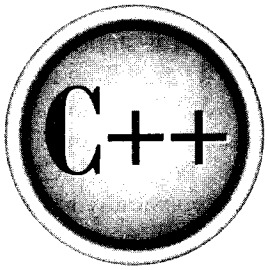
The Complete Reference



Chapter 26

The String and Character Functions

The standard function library has a rich and varied set of string and character handling functions. The string functions operate on null-terminated arrays of characters and require the header **<cstring>**. The character functions use the header **<cctype>**. C programs must use the header files **string.h** and **ctype.h**.

Because C/C++ has no bounds checking on array operations, it is the programmer's responsibility to prevent an array overflow. Neglecting to do so may cause your program to crash.

In C/C++, a *printable character* is one that can be displayed on a terminal. These are usually the characters between a space (0x20) and tilde (0xFE). *Control characters* have values between (0) and (0x1F) as well as DEL (0x7F).

For historical reasons, the parameters to the character functions are integers, but only the low-order byte is used; the character functions automatically convert their arguments to **unsigned char**. However, you are free to call these functions with character arguments because characters are automatically elevated to integers at the time of the call.

The header **<cstring>** defines the **size_t** type, which is essentially the same as **unsigned**.

This chapter describes only those functions that operate on characters of type **char**. These are the functions originally defined by Standard C and C++, and they are by far the most widely used and supported. Wide-character functions that operate on characters of type **wchar_t** are discussed in Chapter 31.

isalnum

```
#include <cctype>
int isalnum(int ch);
```

The **isalnum( )** function returns nonzero if its argument is either a letter of the alphabet or a digit. If the character is not alphanumeric, zero is returned.

Related functions are **isalpha( )**, **iscntrl( )**, **isdigit( )**, **isgraph( )**, **isprint( )**, **ispunct( )**, and **isspace( )**.

isalpha

```
#include <cctype>
int isalpha(int ch);
```

The **isalpha( )** function returns nonzero if *ch* is a letter of the alphabet; otherwise zero is returned. What constitutes a letter of the alphabet may vary from language to language. For English, these are the upper- and lowercase letters A through Z.

Related functions are **isalnum( )**, **iscntrl( )**, **isdigit( )**, **isgraph( )**, **isprint( )**, **ispunct( )**, and **isspace( )**.

iscntrl

```
#include <cctype>
int iscntrl(int ch);
```

The **iscntrl( )** function returns nonzero if *ch* is between zero and 0x1F or is equal to 0x7F (DEL); otherwise zero is returned.

Related functions are **isalnum( )**, **isalpha( )**, **isdigit( )**, **isgraph( )**, **isprint( )**, **ispunct( )**, and **isspace( )**.

isdigit

```
#include <cctype>
int isdigit(int ch);
```

The **isdigit( )** function returns nonzero if *ch* is a digit, that is, 0 through 9. Otherwise zero is returned.

Related functions are **isalnum( )**, **isalpha( )**, **iscntrl( )**, **isgraph( )**, **isprint( )**, **ispunct( )**, and **isspace( )**.

isgraph

```
#include <cctype>
int isgraph(int ch);
```

The **isgraph( )** function returns nonzero if *ch* is any printable character other than a space; otherwise zero is returned. These are characters generally in the range 0x21 through 0x7E.

Related functions are **isalnum( )**, **isalpha( )**, **iscntrl( )**, **isdigit( )**, **isprint( )**, **ispunct( )**, and **isspace( )**.

islower

```
#include <cctype>
int islower(int ch);
```

The **islower( )** function returns nonzero if *ch* is a lowercase letter; otherwise zero is returned.

A related function is **isupper( )**.

isprint

```
#include <cctype>
int isprint(int ch);
```

The **isprint( )** function returns nonzero if *ch* is a printable character, including a space; otherwise zero is returned. Printable characters are often in the range 0x20 through 0x7E.

Related functions are **isalnum( )**, **isalpha( )**, **iscntrl( )**, **isdigit( )**, **isgraph( )**, **ispunct( )**, and **isspace( )**.

ispunct

```
#include <cctype>
int ispunct(int ch);
```

The **ispunct( )** function returns nonzero if *ch* is a punctuation character; otherwise zero is returned. The term "punctuation," as defined by this function, includes all printing characters that are neither alphanumeric nor a space.

Related functions are **isalnum( )**, **isalpha( )**, **iscntrl( )**, **isdigit( )**, **isgraph( )**, and **isspace( )**.

isspace

```
#include <cctype>
int isspace(int ch);
```

The **isspace( )** function returns nonzero if *ch* is either a space, horizontal tab, vertical tab, formfeed, carriage return, or newline character; otherwise zero is returned.

Related functions are **isalnum( )**, **isalpha( )**, **iscntrl( )**, **isdigit( )**, **isgraph( )**, and **ispunct( )**.

isupper

```
#include <cctype>
int isupper(int ch);
```

The **isupper( )** function returns nonzero if *ch* is an uppercase letter; otherwise zero is returned.

A related function is **islower( )**.

isxdigit

```
#include <cctype>
int isxdigit(int ch);
```

The **isxdigit( )** function returns nonzero if *ch* is a hexadecimal digit; otherwise zero is returned. A hexadecimal digit will be in one of these ranges: A–F, a–f, or 0–9.

Related functions are **isalnum( )**, **isalpha( )**, **iscntrl( )**, **isdigit( )**, **isgraph( )**, **ispunct( )**, and **isspace( )**.

memchr

```
#include <cstring>
void *memchr(const void *buffer, int ch, size_t count);
```

The **memchr( )** function searches the array pointed to by *buffer* for the first occurrence of *ch* in the first *count* characters.

The **memchr( )** function returns a pointer to the first occurrence of *ch* in *buffer*, or it returns a null pointer if *ch* is not found.

Related functions are **memcpy( )** and **isspace( )**.

memcmp

```
#include <cstring>
int memcmp(const void *buf1, const void *buf2, size_t count);
```

The **memcmp( )** function compares the first *count* characters of the arrays pointed to by *buf1* and *buf2*.

The **memcmp( )** function returns an integer that is interpreted as indicated here:

Value	Meaning
Less than zero	*buf1* is less than *buf2*.
Zero	*buf1* is equal to *buf2*.
Greater than zero	*buf1* is greater than *buf2*.

Related functions are **memchr( )**, **memcpy( )**, and **strcmp( )**.

memcpy

```
#include <cstring>
void *memcpy(void *to, const void *from, size_t count);
```

The **memcpy( )** function copies *count* characters from the array pointed to by *from* into the array pointed to by *to*. If the arrays overlap, the behavior of **memcopy( )** is undefined.

The **memcpy( )** function returns a pointer to *to*.

A related function is **memmove( )**.

memmove

```
#include <cstring>
void *memmove(void *to, const void *from, size_t count);
```

The **memmove( )** function copies *count* characters from the array pointed to by *from* into the array pointed to by *to*. If the arrays overlap, the copy will take place correctly, placing the correct contents into *to* but leaving *from* modified.

The **memmove( )** function returns a pointer to *to*.

A related function is **memcpy( )**.

memset

```
#include <cstring>
void *memset(void *buf, int ch, size_t count);
```

The **memset( )** function copies the low-order byte of *ch* into the first *count* characters of the array pointed to by *buf*. It returns *buf*.

The most common use of **memset( )** is to initialize a region of memory to some known value.

Related functions are **memcmp( )**, **memcpy( )**, and **memmove( )**.

strcat

```
#include <cstring>
char *strcat(char *str1, const char *str2);
```

The **strcat( )** function concatenates a copy of *str2* to *str1* and terminates *str1* with a null. The null terminator originally ending *str1* is overwritten by the first character of *str2*. The string *str2* is untouched by the operation. If the arrays overlap, the behavior of **strcat( )** is undefined.

The **strcat( )** function returns *str1*.

Remember, no bounds checking takes place, so it is the programmer's responsibility to ensure that *str1* is large enough to hold both its original contents and also those of *str2*.

Related functions are **strchr( )**, **strcmp( )**, and **strcpy( )**.

strchr

```
#include <cstring>
char *strchr(const char *str, int ch);
```

The **strchr( )** function returns a pointer to the first occurrence of the low-order byte of *ch* in the string pointed to by *str*. If no match is found, a null pointer is returned.

Related functions are **strpbrk( )**, **strspn( )**, **strstr( )**, and **strtok( )**.

strcmp

```
#include <cstring>
int strcmp(const char *str1, const char *str2);
```

The **strcmp( )** function lexicographically compares two strings and returns an integer based on the outcome as shown here:

Value	Meaning
Less than zero	*str1* is less than *str2*.
Zero	*str1* is equal to *str2*.
Greater than zero	*str1* is greater than *str2*.

Related functions are **strchr( )**, **strcpy( )**, and **strcmp( )**.

strcoll

```
#include <cstring>
int strcoll(const char *str1, const char *str2);
```

The **strcoll( )** function compares the string pointed to by *str1* with the one pointed to by *str2*. The comparison is performed in accordance to the locale specified using the **setlocale( )** function (see **setlocale** for details).

The **strcoll( )** function returns an integer that is interpreted as indicated here:

Value	Meaning
Less than zero	*str1* is less than *str2*.
Zero	*str1* is equal to *str2*.
Greater than zero	*str1* is greater than *str2*.

Related functions are **memcmp( )** and **strcmp( )**.

strcpy

```
#include <cstring>
char *strcpy(char *str1, const char *str2);
```

The **strcpy( )** function copies the contents of *str2* into *str1*. *str2* must be a pointer to a null-terminated string. The **strcpy( )** function returns a pointer to *str1*.

If *str1* and *str2* overlap, the behavior of **strcpy( )** is undefined.

Related functions are **memcpy( )**, **strchr( )**, **strcmp( )**, and **strncmp( )**.

strcspn

```
#include <cstring>
size_t strcspn(const char *str1, const char *str2);
```

The **strcspn( )** function returns the length of the initial substring of the string pointed to by *str1* that is made up of only those characters not contained in the string pointed to by *str2*. Stated differently, **strcspn( )** returns the index of the first character in the string pointed to by *str1* that matches any of the characters in the string pointed to by *str2*.

Related functions are **strrchr( )**, **strpbrk( )**, **strstr( )**, and **strtok( )**.

strerror

```
#include <cstring>
char *strerror(int errnum);
```

The **strerror( )** function returns a pointer to an implementation-defined string associated with the value of *errnum*. Under no circumstances should you modify the string.

strlen

```
#include <cstring>
size_t strlen(const char *str);
```

The **strlen( )** function returns the length of the null-terminated string pointed to by *str*. The null terminator is not counted.

Related functions are **memcpy( )**, **strchr( )**, **strcmp( )**, and **strncmp( )**.

strncat

```
#include <cstring>
char *strncat(char *str1, const char *str2, size_t count);
```

The **strncat( )** function concatenates not more than *count* characters of the string pointed to by *str2* to the string pointed to by *str1* and terminates *str1* with a null. The null terminator originally ending *str1* is overwritten by the first character of *str2*. The string *str2* is untouched by the operation. If the strings overlap, the behavior is undefined.

The **strncat( )** function returns *str1*.

Remember that no bounds checking takes place, so it is the programmer's responsibility to ensure that *str1* is large enough to hold both its original contents and also those of *str2*.

Related functions are **strcat( )**, **strnchr( )**, **strncmp( )**, and **strncpy( )**.

strncmp

```
#include <cstring>
int strncmp(const char *str1, const char *str2, size_t count);
```

The **strncmp( )** function lexicographically compares not more than *count* characters from the two null-terminated strings and returns an integer based on the outcome, as shown here:

Value	Meaning
Less than zero	*str1* is less than *str2*.
Zero	*str1* is equal to *str2*.
Greater than zero	*str1* is greater than *str2*.

If there are less than *count* characters in either string, the comparison ends when the first null is encountered.

Related functions are **strcmp( )**, **strnchr( )**, and **strncpy( )**.

strncpy

```
#include <cstring>
char *strncpy(char *str1, const char *str2, size_t count);
```

The **strncpy( )** function copies up to *count* characters from the string pointed to by *str2* into the string pointed to by *str1*. *str2* must be a pointer to a null-terminated string.

If *str1* and *str2* overlap, the behavior of **strncpy( )** is undefined.

If the string pointed to by *str2* has less than *count* characters, nulls will be appended to the end of *str1* until *count* characters have been copied.

Alternatively, if the string pointed to by *str2* is longer than *count* characters, the resultant string pointed to by *str1* will not be null terminated.

The **strncpy( )** function returns a pointer to *str1*.

Related functions are **memcpy( )**, **strchr( )**, **strncat( )**, and **strncmp( )**.

strpbrk

```
#include <cstring>
char *strpbrk(const char *str1, const char *str2);
```

The **strpbrk( )** function returns a pointer to the first character in the string pointed to by *str1* that matches any character in the string pointed to by *str2*. The null terminators are not included. If there are no matches, a null pointer is returned.

Related functions are **strspn( )**, **strrchr( )**, **strstr( )**, and **strtok( )**.

strrchr

```
#include <cstring>
char *strrchr(const char *str, int ch);
```

The **strrchr( )** function returns a pointer to the last occurrence of the low-order byte of *ch* in the string pointed to by *str*. If no match is found, a null pointer is returned.

Related functions are **strpbrk( )**, **strspn( )**, **strstr( )**, and **strtok( )**.

strspn

```
#include <cstring>
size_t strspn(const char *str1, const char *str2);
```

The **strspn( )** function returns the length of the initial substring of the string pointed to by *str1* that is made up of only those characters contained in the string pointed to by *str2*. Stated differently, **strspn( )** returns the index of the first character in the string pointed to by *str1* that does not match any of the characters in the string pointed to by *str2*.

Related functions are **strpbrk( )**, **strrchr( )**, **strstr( )**, and **strtok( )**.

strstr

```
#include <cstring>
char *strstr(const char *str1, const char *str2);
```

The **strstr( )** function returns a pointer to the first occurrence in the string pointed to by *str1* of the string pointed to by *str2*. It returns a null pointer if no match is found.

Related functions are **strchr( )**, **strcspn( )**, **strpbrk( )**, **strspn( )**, **strtok( )**, and **strrchr( )**.

strtok

```
#include <cstring>
char *strtok(char *str1, const char *str2);
```

The **strtok( )** function returns a pointer to the next token in the string pointed to by *str1*. The characters making up the string pointed to by *str2* are the delimiters that determine the token. A null pointer is returned when there is no token to return.

To tokenize a string, the first call to **strtok( )** must have *str1* point to the string being tokenized. Subsequent calls must use a null pointer for *str1*. In this way, the entire string can be reduced to its tokens.

It is possible to use a different set of delimiters for each call to **strtok( )**.

Related functions are **strchr( )**, **strcspn( )**, **strpbrk( )**, **strrchr( )**, and **strspn( )**.

strxfrm

```
#include <cstring>
size_t strxfrm(char *str1, const char *str2, size_t count);
```

The **strxfrm( )** function transforms the string pointed to by *str2* so that it can be used by the **strcmp( )** function and puts the result into the string pointed to by *str1*. After the transformation, the outcome of a **strcmp( )** using *str1* and a **strcoll( )** using the original string pointed to by *str2* will be the same. Not more than *count* characters are written to the array pointed to by *str1*.

The **strxfrm( )** function returns the length of the transformed string.

A related function is **strcoll( )**.

tolower

```
#include <cctype>
int tolower(int ch);
```

The **tolower( )** function returns the lowercase equivalent of *ch* if *ch* is a letter; otherwise *ch* is returned unchanged.

A related function is **toupper( )**.

toupper

```
#include <cctype>
int toupper(int ch);
```

The **toupper( )** function returns the uppercase equivalent of *ch* if *ch* is a letter; otherwise *ch* is returned unchanged.

A related function is **tolower( )**.