Chapter 31

The Wide-Character Functions

In 1995, a number of wide-character functions were added to Standard C and subsequently adopted by Standard C++. The wide-character functions operate on characters of type **wchar_t**, which are 16 bits. For the most part these functions parallel their **char** equivalents. For example, the function **iswspace( )** is the wide-character version of **isspace( )**. In general, the wide-character functions use the same names as their **char** equivalents, except that a "w" is added.

The wide-character functions use two headers: **<cwchar>** and **<cwctype>**. The C header files **wchar.h** and **wctype.h** are also supported.

The header **<cwctype>** defines the types **wint_t**, **wctrans_t**, and **wctype_t**. Many of the wide-character functions receive a wide character as a parameter. The type of this parameter is **wint_t**. It is capable of holding a wide character. The use of the **wint_t** type in the wide-character functions parallels the use of **int** in the **char**-based functions. **wctrans_t** and **wctype_t** are the types of objects used to represent a character mapping (i.e., character translation) and the classification of a character, respectively. The wide-character EOF mark is defined as **WEOF**.

In addition to defining **win_t**, the header **<cwchar>** defines the types **wchar_t**, **size_t**, and **mbstate_t**. The **wchar_t** type creates a wide character object, and **size_t** is the type of value returned by **sizeof**. The **mbstate_t** type describes an object that holds the state of a multibyte-to-wide-character conversion. The **<cwchar>** header also defines the macros **NULL**, **WEOF**, **WCHAR_MAX**, and **WCHAR_MIN**. The last two define the maximum and minimum value that can be held in an object of type **wchar_t**.

Although the standard function library's support for wide characters is quite extensive, these functions are not frequently used. One reason for this is that the Standard C++ I/O system and class libraries provide both normal and wide-character support through the use of template classes. Also, interest in wide-character-compliant programs has been less than expected. Of course, this situation may change.

Since most of the wide-character functions simply parallel their **char** equivalents and are not frequently used by most C++ programmers, only a brief description of these functions is provided.

The Wide-Character Classification Functions

The header **<cwctype>** provides the prototypes for the wide-character functions that support character classification. These functions categorize wide characters as to their type or convert the case of a character. Table 31-1 lists these functions along with their **char** equivalents, which are described in Chapter 26.

In addition to the functions shown in Table 31-1, **<cwctype>** defines the following ones, which provide an open-ended means of classifying characters.

```
wctype_t wctype(const char *attr);
int iswctype(wint_t ch, wctype_t attr_ob);
```

Function	char Equivalent
int iswalnum(wint_t *ch*)	isalnum( )
int iswalpha(wint_t *ch*)	isalpha( )
int iswcntrl(wint_t *ch*)	iscntrl( )
int iswdigit(wint_t *ch*)	isdigit( )
int iswgraph(wint_t *ch*)	isgraph( )
int iswlower(wint_t *ch*)	islower( )
int iswprint(wint_t *ch*)	isprint( )
int iswpunct(wint_t *c*)	ispunct( )
int iswspace(wint_t *ch*)	isspace( )
int iswupper(wint_t *ch*)	isupper( )
int iswxdigit(wint_t *ch*)	isxdigit( )
wint_t tolower(wint_t *ch*)	tolower( )
wint_t toupper(wint_t *ch*)	toupper( )

Table 31-1. *The Wide-Character Classification Functions*

The function **wctype( )** returns a value that can be passed to the *attr_ob* parameter to **iswctype( )**. The string pointed to by *attr* specifies a property that a character must have. The value in *attr_ob* is used to determine if *ch* is a character that has that property. If it does, **iswctype( )** returns nonzero. Otherwise, it returns zero. The following property strings are defined for all execution environments.

alnum	alpha	cntrl	digit
graph	lower	print	punct
space	upper	xdigit	

The following program demonstrates the **wctype( )** and **iswctype( )** functions.

```
#include <iostream>
#include <cwctype>
using namespace std;
```

```
int main()
{
  wctype_t x;

  x = wctype("space");

  if(iswctype(L' ', x))
    cout << "Is a space.\n";

  return 0;
}
```

This program displays "Is a space."

The functions **wctrans( )** and **towctrans( )** are also defined in **<cwctype>**. They are shown here:

wctrans_t wctrans(const char **mapping*);
wint_t towctrans(wint_t *ch*, wctrans_t *mapping_ob*);

The function **wctrans( )** returns a value that can be passed to the *mapping_ob* parameter to **towctrans( )**. Here, the string pointed to by *mapping* specifies a mapping of one character to another. This value can then be used by **iswctrans( )** to map *ch*. The mapped value is returned. The following mapping strings are supported in all execution environments.

tolower	toupper

Here is a short example that demonstrates **wctrans( )** and **towctrans( )**.

```
#include <iostream>
#include <cwctype>
using namespace std;

int main()
{
  wctrans_t x;

  x = wctrans("tolower");
  wchar_t ch = towctrans(L'W', x);
  cout << (char) ch;

  return 0;
}
```

This program displays a lowercase "w".

The Wide-Character I/O Functions

Several of the I/O functions described in Chapter 25 have wide-character implementations. These functions are shown in Table 31-2. The wide-character I/O functions use the header **<cwchar>**. Notice that **swprintf( )** and **vswprintf( )** require an additional parameter not needed by their **char** equivalents.

In addition to those shown in the table, the following wide-character I/O function has been added:

int fwide(FILE **stream*, int *how*);

If *how* is positive, **fwide( )** makes *stream* a wide-character stream. If *how* is negative, **fwide( )** makes stream into a **char** stream. If *how* is zero, the stream is unaffected. If the stream has already been oriented to either wide or normal characters, it will not be changed. The function returns positive if the stream uses wide characters, negative if the stream uses **chars**, and zero if the stream has not yet been oriented. A stream's orientation is also determined by its first use.

The Wide-Character String Functions

There are wide-character versions of the string manipulation functions described in Chapter 26. These are shown in Table 31-3. They use the header **<cwchar>**. Note that **wcstok( )** requires an additional parameter not used by its **char** equivalent.

Wide-Character String Conversion Functions

The functions shown in Table 31-4 provide wide-character versions of the standard numeric and time conversion functions. These functions use the header **<cwchar>**.

Function	char Equivalent
win_t fgetwc(FILE **stream*)	fgetc( )
wchar_t *fgetws(wchar_t **str*, int *num*, FILE **stream*)	fgets( )
wint_t fputwc(wchar_t *ch*, FILE **stream*)	fputc( )
int fputws(const wchar_t **str*, FILE **stream*)	fputs( )
int fwprintf(FILE **stream*, const wchar_t *fmt*, ...)	fprintf( )
int fwscanf(FILE **stream*, const wchar_t *fmt*, ...)	fscanf( )
wint_t getwc(FILE **stream*)	getc( )
wint_t getwchar( )	getchar( )
wint_t putwc(wchar_t *ch*, FILE **stream*)	putc( )
wint_t putwchar(wchar_t *ch*)	putchar( )
int swprintf(wchar_t **str*, size_t *num*, const wchar_t **fmt*, ...)	sprintf( ) Note the addition of the parameter *num*, which limits the number of characters written to *str*.
int swscanf(const wchar_t **str*, const wchar_t **fmt*, ...)	sscanf( )
wint_t ungetwc(wint_t *ch*, FILE **stream*)	ungetc( )
int vfwprintf(FILE **stream*, const wchar_t *fmt*, va_list *arg*)	vfprintf( )
int vswprintf(wchar_t **str*, size_t *num*, const wchar_t **fmt*, va_list *arg*)	vsprintf( ) Note the addition of the parameter *num*, which limits the number of characters written to *str*.
int vwprintf(const wchar_t **fmt*, va_list *arg*)	vprintf( )
int wprintf(const wchar_t **fmt*, ...)	printf( )
int wscanf(const wchar_t **fmt*, ...)	scanf( )

Table 81-2. *The Wide-Character I/O Functions*

Function	char Equivalent
wchar_t *wcscat(wchar_t **str1*, const wchar_t **str2*)	strcat( )
wchar_t *wcschr(const wchar_t **str*, wchar_t *ch*)	strchr( )
int wcscmp(const wchar_t **str1*, const wchar_t **str2*)	strcmp( )
int wcscoll(const wchar_t **str1*, const wchar_t **str2*)	strcoll( )
size_t wcscspn(const wchar_t **str1*, const wchar_t **str2*)	strcspn( )
wchar_t *wcscpy(wchar_t **str1*, const wchar_t **str2*)	strcpy( )
size_t wcslen(const wchar_t **str*)	strlen( )
wchar_t *wcsncpy(wchar_t **str1*, const wchar_t *str2*, size_t *num*)	strncpy( )
wchar_t *wcsncat(wchar_t **str1*, const wchar_t *str2*, size_t *num*)	strncat( )
int wcsncmp(const wchar_t **str1*, const wchar_t **str2*, size_t *num*)	strncmp( )
wchar_t *wcspbrk(const wchar_t **str1*, const wchar_t **str2*)	strpbrk( )
wchar_t *wcsrchr(const wchar_t **str*, wchar_t *ch*)	strrchr( )
size_t wcsspn(const wchar_t **str1*, const wchar_t *str2*)	strspn( )
wchar_t *wcstok(wchar_t **str1*, const wchar_t **str2*, wchar_t ***endptr*)	strtok( ) Here, *endptr* is a pointer that holds information necessary to continue the tokenizing process.
wchar_t *wcsstr(const wchar_t **str1*, const wchar_t **str2*)	strstr( )
size_t wcsxfrm(wchar_t **str1*, const wchar_t **str2*, size_t *num*)	strxfrm( )

Table 31-3. *The Wide-Character String Functions*

Function	char Equivalent
size_t wcsftime(wchar_t *str, size_t max, const wchar_t *fmt, const struct tm *ptr)	strftime( )
double wcstod(const wchar_t *start, wchar_t **end);	strtod( )
long wcstol(const wchar_t *start, wchar_t **end, int radix)	strtol( )
unsigned long wcstoul(const wchar_t *start, wchar_t **end, int radix)	strtoul( )

Table 31-4. *The Wide-Character Conversion Functions*

Wide-Character Array Functions

The standard character array-manipulation functions, such as **memcpy( )**, also have wide-character equivalents. They are shown in Table 31-5. These functions use the header **<cwchar>**.

Function	char Equivalent
wchar_t *wmemchr(const wchar_t *str, wchar_t ch, size_t num)	memchr( )
int wmemcmp(const wchar_t *str1, const wchar_t *str2, size_t num)	memcmp( )
wchar_t *wmemcpy(wchar_t *str1, const wchar_t *str2, size_t num)	memcpy( )
wchar_t *wmemmove(wchar_t *str1, const wchar_t *str2, size_t num)	memmove( )
wchar_t *wmemset(wchar_t *str, wchar_t ch, size_t num)	memset( )

Table 31-5. *The Wide-Character Array Functions*

Multibyte/Wide-Character Conversion Functions

The Standard C++ function library supplies various functions that support conversions between multibyte and wide characters. These functions, shown in Table 31-6, use the header **<cwchar>**. Many of them are *restartable* versions of the normal multibyte functions. The restartable version utilizes the state information passed to it in a parameter of type **mbstate_t**. If this parameter is null, the function will provide its own **mbstate_t** object.

Function	Description
win_t btowc(int *ch*)	Converts *ch* into its wide-character equivalent and returns the result. Returns **WEOF** on error or if *ch* is not a one-byte, multibyte character.
size_t mbrlen(const char **str*, size_t *num*, mbstate_t **state*)	Restartable version of **mblen( )** as described by *state*. Returns a positive value that indicates the length of the next multibyte character. Zero is returned if the next character is null. A negative value is returned if an error occurs.
size_t mbrtowc(wchar_t **out*, const char **in*, size_t *num*, mbstate_t **state*)	Restartable version of **mbtowc( )** as described by *state*. Returns a positive value that indicates the length of the next multibyte character. Zero is returned if the next character is null. A negative value is returned if an error occurs. If an error occurs, the macro **EILSEQ** is assigned to **errno**.
int mbsinit(const mbstate_t **state*)	Returns true if *state* represents an initial conversion state.
size_t mbsrtowcs(wchar_t **out*, const char ***in*, size_t *num*, mbstate_t *state*)	Restartable version of **mbstowcs( )** as described by *state*. Also, **mbsrtowcs( )** differs from **mbstowcs( )** in that *in* is an indirect pointer to the source array. If an error occurs, the macro **EILSEQ** is assigned to **errno**.

Table 31-6. *Wide-Character/Multibyte Conversion Functions*

Function	Description
size_t wcrtomb(char **out*, wchar_t *ch*, mbstate_t **state*)	Restartable version of **wctomb( )** as described by *state*. If an error occurs, the macro **EILSEQ** is assigned to **errno**.
size_t wcsrtombs(char **out*, const wchar_t ***in*, size_t *num*, mbstate_t **state*)	Restartable version of **wcstombs( )** as described by *state*. Also, **wcsrtombs( )** differs from **wcstombs( )** in that *in* is an indirect pointer to the source array. If an error occurs, the macro **EILSEQ** is assigned to **errno**.
int wctob(wint_t *ch*)	Converts *ch* into its one-byte, multibyte equivalent. It returns **EOF** on failure.

Table 31-6. *Wide-Character/Multibyte Conversion Functions* (continued)