第30章 实 用 函 数

标准函数库定义了一些提供各种常用服务的实用函数。这些函数包括各种转换、可变长参数处理、排序与查找以及随机数生成等。本章介绍的许多函数需要用到头文件<cstdlib>(C程序必须使用头文件stdlib.h)。在这个头文件中定义了div_t和ldiv_t，它们分别是div( )和ldiv( )返回的值的类型。该头文件中还定义了类型size_t，它是sizeof返回的无符号值。此外，头文件中还定义了下面的宏：

宏	含义
MB_CUR_MAX	多字节字符的最大长度（以字节为单位）
NULL	一个空指针
RAND_MAX	可以被rand( )函数返回的最大值
EXIT_FAILURE	如果程序非正常终止，返回给调用过程的值
EXIT_SUCCESS	如果程序正常终止，返回给调用过程的值

如果一个函数需要一个与<cstdlib>不同的头文件，该函数描述将对其进行讨论。

30.1 abort函数

```
#include <cstdlib>
void abort(void);
```

abort( )函数可立即非正常地终止一个程序。一般来说，这个函数不刷新任何文件。在支持这个函数的环境中，abort( )将把一个实现定义的值返回给调用过程（通常是操作系统）以表明出现了故障。

与abort( )相关的函数有exit( )和atexit( )。

30.2 abs函数

```
#include <cstdlib>
int abs(int num);
long abs(long num);
double abs(double num);
```

abs( )函数返回num的绝对值。abs( )的long型版本与labs( )相同，abs( )的double型版本与fabs( )相同。

与abs( )相关的函数有labs( )。

30.3 assert函数

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

assert( )宏在头文件<cassert>中定义，如果表达式exp等于0，它将把错误信息写到stderr并终止程序的执行；否则，assert( )什么事情也不做。虽然具体输出是由实现定义的，但许多编译器使用与下面相似的信息：

Assertion failed: *<expression>*, file *<file>*, line *<linenum>*

assert( )宏通常用于帮助验证一个程序是否正常运行，只有当没有发生错误时，表达式才为true。

因为如果定义了NDEBUG，assert( )就会被忽略，所以调试程序时没有必要从源代码中删除assert( )语句。

与assert( )相关的函数有abort( )。

30.4 atexit函数

```
#include <cstdlib>
int atexit(void (*func)(void));
```

程序正常终止时，atexit( )函数调用func所指的函数。如果该函数成功地注册为一个终止函数，atexit( )函数返回0；否则返回一个非0值。

至少可以建立32个终止函数，这些函数将按照建立顺序的逆序被调用。

与atexit( )相关的函数有exit( )和abort( )。

30.5 atof函数

```
#include <cstdlib>
double atof(const char   * str);
```

atof( )函数把str所指的字符串转换为一个double值。该字符串必须包含一个有效的浮点数。否则，返回值将是不确定的。

该浮点数可以以任何不能作为有效浮点数的一部分的字符结束，这些字符包括空格、标点符号（句号除外）和除E或e之外的其他字符。这意味着如果用“100.00HELLO”调用atof( )，函数将返回值100.00。

与atof( )相关的函数有atoi( )和atol( )。

30.6 atoi函数

```
#include <cstdlib>
int atoi(const char *str);
```

atoi( )函数把str所指的字符串转换为一个int值。该字符串必须包含一个有效的整型数。否则，返回值将是不确定的。然而，大多数实现将返回0。

该整型数可以以任何不能作为有效整型数的一部分的字符结束，这些字符包括空格、标点符号和字符。这意味着如果用“123.23”调用atoi( )，函数将返回整型值123并忽略“.23”。

与atoi( )相关的函数有atof( )和atol( )。

30.7 atol函数

```
#include <cstdlib>
long atol(const char *str);
```

atol( )函数把str所指的字符串转换为一个long型值。该字符串必须包含一个有效的长整型数。否则，返回值将是不确定的。然而，大多数实现将返回0。

该长整型数可以以任何不能作为有效整型数的一部分的字符结束，这些字符包括空格、标点符号和字符。这意味着如果用“123.23”调用atol( )，函数将返回长整型值123L并忽略“.23”。

与atol( )相关的函数有atof( )和atoi( )。

30.8 bsearch函数

```
#include <cstdlib>
void *bsearch(const void *key, const void *buf,
size_t num, size_t size,
int ( *compare)(const void *, const void *));
```

bsearch( )函数对buf所指的排好序的数组进行二分查找并返回一个指向第一个与key所指的关键字相匹配的成员的指针。该数组的元素个数由num指定，某个元素的大小（以字节为单位）由size说明。

compare所指的函数用于把数组中的某个元素与关键字相比较。compare函数必须遵循以下形式：

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

该函数必须返回下列表格中描述的值：

比较	返回值
arg1小于arg2	小于0
arg1等于arg2	0
arg1大于arg2	大于0

数组必须按照升序排序，即最低的地址包含最小的元素。

如果数组不包含关键字，函数将返回一个空指针。

与bsearch( )相关的函数是qsort( )。

30.9 div函数

```
#include <cstdlib>
div_t div(int numerator, int denominator);
ldiv_t div(long numerator, long denominator);
```

int版本的div( )函数以一个div_t类型的结构返回分子/分母的商和余数；long版本的div( )以一个ldiv_t类型的结构返回分子/分母的商和余数。long版本的div( )与ldiv( )函数具有同样的功能。

结构类型的div_t至少包含以下两个域：

```
int quot; /* quotient */
int rem; /* remainder */
```

结构类型的ldiv_t至少包含以下两个域：

```
long quot; /* quotient */
long rem; /* remainder */
```

与div( )相关的函数是ldiv( )。

30.10 exit函数

```
#include <cstdlib>
void exit(int exit_code);
```

exit( )函数可以立即使程序正常终止。如果环境支持这个函数，exit_code的值则被传递给调用过程（通常是操作系统）。按照约定，如果exit_code的值为0或者为EXIT_SUCCESS，则表明程序正常结束；如果该值为非0或者为EXIT_FAILURE，则表明出现了一个实现定义的错误。

与exit( )相关的函数有atexit( )和abort( )。

30.11 getenv函数

```
#include <cstdlib>
char *getenv(const char *name);
```

getenv( )函数返回一个指向环境信息的指针，该环境信息与实现定义的环境信息表中name所指的字符串相关联。函数返回的字符串永远不能被程序改变。

一个程序的环境可以包括诸如路径名和联机设备等内容，这个数据的具体特性由实现定义。详细情况可查阅所用编译器的文档。

如果用一个与任何环境数据都不匹配的参数调用getenv( )，函数将返回一个空指针。

与getenv( )相关的函数是system( )。

30.12 labs函数

```
#include <cstdlib>
long labs(long num);
```

labs( )函数返回num的绝对值。

与labs( )相关的函数是abs( )。

30.13 ldiv函数

```
#include <cstdlib>
ldiv_t ldiv(long numerator, long denominator);
```

ldiv( )函数返回分子/分母的商和余数。

结构类型ldiv_t至少还有下面两个域：

```
long quot; /* quotient */
long rem; /* remainder */
```

与ldiv( )相关的函数是div( )。

30.14 longjmp函数

```
#include <csetjmp>
void longjmp(jmp_buf envbuf, int status);
```

longjmp( )函数使程序从最后一次调用setjmp( )的地方重新开始执行。这两个函数提供了一种在函数之间跳转的方法。注意，该函数需要头文件<csetjmp>。

longjmp( )函数将堆栈重新设置为envbuf中描述的状态，该参数必须事先通过调用setjmp( )设置。此举使程序在调用setjmp( )后面的语句处重新开始执行。也就是说，它可以使计算机认为从未离开过称为setjmp( )的函数（形象地说，longjmp( )不执行正常的函数返回过程，而是改变跨越时间和存储器空间的路径，从而到达程序中的前一点）。

缓冲区evnbuf的类型是jmp_buf，这个类型在头文件<csetjmp>中定义。该缓冲区必须通过调用setjmp( )设置，而setjmp( )要在调用longjmp( )之前调用。status的值将成为setjmp( )的返回值，通过这个返回值可以确定该跳转来自何处。惟一不能允许出现的值是0。

到目前为止，longjmp( )最常见的用法是当出现错误时从深层嵌套的例程中返回。

与longjmp( )相关的函数是setjmp( )。

30.15 mblen函数

```
#include <cstdlib>
int mblen(const char *str, size_t size);
```

mblen( )函数返回str所指的多字节字符的长度（以字节为单位）。该函数只检查前size个字符。如果出现错误，则返回-1。如果str为空并且多字节字符具有与状态相关的编码，mblen( )将返回非0值；否则，返回0。

与mblen( )相关的函数有mbtowc( )和wctomb( )。

30.16 mbstowcs函数

```
#include <cstdlib>
size_t mbstowcs(wchar_t *out, const char *in, size_t size);
```

mbstowcs( )函数把in所指的多字节字符串转换为宽字符串并将结果放入out所指的数组。该函数只将size个字节存入out。

mbstowcs( )函数返回被转换的多字节字符的个数。如果发生错误，函数返回-1。

与mbstowcs( )相关的函数有wcstombs( )和mbtowc( )。

30.17 mbtowc函数

```
#include <cstdlib>
int mbtowc(wchar_t *out, const char *in, size_t size);
```

mbtowc( )函数把in所指的数组中的多字节字符转换为等价的宽字符并将结果放入out所指的对象中。该函数只检查 size 个字符。

该函数将返回放入out中的字节数。如果发生错误，则返回 -1。如果in为空并且多字节字符具有与状态相关的编码，mbtowc( )将返回非0值；否则，返回0。

与 mbtowc( )相关的函数有 mblen( )和 wctomb( )。

30.18 qsort 函数

```
#include <cstdlib>
void qsort(void *buf, size_t num, size_t size,
           int (*compare) (const void *, const void *));
```

qsort( )函数利用 Quicksort（由 C.A.R. Hoare 开发）对 buf 所指的数组进行排序。Quicksort 是具有多种用途的最佳排序算法。数组中的元素个数由 num 指定，每个元素的大小（以字节为单位）由 size 描述。

compare 所指的函数用于把数组的某个元素和关键字进行比较。compare 函数的形式如下所示：

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

该函数必须返回下面描述的值：

比较	返回值
arg1 小于 arg2	小于 0
arg1 等于 arg2	0
arg1 大于 arg2	大于 0

数组被按照升序进行排序，即最低的地址包含最小的元素。

与 qsort( )相关的函数有 bsearch( )。

30.19 raise 函数

```
#include <csignal>
int raise(int signal);
```

raise( )函数把 signal 指定的信号发送到执行程序。如果发送成功，该函数返回0；否则，返回非0值。该函数需要用到头文件<csignal>。

下面是标准 C++ 定义的信号。当然，你使用的编译器还可以提供一些附加信号。

宏	含义
SIGABRT	终止错误
SIGFPE	浮点错误
SIGILL	错误指令
SIGINT	用户按了 CTRL-C
SIGSEGV	非法的存储器访问
SIGTERM	终止程序

与 raise( )相关的函数有 signal( )。

30.20 rand函数

```
#include <cstdlib>
int rand(void);
```

rand( )函数生成一个伪随机数的序列。每次调用该函数时都将返回一个范围在0到RAND_MAX之间的整数。

与rand( )相关的函数有srand( )。

30.21 setjmp函数

```
#include <csetjmp>
int setjmp(jmp_buf envbuf);
```

setjmp( )函数将系统堆栈中的内容保存到缓冲区envbuf中以备以后为longjmp( )所用。该函数需要用到头文件<csetjmp>。

setjmp( )函数在调用时返回0。然而，当执行longjmp( )时，它将参数传送给setjmp( )，在调用longjmp( )之后，该值（总是非0值）看起来似乎是setjmp( )的值。参见longjmp以获得其他信息。

与setjmp( )相关的函数有longjmp( )。

30.22 signal函数

```
#include <csignal>
void (*signal(int signal, void ( *func)(int))) (int);
```

signal( )函数把func所指的函数注册为用于signal指定的信号处理器。也就是说，当程序接收到signal时，就会调用func所指的函数。

func的值可以是信号处理器的地址，也可以是下面列出的宏中的一个。这些宏在<csignal>中定义。

宏	含义
SIG_DFL	采用默认信号处理
SIG_IGN	忽略信号

如果采用某个函数地址，那么当相应的信号被接收时将执行特定的处理程序。

如果运行成功，signal( )将返回先前为特定信号定义的函数的地址；如果出现错误，则返回SIG_ERR（在<csignal>中定义）。

与signal( )相关的函数是raise( )。

30.23 srand函数

```
#include <cstdlib>
void srand(unsigned seed);
```

srand( )函数用于为rand( )（rand( )函数返回一些伪随机数）生成的序列设置一个起始点。

srand( )一般用于运行多个程序，这些程序通过指定不同的起始点来使用不同的伪随机数序

列。相反，你也可以使用srand( )一次又一次地生成同样的伪随机序列，其方法是通过在获得每一个序列之前用相同的“种子”调用该函数。

与srand( )相关的函数是rand( )。

30.24 strtod函数

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

strtod( )函数把start所指的字符串中存储的用字符串表示的数字转换为double型并返回转换结果。

strtod( )函数的工作原理如下：首先，函数将去掉字符串中的所有空白符。然后，读取每一个包含数字的字符。只要字符不是浮点数的一部分，就会使该过程停止。这样的字符包括空白符、标点符号（句点除外）和除E或e之外的所有字符。最后，如果有余数，end则被设置为指向初始字符串的余项。也就是说，如果用“100.00 Pliers”调用strtod( )，函数的返回值为100.00，end将指向Pliers前面的空白符。

如果没有发生转换，函数将返回0。如果发生溢出，strtod( )将返回HUGE_VAL或-HUGE_VAL（分别表示上溢和下溢）并把全局变量errno设置为ERANGE以说明出现了一个范围错误。如果是下溢，函数将返回0，全局变量errno被设置为ERANGE。

与strtod( )相关的函数是atof( )。

30.25 strtol函数

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

strtol( )函数把start所指的字符串中存储的用字符串表示的数字转换为long型并返回转换结果。数字的基由radix决定。如果radix是0，基数由调节常数说明的规则确定；如果radix不为0，则必须在2～36的范围内。

strtol( )函数的工作原理如下：首先，函数将去掉字符串中的所有空白符。然后，读取每一个包含数字的字符。只要字符不是长整型数的一部分，就会使该过程停止。这样的字符包括空白符、标点符号和字符。最后，如果有余数，end则被设置为指向初始字符串的余项。也就是说，如果用“100 Pliers”调用strtol( )，函数的返回值为100L，end将指向Pliers前面的空白符。

如果结果不能表示为长整型数，strtol( )将返回LONG_MAX或LONG_MIN，全局变量errno被设置为ERANGE，表示出现了一个范围错误。如果没有发生转换，则返回0。

与strtol( )相关的函数是atol( )。

30.26 strtoul函数

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

strtoul( )函数把start所指的字符串中存储的用字符串表示的数字转换为unsigned long型并返回结果。数字的基由radix决定。如果radix是0，基数由调节常数说明的规则确定；如果radix是其他指定的值，则必须在2～36的范围内。

strtoul( )函数的工作原理如下：首先，函数将去掉字符串中的所有空白符。然后，读取每一个包含数字的字符。只要字符不是无符号长整型数的一部分，就会使该过程停止。这样的字符包括空白符、标点符号和字符。最后，如果有余数，end则被设置为指向初始字符串的余项。也就是说，如果用“100 Pliers”调用strtoul( )，函数的返回值为100L，end将指向Pliers前面的空白符。

如果结果不能表示为无符号长整型数，strtoul( )将返回LONG_MAX，全局变量errno被设置为ERANGE，表明出现了一个范围错误。如果没有发生转换，则返回0。

与strtoul( )相关的函数是strtol( )。

30.27 system函数

```
#include <cstdlib>
int system(const char *str);
```

system( )函数把str所指的字符串作为操作系统的命令处理程序传递。

如果用一个空指针调用system( )，若存在命令处理程序，函数返回一个非0值；否则，返回0（在没有操作系统和命令处理程序的专用系统中，将执行一些C++代码，因此不能假定已存在命令处理程序）。system( )的返回值是实现定义的。然而，一般来说，如果命令执行成功，函数返回0；否则，返回非0值。

与system( )相关的函数是exit( )。

30.28 va_arg，va_start和va_end函数

```
#include <cstdarg>
type va_arg(va_list argptr, type);
void va_end(va_list argptr);
void va_start(va_list argptr, last_parm);
```

va_arg( )，va_start( )和va_end( )宏一同工作，以允许数量可变的参数传递给一个函数。最常见的具有可变数量参数的函数例子是printf( )。类型va_list在<cstdarg>中定义。

创建带有可变参数个数的函数的一般过程如下：函数必须至少在可变参数列表前有一个（或多个）已知的参数。最右边的已知参数称为last_parm，last_parm的名字在调用va_start( )时被用做第二个参数。在访问任何可变长参数之前，参数指针argptr必须通过调用va_start( )被初始化。此后，这些参数通过调用va_arg( )返回，而且type为下一个参数的类型。最后，一旦读取了所有参数，在从该函数中返回之前，必须要调用va_end( )以确保堆栈被完全复原。如果没有调用va_end( )，程序很可能会崩溃。

与这些函数相关的函数有vprintf( )。

30.29 wcstombs函数

```
#include <cstdlib>
```

```
size_t wcstombs(char *out, const wchar_t *in, size_t size);
```

wcstombs( )把in所指的宽字节转换为等效的多字节并将结果放入out所指的数组。该函数只转换in中的前size个字节。如果在此之前遇到了null结束符，函数将停止转换。

如果转换成功，wcstombs( )返回转换的字节数；如果失败，则返回-1。

与wcstombs( )相关的函数有wctomb( )和mbstowcs( )。

30.30 wctomb函数

```
#include <cstdlib>
int wctomb(char *out, wchar_t in);
```

wctomb( )把in中的宽字符转换为等效的多字节并将结果放入out所指的对象中。out所指的数组长度至少必须为MB_CUR_MAX个字符。

如果转换成功，wctomb( )返回多字节字符包含的字节数；如果失败，则返回-1。

如果out为空，且多字节字符具有与状态相关的编码，wctomb( )将返回非0值；否则，返回0。

与wctomb( )相关的函数有wcstombs( )和mbtowc( )。