NAME

wcstring, wcscat, wscat, wcsncat, wsncat, wcscmp, wscmp, wcsncmp, wsncmp, wcscpy, wscpy, wcsncpy, wsncpy, wcslen, wslen, wcwidth, wcswidth, wcschr, wschr, wcsrchr, wsrchr, windex, wrindex, wcspbrk, wspbrk, wcswcs, wcsspn, wsspn, wcscspn, wscspn, wcstok, wstok – wide character string operations

SYNOPSIS

cc [*flag* ...] *file* ... **–lw** [*library* ...]

#include <wchar.h>

wchar_t ∗**wcscat(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2***);**
wchar_t ∗**wscat(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2***);**

wchar_t ∗**wcsncat(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2,* **size_t** *n***);**
wchar_t ∗**wsncat(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2,* **size_t** *n***);**

int wcscmp(const wchar_t ∗ws1**, const wchar_t** ∗ws2**);**
int wscmp(const wchar_t ∗ws1**, const wchar_t** ∗ws2**);**

int wcsncmp(const wchar_t ∗*ws1,* **const wchar_t** ∗*ws2,* **size_t** *n***);**
int wsncmp(const wchar_t ∗*ws1,* **const wchar_t** ∗*ws2,* **size_t** *n***);**

wchar_t ∗**wcscpy(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2***);**
wchar_t ∗**wscpy(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2***);**

wchar_t ∗**wcsncpy(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2,* **size_t** *n***);**
wchar_t ∗**wsncpy(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2,* **size_t** *n***);**

size_t wcslen(const wchar_t ∗*ws***);**
size_t wslen(const wchar_t ∗*ws***);**

int wcwidth(wint_t *wc***);**

int wcswidth(const wchar_t ∗*pwcs,* **size_t** *n***);**

wchar_t ∗**wcschr(const wchar_t** ∗*ws,* **wint_t** *wc***);**
wchar_t ∗**wschr(const wchar_t** ∗*ws,* **wint_t** *wc***);**

wchar_t ∗**wcsrchr(const wchar_t** ∗*ws,* **wint_t** *wc***);**
wchar_t ∗**wsrchr(const wchar_t** ∗*ws,* **wint_t** *wc***);**

wchar_t ∗**windex(const wchar_t** ∗*ws***, wchar_t** *wc***);**
wchar_t ∗**wrindex(const wchar_t** ∗*ws***, wchar_t** *wc***);**

wchar_t ∗**wcspbrk(const wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2***);**
wchar_t ∗**wspbrk(const wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2***);**

wchar_t ∗**wcswcs(const wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2***);**

size_t wcsspn(const wchar_t ∗*ws1,* **const wchar_t** ∗*ws2***);**
size_t wsspn(const wchar_t ∗*ws1,* **const wchar_t** ∗*ws2***);**

size_t wcscspn(const wchar_t ∗*ws1,* **const wchar_t** ∗*ws2***);**
size_t wscspn(const wchar_t ∗*ws1,* **const wchar_t** ∗*ws2***);**

wchar_t ∗**wcstok(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2***);**
wchar_t ∗**wstok(wchar_t** ∗*ws1,* **const wchar_t** ∗*ws2***);**

MT-LEVEL

MT-Safe

DESCRIPTION

These functions operate on wide character strings terminated by **wchar_t NULL** characters. During appending or copying, these routines do not check for an overflow condition of the receiving string. In the following, *ws*, *ws1*, and *ws2* point to wide character strings terminated by a **wchar_t NULL**.

wcscat( ), wscat( )

The **wcscat( )** and **wscat( )** functions append a copy of the wide character string pointed to by *ws2* (including the terminating null wide-character code) to the end of the wide character string pointed to by *ws1*. The initial wide-character code of *ws2* overwrites the null wide-character code at the end of *ws1*. If copying takes place between objects that overlap, the behaviour is undefined. Both functions return *s1*; no return value is reserved to indicate an error.

wcsncat( ), wsncat( )

The **wcsncat( )** and **wsncat( )** functions append not more than *n* wide-character codes (a null wide-character code and wide character codes that follow it are not appended) from the array pointed to by *ws2* to the end of the wide character string pointed to by *ws1*. The initial wide-character code of *ws2* overwrites the null wide-character code at the end of *ws1*. A terminating null wide-character code is always appended to the result. Both functions return *ws1*; no return value is reserved to indicate an error.

wcscmp( ), wscmp( )

The **wcscmp( )** and **wscmp( )** functions compare the wide character string pointed to by *ws1* to the wide character string pointed to by *ws2*. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared. Upon completion, both functions return an integer greater than, equal to, or less than zero, if the wide character string pointed to by *ws1* is greater than, equal to, or less than the wide character string pointed to by *ws2*.

wcsncmp( ), wsncmp( )

The **wcsncmp( )** and **wsncmp( )** functions compare not more than *n* wide-character codes (wide-character codes that follow a null wide character code are not compared) from the array pointed to by *ws1* to the array pointed to by *ws2*. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared. Upon successful completion, both functions return an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less than the possibly null-terminated array pointed to by *ws2*.

wcscpy( ), wscpy( )

The **wcscpy( )** and **wscpy( )** functions copy the wide character string pointed to by *ws2* (including the terminating null wide-character code) into the array pointed to by *ws1*. If copying takes place between objects that overlap, the behaviour is undefined. Both functions return *ws1*; no return value is reserved to indicate an error.

wcsncpy( ), wsncpy( )

The **wcsncpy( )** and **wsncpy( )** functions copy not more than *n* wide-character codes (wide-character codes that follow a null wide character code are not copied) from the array pointed to by *ws2* to the array pointed to by *ws1*. If copying takes place between objects that overlap, the behaviour is undefined. If the array pointed to by *ws2* is a wide character string that is shorter than *n* wide-character codes, null wide-character codes are appended to the copy in the array pointed to by *ws1*, until a total *n* wide-character codes are written. Both functions return *ws1*; no return value is reserved to indicate an error.

wcslen( ), wslen( )

The **wcslen( )** and **wslen( )** functions compute the number of wide-character codes in the wide character string to which *ws* points, not including the terminating null wide-character code. Both functions return *ws*; no return value is reserved to indicate an error.

wcwidth( )

The **wcwidth( )** function determines the number of column positions required for the wide character *wc*. The value of *wc* must be a character representable as a **wchar_t**, and must be a wide-character code corresponding to a valid character in the current locale. The function returns **0** if *wc* is a null wide-character code; the number of column positions to be occupied by the wide-character code *wc*; or −**1** if *wc* does not correspond to a printing wide-character code.

wcswidth( )

The **wcswidth( )** function determines the number of column positions required for *n* wide-character codes (or fewer than *n* wide-character codes if a null wide-character code is encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*. The function returns **0** if *pwcs* points to a null wide-character code; the number of column positions to be occupied by the wide character string pointed to by *pwcs*; or −**1** if any wide-character code in the wide character string pointed to by *pwcs* is not a printing wide-character code.

wcschr( ), wschr( )

The **wcschr( )** and **wschr( )** functions locate the first occurrence of *wc* in the wide character string pointed to by *ws*. The value of *wc* must be a character representable as a type **wchar_t** and must be a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code is considered to be part of the wide character string. Upon completion, both functions return a pointer to the wide-character code, or a null pointer if the wide-character code is not found.

wcsrchr( ), wsrchr( )

The **wcsrchr( )** and **wsrchr( )** functions locate the last occurrence of *wc* in the wide character string pointed to by *ws*. The value of *wc* must be a character representable as a type **wchar_t** and must be a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code is considered to be part of the wide character string. Upon successful completion, both functions return a pointer to the wide-character code, or a null pointer if *wc* does not occur in the wide character string.

windex( ), wrindex( )

The **windex( )** and **wrindex( )** functions behave the same as **wschr( )** and **wsrchr( )**, respectively.

wcspbrk( ), wspbrk( )

The **wcspbrk( )** and **wspbrk( )** functions locate the first occurrence in the wide character string pointed to by *ws1* of any wide-character code from the wide character string pointed to by *ws2*. Upon successful completion, the function returns a pointer to the wide-character code, or a null pointer if no wide-character code from *ws2* occurs in *ws1*.

wcswcs( )

The **wcswcs( )** function locates the first occurrence in the wide character string pointed to by *ws1* of the sequence of wide-character codes (excluding the terminating null wide-character code) in the wide character string pointed to by *ws2*. Upon successful completion, the function returns a pointer to the located wide character string, or a null pointer if the wide character string is not found. If *ws2* points to a wide character string with zero length, the function returns *ws1*.

wcsspn( ), wsspn( )

The **wcsspn( )** and **wsspn( )** functions compute the length of the maximum initial segment of the wide character string pointed to by *ws1* which consists entirely of wide-character codes from the wide string pointed to by *ws2*. Both functions return *ws1*; no return value is reserved to indicate an error.

wcscspn( ), wscspn( )

The **wcscspn( )** and **wscspn( )** functions compute the length of the maximum initial segment of the wide character string pointed to by *ws1* which consists entirely of wide-character codes *not* from the wide character string pointed to by *ws2*. Both functions return *ws1*; no return value is reserved to indicate an error.

wcstok( ), wstok( )

A sequence of calls to the **wcstok( )** and **wstok( )** functions break the wide character string pointed to by *ws1* into a sequence of tokens, each of which is delimited by a wide-character code from the wide character string pointed to by *ws2*. The first call in the sequence has *ws1* as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by *ws2* may be different from call to call.

The first call in the sequence searches the wide character string pointed to by *ws1* for the first wide-character code that is *not* contained in the current separator string pointed to by *ws2*. If no such wide-character code is found, then there are no tokens in the wide character string pointed to by *ws1*, and **wcstok( )** and **wstok( )** return a null pointer. If such a wide-character code is found, it is the start of the first token.

wcstok( ) and **wstok( )** then search from that point for a wide-character code that *is* contained in the current separator string. If no such wide-character code is found, the current token extends to the end of the wide character string pointed to by *ws1*, and subsequent searches for a token will return a null pointer. If such a wide-character code is found, it is overwritten by a null wide character, which terminates the current token. **wcstok( )** and **wstok( )** save a pointer to the following wide-character code, from which the next search for a token will start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

Upon successful completion, both functions return a pointer to the first wide-character code of a token. Otherwise, if there is no token, a null pointer is returned.

SEE ALSO

malloc(3C), **string**(3C), **wstring**(3I)