程序 14-3 linux/include/ctype.h

```
 1 #ifndef _CTYPE_H
 2 #define _CTYPE_H
 3
 4 #define _U      0x01    /* upper */          // 该比特位用于大写字符[A-Z]。
 5 #define _L      0x02    /* lower */          // 该比特位用于小写字符[a-z]。
 6 #define _D      0x04    /* digit */          // 该比特位用于数字[0-9]。
 7 #define _C      0x08    /* cntrl */          // 该比特位用于控制字符。
 8 #define _P      0x10    /* punct */          // 该比特位用于标点字符。
 9 #define _S      0x20    /* white space (space/lf/tab) */ // 空白字符，如空格、\t、\n 等。
10 #define _X      0x40    /* hex digit */          // 该比特位用于十六进制数字。
11 #define _SP     0x80    /* hard space (0x20) */  // 该比特位用于空格字符（0x20）。
12
13 extern unsigned char _ctype[];     // 字符特性数组（表），定义各个字符对应上面的属性。
14 extern char _ctmp;                 // 一个临时字符变量（在定义 lib/ctype.c 中）。
15
   // 下面是一些确定字符类型的宏。
16 #define isalnum(c) ((_ctype+1)[c]&(_U|_L|_D))     // 是字符或数字[A-Z]、[a-z]或[0-9]。
17 #define isalpha(c) ((_ctype+1)[c]&(_U|_L))        // 是字符。
18 #define iscntrl(c) ((_ctype+1)[c]&(_C))           // 是控制字符。
19 #define isdigit(c) ((_ctype+1)[c]&(_D))           // 是数字。
20 #define isgraph(c) ((_ctype+1)[c]&(_P|_U|_L|_D))  // 是图形字符。
21 #define islower(c) ((_ctype+1)[c]&(_L))           // 是小写字符。
22 #define isprint(c) ((_ctype+1)[c]&(_P|_U|_L|_D|_SP))  // 是可打印字符。
23 #define ispunct(c) ((_ctype+1)[c]&(_P))           // 是标点符号。
24 #define isspace(c) ((_ctype+1)[c]&(_S))           // 是空白字符如空格,\f,\n,\r,\t,\v。
25 #define isupper(c) ((_ctype+1)[c]&(_U))           // 是大写字符。
26 #define isxdigit(c) ((_ctype+1)[c]&(_D|_X))       // 是十六进制数字。
27
   // 在下面两个定义中，宏参数前使用了前缀（unsigned），因此 c 应该加括号，即表示成 (c)。
   // 因为在程序中 c 可能是一个复杂的表达式。例如，如果参数是 a + b，若不加括号，则在宏定
   // 义中变成了：(unsigned) a + b。这显然不对。加了括号就能正确表示成(unsigned)(a + b)。
28 #define isascii(c) (((unsigned) c)<=0x7f)           // 是 ASCII 字符。
29 #define toascii(c) (((unsigned) c)&0x7f)            // 转换成 ASCII 字符。
30
   // 以下两个宏定义中使用一个临时变量_ctmp 的原因是：在宏定义中，宏的参数只能被使用一次。
   // 但对于多线程来说这是不安全的，因为两个或多个线程可能在同一时刻使用这个公共临时变量。
   // 因此从 Linux 2.2.x 版本开始更改为使用两个函数来取代这两个宏定义。
31 #define tolower(c) (_ctmp=c,isupper(_ctmp)?_ctmp-('A'-'a'):_ctmp)  // 转换成小写字符。
32 #define toupper(c) (_ctmp=c,islower(_ctmp)?_ctmp-('a'-'A'):_ctmp)  // 转换成大写字符。
33
34 #endif
35
```