.h>
void
main (void)
{
int c;
while((c = getchar()) != EOF)
putchar(c);
}
```

另请参见

getc() 和 getche()

注

该程序会调用 getche()。

GETS

概要

#include <stdio.h>

char * gets (char * s)

说明

gets() 函数从标准输入读取一行并放入 s 处的缓冲区中，并删除换行符 （比较：fgets()）。缓冲区使用空值终止。在嵌入式系统中， gets() 等效于 cgets()，导致重复地调用 getche() 来获取字符。可以进行编辑 （使用退格键）。

示例

```
#include <stdio.h>
void
main (void)
{
char buf[80];
printf("Type a line: ");
if(gets(buf))
puts(buf);
}
```

另请参见

fgets()、 freopen() 和 puts()

返回值

它返回其参数，或对于文件结束符返回 NULL。

GMTIME

概要

#include <time.h>

struct tm * gmtime (time_t * t)

说明

该函数将 t 指向的时间 （自 1970 年 1 月 1 日 00:00:00 以来的秒数）转换为分解的时间，存储在 time.h 中定义的结构中。该结构在 “数据类型”部分中定义。

示例

```
#include <stdio.h>
#include <time.h>
void
main (void)
{
time_t clock;
struct tm * tp;
time(&clock);
tp = gmtime(&clock);
printf("It’s %d in London\n", tp->tm_year+1900);
}
```

另请参见

ctime()、 asctime()、 time() 和 localtime()

返回值

返回 tm 类型的结构。

注

示例要求用户提供 time() 程序，因为它无法由编译器提供。更多详细信息，请参见 time()。

ISALNUM、 ISALPHA、 ISDIGIT 和 ISLOWER 等

概要

#include <ctype.h>

int isalnum (char c)

int isalpha (char c)

int isascii (char c)

int iscntrl (char c)

int isdigit (char c)

int islower (char c)

int isprint (char c)

int isgraph (char c)

int ispunct (char c)

int isspace (char c)

int isupper (char c)

int isxdigit(char c)

说明

这些宏（在 ctype.h 中定义）测试所提供的字符是否属于几种重叠字符组之一的成员。请注意， isascii() 除外，所有这些宏都是针对 c 定义的：如果 isascii(c) 为 true，或如果 c = EOF。

isalnum(c) c 处于 0-9、 a-z 或 A-Z 中

isalpha(c) c 处于 A-Z 或 a-z 中

isascii(c) c 为 7 位 ASCII 字符

iscntrl(c) c 为控制字符

isdigit(c) c 为十进制数字

islower(c) c 处于 a-z 中

isprint(c) c 为打印字符

isgraph(c) c 为非空格可打印字符

ispunct(c) c 不是字母数字

isspace(c) c 是空格、制表符或换行符

isupper(c) c 处于 A-Z 中

isxdigit(c) c 处于 0-9、 a-f 或 A-F 中

示例

```
#include <ctype.h>
#include <stdio.h>
void
main (void)
{
char buf[80];
int i;
gets(buf);
i = 0;
while(isalnum(buf[i]))
```

```
i++;
buf[i] = 0;
printf("'%s' is the word\n", buf);
}
```

另请参见

toupper()、 tolower() 和 toascii()

ISDIG

概要

#include <ctype.h>

int isdig (int c)

说明

isdig() 函数测试输入字符 c，确定它是否为十进制数字（0 – 9），如果是则返回 true；否则返回 false。

示例

```
#include <ctype.h>
void
main (void)
{
char buf[] = "1998a";
if(isdig(buf[0]))
printf(" type detected\n");
}
```

另请参见

isdigit() （列在 isalnum() 下）

返回值

如果为十进制数字则返回零，否则返回一个非零值。

ITOA

概要

#include <stdlib.h>

char * itoa (char * buf, int val, int base)

说明

函数 itoa 将 val 的内容转换为一个字符串，字符串存储到 buf 中。转换将根据 base 中指定的基数执行。假定 buf 引用的缓冲区分配了足够的空间。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
char buf[10];
itoa(buf, 1234, 16);
printf("The buffer holds %s\n", buf);
}
```

另请参见

strtol()、 utoa()、 ltoa() 和 ultoa()

返回值

该程序返回写入结果的缓冲区的副本。

LABS

概要

#include <stdlib.h>

int labs (long int j)

说明

labs() 函数将返回长整型值 j 的绝对值。

示例

```
#include <stdio.h>
#include <stdlib.h>
void
main (void)
{
long int a = -5;
printf("The absolute value of %ld is %ld\n", a, labs(a));
}
```

另请参见

abs()

返回值

j 的绝对值。

LDEXP

概要

#include <math.h>

double ldexp (double f, int i)

说明

ldexp() 函数执行 frexp() 的逆运算；整数 i 与浮点数 f 的指数相加，并返回结果。

示例

```
#include <math.h>
#include <stdio.h>
void
main (void)
{
double f;
f = ldexp(1.0, 10);
printf("1.0 * 2^10 = %f\n", f);
}
```

另请参见

frexp()

返回值

返回值为整数 i 与浮点值 f 的指数相加获得的结果。

LDIV

概要

#include <stdlib.h>

ldiv_t ldiv (long number, long denom)

说明

ldiv() 程序将分子除以分母，计算得到商和余数。商的符号与算术商相同。其绝对值是小于算术商绝对值的最大整数。

ldiv() 函数类似于 div() 函数，区别在于参数和返回结构的成员均为 long int 类型。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
ldiv_t lt;
lt = ldiv(1234567, 12345);
printf("Quotient = %ld, remainder = %ld\n", lt.quot, lt.rem);
}
```

另请参见

div()、 uldiv() 和 udiv()

返回值

返回 ldiv_t 类型的结构。

LOCALTIME

概要

#include <time.h>

struct tm * localtime (time_t * t)

说明

localtime() 函数将 t 指向的时间 （自 1970 年 1 月 1 日 00:00:00 以来的秒数）转换为分解的时间，存储在 time.h 中定义的结构中。程序 localtime() 会考虑到全局整型 time_zone 的内容。它应包含本地时区相对于格林威治时间向西的分钟数。在无法预先确定该值的系统上， localtime() 将返回与 gmtime() 相同的结果。

示例

```
#include <stdio.h>
#include <time.h>
char * wday[] = {
"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"
};
void
main (void)
{
time_t clock;
struct tm * tp;
time(&clock);
tp = localtime(&clock);
printf("Today is %s\n", wday[tp->tm_wday]);
}
```

另请参见

ctime()、 asctime() 和 time()

返回值

返回 tm 类型的结构。

注

示例要求用户提供 time() 程序，因为它无法由编译器提供。更多详细信息，请参见 time()。

LOG 和 LOG10

概要

#include <math.h>

double log (double f)

double log10 (double f)

说明

log() 函数返回 f 的自然对数。函数 log10() 返回 f 以 10 为底的对数。

示例

```
#include <math.h>
#include <stdio.h>
void
main (void)
{
double f;
for(f = 1.0 ; f <= 10.0 ; f += 1.0)
printf("log(%1.0f) = %f\n", f, log(f));
}
```

另请参见

exp() 和 pow()

返回值

如果参数为负值，则返回零。

LONGJMP

概要

#include <setjmp.h>

void longjmp (jmp_buf buf, int val)

说明

longjmp() 函数与 setjmp() 配合使用可提供一种非本地 goto 跳转的机制。要使用这项功能，应在外层函数中使用 jmp_buf 参数调用 setjmp()。对 setjmp() 的调用将返回 0。

要返回到该执行层级，可使用相同的 jmp_buf 参数从内层执行中调用 longjmp()。但是，在调用 longjmp() 时，调用 setjmp() 的函数必须仍然处于活动状态。违反该规则会导致错误，因为这将会使用包含无效数据的堆栈。 longjmp() 的 val 参数将为 setjmp() 表面上返回的值。它通常不为零，以区别于真正的 setjmp() 调用。

示例

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>
jmp_buf jb;
void inner (void)
{
longjmp(jb, 5);
}
void main (void)
{
int i;
if(i = setjmp(jb)) {
printf("setjmp returned %d\n" i);
exit(0);
}
printf("setjmp returned 0 - good\n");
printf("calling inner...\n");
inner();
printf("inner returned - bad!\n");
}
```

另请参见

setjmp()

返回值

longjmp() 程序永不返回。

注

在调用 longjmp() 时，调用 setjmp() 的函数必须仍然处于活动状态。违反该规则会导致灾难，因为这将会使用包含无效数据的堆栈。

LTOA

概要

#include <stdlib.h>

char * ltoa (char * buf, long val, int base)

说明

函数 ltoa 将 val 的内容转换为一个字符串，字符串存储到 buf 中。转换将根据 base 中指定的基数执行。假定 buf 引用的缓冲区分配了足够的空间。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
char buf[10];
ltoa(buf, 12345678L, 16);
printf("The buffer holds %s\n", buf);
}
```

另请参见

strtol()、 itoa()、 utoa() 和 ultoa()

返回值

该程序返回写入结果的缓冲区的副本。

MEMCHR

概要

#include <string.h>

void * memchr (const void * block, int val, size_t length)

说明

memchr() 函数类似于 strchr()，只是它不是搜索以空值终止的字符串，而是搜索指定的存储器块，在一定长度内查找某个特定字节。其参数为指向要搜索的存储器的指针、要搜索的字节值，以及块的长度。将返回一个指向块中第一个匹配字节的指针。

示例

```
#include <string.h>
#include <stdio.h>
unsigned int ary[] = {1, 5, 0x6789, 0x23};
void
main (void)
{
char * cp;
cp = memchr(ary, 0x89, sizeof ary);
if(!cp)
printf("Not found\n");
else
printf("Found at offset %u\n", cp - (char *)ary);
}
```

另请参见

strchr()

返回值

返回与参数匹配的第一个字节的指针 （如存在）；否则返回 NULL。

MEMCMP

概要

#include <string.h>

int memcmp (const void * s1, const void * s2, size_t n)

说明

类似于 strncmp()， memcmp() 函数比较两个长度为 n 的存储器块，并返回一个有符号值。与 strncmp() 不同，比较不会在空字符处停止。

示例

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
int buf[10], cow[10], i;
buf[0] = 1;
buf[2] = 4;
cow[0] = 1;
cow[2] = 5;
buf[1] = 3;
cow[1] = 3;
i = memcmp(buf, cow, 3*sizeof(int));
if(i < 0)
printf("Less than\n");
else if(i > 0)
printf("Greater than\n");
else
printf("Equal\n");
}
```

另请参见

strncpy()、 strncmp()、 strchr()、 memset() 和 memchr()

返回值

根据 s1 指向的字符串按排序序列是小于、等于还是大于 s2 指向的字符串，返回 -1、 0 或 1。

MEMCPY

概要

#include <string.h>

void * memcpy (void * d, const void * s, size_t n)

说明

memcpy() 函数会从 s 所指向的位置开始，将 n 字节的存储器内容复制到 d 所指向的存储器块。复制重叠块的结果是未定义的。 memcpy() 函数与 strcpy() 的区别在于它将复制指定的字节数，而不是复制直到空终止符的所有字节。

示例

```
#include <string.h>
#include <stdio.h>
void
main (void)
{
char buf[80];
memset(buf, 0, sizeof buf);
memcpy(buf, "A partial string", 10);
printf("buf = ’%s’\n", buf);
}
```

另请参见

strncpy()、 strncmp()、 strchr() 和 memset()

返回值

memcpy() 程序将返回它的第一个参数。

MEMMOVE

概要

#include <string.h>

void * memmove (void * s1, const void * s2, size_t n)

说明

memmove() 函数类似于函数 memcpy()，只是它可以正确地处理重叠块的复制。即，它将根据情况进行正向或反向复制，以正确地将一个块复制到另一个与之重叠的块。

另请参见

strncpy()、 strncmp()、 strchr() 和 memcpy()

返回值

函数 memmove() 将返回它的第一个参数。

MEMSET

概要

#include <string.h>

void * memset (void * s, int c, size_t n)

说明

memset() 函数将使用字节 c，从 s 所指向的位置开始填充 n 字节的存储器。

示例

```
#include <string.h>
#include <stdio.h>
void
main (void)
{
char abuf[20];
strcpy(abuf, "This is a string";
memset(abuf, ’x’, 5);
printf("buf = ’%s’\n", abuf);
}
```

另请参见

strncpy()、 strncmp()、 strchr()、 memcpy() 和 memchr()

MKTIME

概要

#include <time.h>

time_t mktime (struct tm * tmptr)

说明

mktime() 函数转换由 tm 结构 tmptr 引用的本地日历时间，转换为并返回以自 1970 年 1 月 1 日经过的秒数表示的时间，或如果无法表示该时间，则返回 -1。

示例

```
#include <time.h>
#include <stdio.h>
void
main (void)
{
struct tm birthday;
birthday.tm_year = 83; // the 5th of May 1983
birthday.tm_mon = 5;
birthday.tm_mday = 5;
birthday.tm_hour = birthday.tm_min = birthday.tm_sec = 0;
printf("you were born approximately %ld seconds after the unix
epoch\n",
mktime(&birthday));
}
```

另请参见

ctime() 和 asctime()

返回值

tm 结构中包含的时间表示自 1970 年纪元时间以来的秒数或-1（如果无法表示该时间）。

MODF

概要

#include <math.h>

double modf (double value, double * iptr)

说明

modf() 函数会将参数 value 拆分为整数和小数部分，每个部分的符号都与 value 相同。例如， -3.17 将拆分为整数部分 （-3）和小数部分 （-0.17）。

整数部分以双精度形式存储在 iptr 指向的对象中。

示例

```
#include <math.h>
#include <stdio.h>
void
main (void)
{
double i_val, f_val;
f_val = modf( -3.17, &i_val);
}
```

返回值

value 的有符号小数部分。

NOP

概要

#include <zc.h>

NOP();

说明

此处执行 NOP 指令。它通常用于微调延时或创建断点的句柄。在硬件中的一些敏感序列期间，有时会需要 NOP 指令。

示例

```
#include <zc.h>
void
crude_delay(unsigned char x) {
while(x--){
NOP(); /* Do nothing for 3 cycles */
NOP();
NOP();
}
}
```

POW

概要

#include <math.h>

double pow (double f, double p)

说明

pow() 函数计算第一个参数 f 的 p 次幂。

示例

```
#include <math.h>
#include <stdio.h>
void
main (void)
{
double f;
for(f = 1.0 ; f <= 10.0 ; f += 1.0)
printf("pow(2, %1.0f) = %f\n", f, pow(2, f));
}
```

另请参见

log()、 log10() 和 exp()

返回值

f 的 p 次幂。

PRINTF 和 VPRINTF

概要

#include <stdio.h>

int printf (const char * fmt, ...)

#include <stdio.h>

#include <stdarg.h>

int vprintf (const char * fmt, va_list va_arg)

说明

printf() 函数是一个格式化输出程序，对 stdout 进行操作。它依靠 putch() 函数来确定 stdout 的目标。 putch() 函数必须作为每个项目的一部分进行编写同时编写用于初始化该程序所用的外设的代码。编译器的 sources 目录中可以找到 putch 函数的桩（stub）。

向 USART 发送一个字节的典型 putch 程序可能需要编写为以下形式。

```
void putch(char data) {
while( ! TXIF)
continue;
TXREG = data;
}
```

完成之后，将 putch 的源文件包含到项目中。

在调用时，将向 printf() 程序传递一个格式字符串，后面跟随零个或多个参数的列表。该格式字符串包含一些转换规范，其中每个规范都用于打印参数列表值中的一个参数值。

每个转换规范的形式均为 %fm.nc，其中的百分比符号 % 用于引入转换，后面跟随 0 个或多个标志 f （按任意顺序），后面再跟随可选的宽度规范 m。 n 规范是可选的精度规范 （使用点号引入）， c 是指定转换类型的字母。

这些标志可以包含以下字符。

• 减号 （“-”），指示在字段中对转换值进行向左调整而不是向右调整。当字段宽度大于转换所需的宽度时，将根据规范在左侧或右侧执行空格填充。

• 加号 （“+”），指示总是打印转换值的符号，即使值为正。

• 空格 （“ ”），如果转换结果不包含符号 （因此如果指定了 + 标志，则它不起作用），或如果带符号转换未产生任何字符，则在结果前面附加一个空格。

• 数字零 （“0”），指示使用零而不是空格进行所有填充。如果指定了 “-”标志，则忽略该标志。

• 井号字符 （#） ，指示使用备用格式。下面介绍了备用格式的性质。并不是所有格式都具有备用格式。在这些情况下，井号字符的存在没有任何作用。

如果使用字符 * 代替十进制常量 （例如，在格式 %*d 中），则会从列表中获取一个整型参数来提供该值。

转换的类型包括：

f 浮点 ——m 代表总宽度， n 代表小数点后的位数。如果省略了 n，则它默认设为 6。如果精度为零，则除非指定备用格式，否则将省略小数点。

e 以科学记数法打印相应的参数。否则则类似于 f。

g 使用 e 或 f 格式，即两者中宽度最小而精度最高的格式。如果未指定备用格式，小数

点后尾随的所有 0 都会被删除，如果小数点后没有任何数字，则小数点也会被删除。

o x X u d 整型转换 —— 分别以 8、 16、 16、 10 和 10 为基数。对于 d，转换是有符号的；否则为无符号的。精度值代表要打印的总位数，并可用于强制使用前导零。例如， %8.4x 将在 8 位宽的字段中至少打印 4 个十六进制数字。键字母前带有 l 表示值参数为长整型。字母 X 将使用大写字母 *A-F* 而不是 *a-f* 打印十六进制数字，后者将在使用字母 x 时使用。当指定备用格式时，对于八进制格式将提供前导零，对于十六进制格式将提供前导 0x 或 0X。

s 打印字符串 —— 假定值参数是一个字符。在 m 个字符宽的字段中，最多将打印来自字符串的 n 个字符。

c 假定参数为单个字符，并按字面打印。

用作转换规范的所有其他字符都会被打印。因此， % 将产生一个百分号。

vprintf() 函数类似于 printf()，但它接受可变参数列表，而不是参数的列表。关于可变参数列表的更多信息，请参见 va_start() 的说明。以下给出了一个使用 vprintf() 的示例。

示例

printf("Total = %4d%", 23);

将产生结果 “Total = 23%”

printf("Size is %lx", size);

其中 size 为长整型，打印长度为十六进制值

printf("Name = %.8s", "a1234567890");

将产生结果 “Name = a1234567”

printf("xx%*d", 3, 4);

将产生结果 “xx 4”

```
/* vprintf example */
#include <stdio.h>
int error (char * s, ...)
{
va_list ap;
va_start(ap, s);
printf("Error: ");
vprintf(s, ap);
putchar('\n');
va_end(ap);
}
void    main (void)
{
int i= 3;
error("testing 1 2 %d", i);
}
```

另请参见

sprintf()

返回值

printf() 和 vprintf() 函数返回写入到 stdout 的字符数。

PUTCH

概要

#include <conio.h>

void putch (char c)

说明

putch() 函数以空桩的形式提供，它可以在每个项目需要时完成。该函数通常接受一个字节的数据，并将它发送到与 stdout 关联的外设。

示例

```
#include <conio.h>
char * x = "This is a string";
void
main (void)
{
char * cp;
cp = x;
while(*x)
putch(*x++);
putch('\n');
}
```

另请参见

printf() 和 putchar()

PUTCHAR

概要

#include <stdio.h>

int putchar (int c)

说明

putchar() 函数调用 putch() 来向 stdout 打印一个字符，并在 stdio.h 中定义。

示例

```
#include <stdio.h>
char * x = "This is a string";
void    main (void)
{
char * cp;
cp = x;
while(*x)
putchar(*x++);
putchar('\n');
}
```

另请参见

putc()、 getc()、 freopen() 和 fclose()

返回值

作为参数传递的字符；如果发生错误则返回 EOF。

PUTS

概要

#include <stdio.h>

int puts (const char * s)

说明

puts() 函数将字符串 s 写入 stdout 流，并追加一个换行符。不会复制终止字符串的空字符。

示例

```
#include <stdio.h>
void    main (void)
{
puts("Hello, world!");
}
```

另请参见

fputs()、 gets()、 freopen() 和 fclose()

返回值

发生错误时返回 EOF ； 否则返回零。

QSORT

概要

#include <stdlib.h>

void qsort (void * base, size_t nel, size_t width,

int (*func)(const void *, const void *))

说明

qsort() 函数是快速排序算法的一种实现。它对由 nel 个项目组成的数组进行排序，每个项目的长度为 width 字节，连续位于基址为 base 的存储器中。 func 的参数是 qsort() 用于比较项目的函数的指针。它使用要进行比较的两个项目的指针来调用 func。如果确认第一个项目大于、等于或小于第二个项目， func 应分别返回大于零、等于零或小于零的值。

示例

```
#include <stdio.h>
#include <stdlib.h>
int array[] = {
567, 23, 456, 1024, 17, 567, 66
};
int
sortem (const void * p1, const void * p2)
{
return *(int *)p1 - *(int *)p2;
}
void main (void)
{
register int i;
qsort(aray, sizeof array/sizeof array[0],
sizeof array[0], sortem);
for(i = 0 ; i != sizeof array/sizeof array[0] ; i++)
printf("%d\t", array[i]);
putchar('\n');
}
```

注

函数参数必须为指向类似以下函数类型的指针：

int func (const void *, const void *)

例如，它必须接受两个 const void* 参数，并且必须具有原型声明。

RAND

概要

#include <stdlib.h>

int rand (void)

说明

rand() 函数是一个伪随机数发生器。它返回一个介于范围 0 至 32767 的整数，每次调用时都会以伪随机方式变化。如果从相同的起始点开始，该算法将生成一个确定性的序列。起始点使用 srand() 调用设置。以下给出了使用 time() 函数每次为序列生成不同起始点的示例。

示例

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
void
main (void)
{
time_t toc;
int i;
time(&toc);
srand((int)toc);
for(i = 0 ; i != 10 ; i++)
printf("%d\t", rand());
putchar('\n');
}
```

另请参见

srand()

注

示例要求用户提供 time() 程序，因为它无法由编译器提供。更多详细信息，请参见 time()。

ROUND

概要

#include <math.h>

double round (double x)

说明

round 函数将参数舍入为最接近的整数值，但采用浮点格式。介于整数值之间的值将进行向上舍入。

示例

```
#include <math.h>
void
main (void)
{
double input, rounded;
input = 1234.5678;
rounded = round(input);
}
```

另请参见

trunc()

SETJMP

概要

#include <setjmp.h>

int setjmp (jmp_buf buf)

说明

setjmp() 函数与 longjmp() 配合使用来实现非本地 goto 跳转。更多信息，请参见 longjmp()。

示例

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>
jmp_buf jb;
void
inner (void)
{
longjmp(jb, 5);
}
void
main (void)
{
int i;
if(i = setjmp(jb)) {
printf("setjmp returned %d\n", i);
exit(0);
}
printf("setjmp returned 0 - good\n");
printf("calling inner...\n");
inner();
printf("inner returned - bad!\n");
}
```

另请参见

longjmp()

返回值

setjmp() 函数在实际调用后返回零，如果是调用 longjmp() 后表面返回的，则返回非零。

SIN

概要

#include <math.h>

double sin (double f)

说明

该函数返回其参数的正弦函数。

示例

```
#include <math.h>
#include <stdio.h>
#define C 3.141592/180.0
void main (void)
{
double i;
for(i = 0 ; i <= 180.0 ; i += 10)
printf("sin(%3.0f) = %f\n", i, sin(i*C));
printf("cos(%3.0f) = %f\n", i, cos(i*C));
}
```

另请参见

cos()、 tan()、 asin()、 acos()、 atan() 和 atan2()

返回值

f 的正弦值。

SLEEP

概要

#include <zc.h>

SLEEP();

说明

该宏用于将器件置为低功耗待机模式。

示例

```
#include <zc.h>
extern void init(void);
void
main (void)
{
init(); /* enable peripherals/interrupts */
while(1)
SLEEP(); /* save power while nothing happening */
}
```

SPRINTF

概要

#include <stdio.h>

int sprintf (char * buf, const char * fmt, ...)

说明

sprintf() 函数的工作方式类似于 printf()，只是它不是将转换的输出放在 stdout 流中，而是将字符放在 buf 处的缓冲区中。结果字符串将以空值终止，并返回缓冲区中的字符数。

另请参见

printf()

返回值

该程序返回放入缓冲区中的字符数。

SQRT

概要

#include <math.h>

double sqrt (double f)

说明

函数 sqrt() 使用牛顿逼近法实现平方根程序。

示例

```
#include <math.h>
#include <stdio.h>
void
main (void)
{
double i;
for(i = 0 ; i <= 20.0 ; i += 1.0)
printf("square root of %.1f = %f\n", i, sqrt(i));
}
```

另请参见

exp()

返回值

返回平方根的值。

注

如果参数为负值，则会发生域错误，并且 errno 将被设置为 EDOM。

SRAND

概要

#include <stdlib.h>

void srand (unsigned int seed)

说明

srand() 函数使用给定的 seed 初始化 rand() 访问的随机数发生器。这提供了一种机制来改变 rand() 所产生的伪随机序列的起始点。

示例

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
void
main (void)
{
time_t toc;
int i;
time(&toc);
srand((int)toc);
for(i = 0 ; i != 10 ; i++)
printf("%d\t", rand());
putchar('\n');
}
```

另请参见

rand()

STRCAT

概要

#include <string.h>

char * strcat (char * s1, const char * s2)

说明

该函数将字符串 s2 追加 （连接）到字符串 s1 的末尾。结果将以空值终止。参数 s1 必须指向一个足以存放结果字符串的字符数组。

示例

```
#include <string.h>
#include <stdio.h>
void
main (void)
{
char buffer[256];
char * s1, * s2;
strcpy(buffer, "Start of line");
s1 = buffer;
s2 = "... end of line";
strcat(s1, s2);
printf("Length = %d\n", strlen(buffer));
printf("string = \"%s\"\n", buffer);
}
```

另请参见

strcpy()、 strcmp()、 strncat() 和 strlen()

返回值

返回 s1 的值。

STRCHR 和 STRICHR

概要

#include <string.h>

char * strchr (const char * s, int c)

char * strichr (const char * s, int c)

说明

strchr() 函数搜索字符串 s 中是否存在字符 c。如果找到一个，则返回一个指向该字符的指针，否则返回空值。

strichr() 函数是该函数不区分大小写的版本。

示例

```
#include <strings.h>
#include <stdio.h>
void
main (void)
{
static char temp[] = "Here it is...";
char c = 's';
if(strchr(temp, c))
printf("Character %c was found in string\n", c);
else
printf("No character was found in string");
}
```

另请参见

strrchr()、 strlen() 和 strcmp()

返回值

返回第一个匹配项的指针，或者如果字符串中不存在该字符，则返回 NULL。

注

虽然函数接受对应于字符的整型参数，但它仅使用值的低 8 位。

STRCMP 和 STRICMP

概要

#include <string.h>

int strcmp (const char * s1, const char * s2)

int stricmp (const char * s1, const char * s2)

说明

strcmp() 函数将比较它的两个以空值终止的字符串参数，并返回一个有符号整数来指示 s1 是小于、等于还是大于 s2。使用标准排序序列进行比较，即 ASCII 字符集的排序序列。

stricmp() 函数是该函数不区分大小写的版本。

示例

```
#include <string.h>
#include <stdio.h>
void
main (void)
{
int i;
if((i = strcmp("ABC", "ABc")) < 0)
printf("ABC is less than ABc\n");
else if(i > 0)
printf("ABC is greater than ABc\n");
else
printf("ABC is equal to ABc\n");
}
```

另请参见

strlen()、 strncmp()、 strcpy() 和 strcat()

返回值

小于、等于或大于零的有符号整数。

注

其他 C 实现可能使用不同的排序序列；返回值为负值、零或正值；即，不要明确地检查返回值是否为 -1 或 1。

STRCPY

概要

#include <string.h>

char * strcpy (char * s1, const char * s2)

说明

该函数会将以空值终止的字符串 s2 复制到 s1 指向的字符数组。目标数组必须足够大，足以容纳包括空终止符在内的整个字符串。

示例

```
#include <string.h>
#include <stdio.h>
void
main (void)
{
char buffer[256];
char * s1, * s2;
strcpy(buffer, "Start of line");
s1 = buffer;
s2 = "... end of line";
strcat(s1, s2);
printf("Length = %d\n", strlen(buffer));
printf("string = \"%s\"\n", buffer);
}
```

另请参见

strncpy()、 strlen()、 strcat() 和 strlen()

返回值

返回目标缓冲区 s1。

STRCSPN

概要

#include <string.h>

size_t strcspn (const char * s1, const char * s2)

说明

strcspn() 函数返回在 s1 指向的字符串中，不包含来自 s2 所指向字符串的字符的起始部分的长度。

示例

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
static char set[] = "xyz";
printf("%d\n", strcspn("abcdevwxyz", set));
printf("%d\n", strcspn("xxxbcadefs", set));
printf("%d\n", strcspn("1234567890", set));
}
```

另请参见

strspn()

返回值

返回该部分的长度。

STRLEN

概要

#include <string.h>

size_t strlen (const char * s)

说明

strlen() 函数返回字符串 s 中的字符数 （不包括空终止符）。

示例

```
#include <string.h>
#include <stdio.h>
void
main (void)
{
char buffer[256];
char * s1, * s2;
strcpy(buffer, "Start of line");
s1 = buffer;
s2 = "... end of line";
strcat(s1, s2);
printf("Length = %d\n", strlen(buffer));
printf("string = \"%s\"\n", buffer);
}
```

返回值

空终止符前的字符数。

STRNCAT

概要

#include <string.h>

char * strncat (char * s1, const char * s2, size_t n)

说明

该函数会将字符串 s2 追加 （连接）到字符串 s1 的末尾。最多复制 n 个字符，结果将以空值终止。 s1 必须指向一个足以容纳结果字符串的字符数组。

示例

```
#include <string.h>
#include <stdio.h>
void
main (void)
{
char buffer[256];
char * s1, * s2;
strcpy(buffer, "Start of line");
s1 = buffer;
s2 = "... end of line";
strncat(s1, s2, 5);
printf("Length = %d\n", strlen(buffer));
printf("string = \"%s\"\n", buffer);
}
```

另请参见

strcpy()、 strcmp()、 strcat() 和 strlen()

返回值

返回 s1 的值。

STRNCMP 和 STRNICMP

概要

#include <string.h>

int strncmp (const char * s1, const char * s2, size_t n)

int strnicmp (const char * s1, const char * s2, size_t n)

说明

strncmp() 函数将比较它的两个以空值终止的字符串参数（最多 n 个字符），并返回一个有符号整数来指示 s1 是小于、等于还是大于 s2。使用标准排序序列进行比较，即 ASCII 字符集的排序序列。

strnicmp() 函数是该函数不区分大小写的版本。

示例

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
int i;
i = strncmp("abcxyz", "abcxyz", 6);
if(i == 0)
printf("The strings are equal\n");
else if(i > 0)
printf("String 2 less than string 1\n");
else
printf("String 2 is greater than string 1\n");
}
```

另请参见

strlen()、 strcmp()、 strcpy() 和 strcat()

返回值

小于、等于或大于零的有符号整数。

注

其他 C 实现可能使用不同的排序序列；返回值为负值、零或正值；即，不要明确地测试**返回值**是否为 -1 或 1。

STRNCPY

概要

#include <string.h>

char * strncpy (char * s1, const char * s2, size_t n)

说明

该函数会将以空值终止的字符串 s2 复制到 s1 指向的字符数组。最多复制 n 个字符。如果字符串 s2 的长度大于 n，目标字符串将不是以空值终止。目标数组必须足够大，足以容纳包括空终止符在内的整个字符串。

示例

```
#include <string.h>
#include <stdio.h>
void
main (void)
{
char buffer[256];
char * s1, * s2;
strncpy(buffer, "Start of line", 6);
s1 = buffer;
s2 = "... end of line";
strcat(s1, s2);
printf("Length = %d\n", strlen(buffer));
printf("string = \"%s\"\n", buffer);
}
```

另请参见

strcpy()、 strcat()、 strlen() 和 strcmp()

返回值

返回目标缓冲区 s1。

STRPBRK

概要

#include <string.h>

char * strpbrk (const char * s1, const char * s2)

说明

strpbrk() 函数返回一个指针 （指向字符串 s1 中第一次出现字符串 s2 中任意字符的位置）或空值 （如果 s1 中不存在 s2 中的任何字符）。

示例

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
char * str = "This is a string.";
while(str != NULL) {
printf("%s\n", str);
str = strpbrk(str+1, "aeiou");
}
}
```

返回值

指向第一个匹配字符的指针，或如果未找到任何字符，则返回 NULL。

STRRCHR 和 STRRICHR

概要

#include <string.h>

char * strrchr (char * s, int c)

char * strrichr (char * s, int c)

说明

strrchr() 函数类似于 strchr() 函数，但它从字符串末尾处开始搜索，而不是从起始处开始；即，它查找以空值终止的字符串 s 中的最后一处字符 c。如果成功，它将返回指向匹配位置的指针，否则将返回 NULL。

strrichr() 函数是该函数不区分大小写的版本。

示例

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
char * str = "This is a string.";
while(str != NULL) {
printf("%s\n", str);
str = strrchr(str+1, ’s’);
}
}
```

另请参见

strchr()、 strlen()、 strcmp()、 strcpy() 和 strcat()

返回值

指向字符的指针，如果未找到则返回 NULL。

STRSPN

概要

#include <string.h>

size_t strspn (const char * s1, const char * s2)

说明

strspn() 函数返回在 s1 指向的字符串中，完全包含来自 s2 所指向字符串的字符的起始部分的长度。

示例

```
#include <stdio.h>
#include <string.h>
void main (void)
{
printf("%d\n", strspn("This is a string", "This"));
printf("%d\n", strspn("This is a string", "this"));
}
```

另请参见

strcspn()

返回值

该部分的长度。

STRSTR 和 STRISTR

概要

#include <string.h>

char * strstr (const char * s1, const char * s2)

char * stristr (const char * s1, const char * s2)

说明

strstr() 函数在 s1 所指向的字符串中查找第一次出现 s2 所指向字符串的位置。

stristr() 程序是该函数不区分大小写的版本。

示例

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
printf("%d\n", strstr("This is a string", "str"));
}
```

返回值

指向所找到字符串的指针，或如果未找到该字符串，则返回空值。

STRSPN

概要

#include <string.h>

size_t strspn (const char * s1, const char * s2)

说明

strspn() 函数返回在 s1 指向的字符串中，完全包含来自 s2 所指向字符串的字符的起始部分的长度。

示例

```
#include <stdio.h>
#include <string.h>
void    main (void)
{
printf("%d\n", strspn("This is a string", "This"));
printf("%d\n", strspn("This is a string", "this"));
}
```

另请参见

strcspn()

返回值

该部分的长度。

STRSTR 和 STRISTR

概要

#include <string.h>

char * strstr (const char * s1, const char * s2)

char * stristr (const char * s1, const char * s2)

说明

strstr() 函数在 s1 所指向的字符串中查找第一次出现 s2 所指向字符串的位置。

stristr() 程序是该函数不区分大小写的版本。

示例

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
printf("%d\n", strstr("This is a string", "str"));
}
```

返回值

指向所找到字符串的指针，或如果未找到该字符串，则返回空值。

STRTOL

概要

#include <stdlib.h>

double strtol (const char * s, const char ** res, int base)

说明

解析字符串 s，将它转换为长整型。该函数会在输入中查找第一个由所期望形式的字符组成的子字符串，并在略过前导空格字符之后对它进行转换。输入的基数将通过 base 确定。如果它为零，则基数默认设为 10。如果 res 不为 NULL，它将被设为指向转换后的子字符串的第一个字符。

示例

```
#include <stdio.h>
#include <strlib.h>
void
main (void)
{
char buf[] = "0X299 0x792";
char * end;
long in1, in2;
in1 = strtol(buf, &end, 16);
in2 = strtol(end, NULL, 16);
printf("in (decimal): %ld, %ld\n", in1, in2);
}
```

另请参见

strtod()

返回值

返回表示输入字符串使用指定基数转换后的长整型值。

STRTOK

概要

#include <string.h>

char * strtok (char * s1, const char * s2)

说明

对 strtok() 进行多次调用会将字符串 s1（它包含由零个或多个文本标记组成的序列，这些标记使用来自分隔符字符串 s2 的一个或多个字符分隔）拆分为单独的标记。

第一次调用必须提供字符串 s1。该调用会返回指向第一个标记的第一个字符的指针，或如果未找到标记，则返回 NULL。标记间的分隔符将被一个空字符覆盖，它将终止当前标记。

对于 strtok() 的后续调用， s1 应设置为 NULL。这些调用会从所找到的上一个标记的末尾开始搜索，并且再次返回指向下一个标记的第一个字符的指针，或如果未找到更多标记，则返回 NULL。

示例

```
#include <stdio.h>
#include <string.h>
void
main (void)
{
char * ptr;
char buf[] = "This is a string of words.";
char * sep_tok = ",?! "
ptr = strtok(buf, sep_tok);
while(ptr != NULL) {
printf("%s\n", ptr);
ptr = strtok(NULL, sep_tok);
}
}
```

返回值

返回指向标记第一个字符的指针，或如果未找到标记，则返回空值。

注

分隔符字符串 s2 在调用之间可能会不同。

TAN

概要

#include <math.h>

double tan (double f)

说明

tan() 函数计算 f 的正切值。

示例

```
#include <math.h>
#include <stdio.h>
#define C 3.141592/180.0
void
main (void)
{
double i;
for(i = 0 ; i <= 180.0 ; i += 10)
printf("tan(%3.0f) = %f\n", i, tan(i*C));
}
```

另请参见

sin()、 cos()、 asin()、 acos()、 atan() 和 atan2()

返回值

f 的正切值。

TIME

概要

#include <time.h>

time_t time (time_t * t)

说明

该函数没有提供，因为它依赖于提供当前时间的目标系统。该函数由用户实现。在实现时，该函数应返回以自 1970 年 1 月 1 日 00:00:00 以来的秒数表示的当前时间。如果参数 t 不等于 NULL，同一个值将存储到 t 所指向的对象。

示例

```
#include <stdio.h>
#include <time.h>
void
main (void)
{
time_t clock;
time(&clock);
printf("%s", ctime(&clock));
}
```

另请参见

ctime()、 gmtime()、 localtime() 和 asctime()

返回值

在实现时，该程序将返回以自 1970 年 1 月 1 日 00:00:00 以来的当前时间（单位为秒）。

注

time() 程序没有提供，如果需要，用户必须按照上述规范实现该程序。

TOLOWER、 TOUPPER 和 TOASCII

概要

#include <ctype.h>

char toupper (int c)

char tolower (int c)

char toascii (int c)

说明

toupper() 函数将其小写形式的字母参数转换为大写形式， tolower() 程序执行反向转换， toascii() 宏返回确保处于范围 0-0177 中的结果。如果其参数不是字母字符，函数 toupper() 和 tolower() 将返回其参数。

示例

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
void
main (void)
{
char * array1 = "aBcDE";
int i;
for(i=0;i < strlen(array1); ++i) {
printf("%c", tolower(array1[i]));
}
printf("n");
}
```

另请参见

islower()、 isupper() 和 isascii() 等

TRUNC

概要

#include <math.h>

double trunc (double x)

说明

trunc 函数会采用浮点格式将参数舍入为最接近的整数值，即不大于该参数的幅值。

示例

```
#include <math.h>
void
main (void)
{
double input, rounded;
input = 1234.5678;
rounded = trunc(input);
}
```

另请参见

round()

UDIV

概要

#include <stdlib.h>

int udiv (unsigned num, unsigned denom)

说明

udiv() 函数计算 number 和 denom 相除得到的商和余数，并将结果存储到返回的 udiv_t 结构中。

示例

```
#include <stdlib.h>
void
main (void)
{
udiv_t result;
unsigned num = 1234, den = 7;
result = udiv(num, den);
}
```

另请参见

uldiv()、 div() 和 ldiv()

返回值

以 udiv_t 结构的形式返回商和余数。

ULDIV

概要

#include <stdlib.h>

int uldiv (unsigned long num, unsigned long denom)

说明

uldiv() 函数计算 number 和 denom 相除得到的商和余数，并将结果存储到返回的 uldiv_t 结构中。

示例

```
#include <stdlib.h>
void
main (void)
{
uldiv_t result;
unsigned long num = 1234, den = 7;
result = uldiv(num, den);
}
```

另请参见

ldiv()、 udiv() 和 div()

返回值

以 uldiv_t 结构的形式返回商和余数。

UTOA

概要

#include <stdlib.h>

char * utoa (char * buf, unsigned val, int base)

说明

函数 utoa() 将 val 的无符号内容转换为一个字符串，字符串存储到 buf 中。转换将根据 base 中指定的基数执行。假定 buf 引用的缓冲区分配了足够的空间。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
char buf[10];
utoa(buf, 1234, 16);
printf("The buffer holds %s\n", buf);
}
```

另请参见

strtol()、 itoa()、 ltoa() 和 ultoa()

返回值

该程序返回写入结果的缓冲区的副本。

VA_START、 VA_ARG 和 VA_END

概要

#include <stdarg.h>

void va_start (va_list ap, *parmN*)

type va_arg (ap, *type*)

void va_end (va_list ap)

说明

这些宏用于通过一种可移植的方式访问函数的参数，这些参数在原型中使用省略号（...）表示，而提供给函数的参数的类型和数量在编译时是未知的。

函数最右侧的参数 （显示为 *parmN*）在这些宏中发挥着重要的作用，因为它是访问更多参数的起始点。在接受可变参数数量的函数中，应声明 va_list 类型的变量，然后用该变量和 *parmN* 的名称调用宏 va_start()。 这将初始化该变量， 以便随后调用宏 va_arg() 来访问后续的参数。

每次调用 va_arg() 都需要两个参数；先前定义的变量，以及下一个参数的预期类型的类型名称。请注意，通过这种方式访问的所有参数都会被默认约定拓宽为 int、 unsigned int 或 double 类型。例如，如果传递了某个字符参数，则应该通过 va_arg(ap, int) 访问它，因为 char 类型会被拓宽为 int 类型。

下面给出了一个接受一个整型参数，并跟随一些其他参数的函数示例。在该示例中，函数要求后续参数为 char 类型，但请注意编译器并不知道这一点，编程人员需要负责确保提供正确的参数。

示例

```
#include <stdio.h>
#include <stdarg.h>
void
pf (int a, ...)
{
va_list ap;
va_start(ap, a);
while(a--)
puts(va_arg(ap, char *));
va_end(ap);
}
void
main (void)
{
pf(3, "Line 1", "Line 2", "Line 3");
}
```

XTOI

概要

#include <stdlib.h>

unsigned xtoi (const char * s)

说明

xtoi() 函数会扫描传递给它的字符串，略过前导空格并读取可选的符号，然后将十六进制数字的 ASCII 表示形式转换为整型。

示例

```
#include <stdlib.h>
#include <stdio.h>
void
main (void)
{
char buf[80];
int i;
gets(buf);
i = xtoi(buf);
printf("Read %s: converted to %x\n", buf, i);
}
```

另请参见

atoi()

返回值

无符号整型。如果未在字符串中找到任何数字，则返回 0。

__DELAY_MS 、 __DELAY_US 、 __DELAYWDT_US 和 __DELAYWDT_MS

概要

__delay_ms(x) // request a delay in milliseconds
__delay_us(x) // request a delay in microseconds
__delaywdt_ms(x) // request a delay in milliseconds
__delaywdt_us(x) // request a delay in microseconds

说明

通常，以基于时间的方式而不是周期计数的方式请求产生延时会比较方便。提供了宏 __delay_ms(*x*) 和 __delay_us(*x*) 来满足这种需求。这些宏会将基于时间的请求转换为可以用于 _delay(*n*) 的指令周期数。为了实现这一点，这些宏需要预先定义预处理器符号 _XTAL_FREQ，该符号用于指示系统频率。该符号应等于系统使用的振荡器频率 （以赫兹为单位）。
可以使用这些函数的替代 WDT 形式，替代形式使用 CLRWDT 指令作为延时代码的一部分。请参见 _delay 函数。
宏参数必须为常量表达式。如果使用这些宏时未定义振荡器频率符号、所请求的延时周期太大或延时周期不是常量，将会产生错误。

另请参见

_delay()

_DELAY() 和 _DELAYWDT

概要

#include <zc.h>
void _delay(unsigned long cycles);
void _delaywdt(unsigned long cycles);

说明

这是一个由代码生成器进行扩展的内联函数。在调用时，该程序将扩展为内联汇编延时
序列。序列将包含根据参数延迟指定指令周期数的代码。参数必须为常量表达式。
_delay 内联函数可以使用循环和 NOP 指令来实现延时。_delaywdt 内联函数执行相同的任务，
但可以使用 CLRWDT 指令以及循环来实现指定 的延时。
如果所请求的延时周期不是常量表达式或太大，将会产生错误。要实现非常大的延时，可以多次调用该函数。

示例

```
#include <xc.h>
void
main(void)
{
control |= 0x80;
_delay(10); // delay for 10 cycles
control &= 0x7F;
}
```

另请参见

_delay3()、 __delay_us() 和 __delay_ms()

_DELAY3()

概要

#include <zc.h>

void _delay3(unsigned char cycles);

说明

这是一个由代码生成器进行扩展的内联函数。在调用时，该程序将扩展为内联汇编延时序列。序列将包含根据参数延迟指定周期数 3 倍时间的代码。参数可以为字节长度的常量或变量。

示例

```
#include <xc.h>
void
main (void)
{
control |= 0x80;
_delay3(10); // delay for 30 cycles
control &= 0x7F;
}
```

另请参见

_delay

附 关于延时函数

库延时函数

新核的 IDE 提供了内联库延时函数：

_delay3(unsigned char cycles);

_delay(unsigned long cycles);

_delaywdt(unsigned long cycles);

__delay_us(x)

__delaywdt_us(x)

__delay_ms(x)

__delaywdt_ms(x)

- **_delay3(unsigned char cycles);**

概要

```
#include <zc.h>
void _delay3(unsigned char cycles);
```

说明

这是一个由代码生成器进行扩展的内联函数。在调用时，该程序将扩展为内联汇编延时序列。序列将包含根据参数延迟指定周期数 3 倍时间的代码。参数可以为字节长度的常量或变量。

示例

```
#include <zc.h>
void main (void)
{
OEB = 0XFF;
IOBOR = 0XFF;
_delay3(10); // delay for 30 cycles
IOBOR = 0X00;
}
```

- **_delay(unsigned long cycles);**
  _delaywdt(unsigned long cycles);

概要

```
#include <zc.h>
void _delay(unsigned long cycles);
void _delaywdt(unsigned long cycles);
```

说明

这是一个由代码生成器进行扩展的内联函数。在调用时，该程序将扩展为内联汇编延时序列。序列将包含根据参数延迟指定指令周期数的代码。参数必须为常量表达式。_delay 内联函数可以使用循环和 NOP 指令来实现延时。_delaywdt 内联函数执行相同的任务，但可以使用

CLRWDT 指令以及循环来实现指定的延时。
如果所请求的延时周期不是常量表达式或太大，将会产生错误。要实现非常大的延时，可以多次调用该函数。

示例

```
#include <zc.h>
void main(void)
{
OEB = 0XFF;
IOBOR = 0XFF;
_delay(10); // delay for 10 cycles
IOBOR = 0X00;
}
```

- __delay_us(x)
  __delaywdt_us(x)

  __delay_ms(x)
  __delaywdt_ms(x)

概要

__delay_ms(x)
__delay_us(x)
__delaywdt_ms(x)
__delaywdt_us(x)

说明

通常，以基于时间的方式而不是周期计数的方式请求产生延时会比较方便。提供了宏 __delay_ms(*x*) 和 __delay_us(*x*) 来满足这种需求。这些宏会将基于时间的请求转换为可以用于 _delay(*n*) 的指令周期数。为了实现这一点，这些宏需要预先定义预处理器符号 _Fcpu，该符号用于指示指令时钟频率。该符号应等于 CPU 时钟频率 （以赫兹为单位）。宏参数必须为常量表达式。如果使用这些宏时未定义振荡器频率符号、所请求的延时周期太大或延时周期不是常量，将会产生错误。

注意：上述函数均为内联函数；只有_delay3（）函数的参数可为变量