NAME

string, strcasecmp, strncasecmp, strcat, strncat, strchr, strrchr, strcmp, strncmp, strcpy, strncpy, strcspn, strspn, strdup, strlen, strpbrk, strstr, strtok, strtok_r – string operations

SYNOPSIS

#include <string.h>

int strcasecmp(const char **s1*, **const char** **s2*);

int strncasecmp(const char **s1*, **const char** **s2*, **int** *n*);

char *strcat(char **dst*, **const char** **src*);

char *strncat(char **dst*, **const char** **src*, **size_t** *n*);

char *strchr(const char **s*, **int** *c*);

char *strrchr(const char **s*, **int** *c*);

int strcmp(const char **s1*, **const char** **s2*);

int strncmp(const char **s1*, **const char** **s2*, **size_t** *n*);

char *strcpy(char **dst*, **const char** **src*);

char *strncpy(char **dst*, **const char** **src*, **size_t** *n*);

size_t strcspn(const char **s1*, **const char** **s2*);

size_t strspn(const char **s1*, **const char** **s2*);

char *strdup(const char **s1*);

size_t strlen(const char **s*);

char *strpbrk(const char **s1*, **const char** **s2*);

char *strstr(const char **s1*, **const char** **s2*);

char *strtok(char **s1*, **const char** **s2*);

char *strtok_r(char **s1*, **const char** **s2*, **char** ***lasts*);

MT-LEVEL

See the **NOTES** section of this page.

DESCRIPTION

string, strcasecmp, strncasecmp, strcat, strncat, strchr, strrchr, strcmp, strncmp, strcpy, strncpy, strcspn, strspn, strdup, strlen, strpbrk, strstr, strtok, strtok_r – string operations The arguments *s*, *s1*, *s2*, *src*, and *dst* point to strings (arrays of characters terminated by a null character). The functions **strcat()**, **strncat()**, **strcpy()**, **strncpy()**, **strtok()**, and **strtok_r()** all alter their first argument. These functions do not check for overflow of the array pointed to by the first argument.

strcasecmp() and **strncasecmp()** are case-insensitive versions of **strcmp()** and **strncmp()** respectively, described below. **strcasecmp()** and **strncasecmp()** assume the ASCII character set and ignore differences in case when comparing lower and upper case characters.

strcat() appends a copy of string *src*, including the terminating null character, to the end of string *dst*. **strncat()** appends at most *n* characters. Each returns a pointer to the null-terminated result. The initial character of *src* overrides the null character at the end of *dst*.

strchr( ) returns a pointer to the first occurrence of *c* (converted to a **char**) in string *s*, or a null pointer if *c* does not occur in the string. **strrchr( )** returns a pointer to the last occurrence of *c*. The null character terminating a string is considered to be part of the string.

strcmp( ) compares two strings byte-by-byte, according to the ordering of your machine's character set. The function returns an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2* respectively. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes that differ in the strings being compared. **strncmp( )** makes the same comparison but looks at a maximum of *n* bytes. Bytes following a null byte are not compared.

strcpy( ) copies string *src* to *dst* including the terminating null character, stopping after the null character has been copied. **strncpy( )** copies exactly *n* bytes, truncating *src* or adding null characters to *dst* if necessary. The result will not be null-terminated if the length of *src* is *n* or more. Each function returns *dst*.

strcspn( ) returns the length of the initial segment of string *s1* that consists entirely of characters not from string *s2*. **strspn( )** returns the length of the initial segment of string *s1* that consists entirely of characters from string *s2*.

strdup( ) returns a pointer to a new string that is a duplicate of the string pointed to by *s1*. The space for the new string is obtained using **malloc**(3C). If the new string cannot be created, a null pointer is returned.

strlen( ) returns the number of bytes in *s*, not including the terminating null character.

strpbrk( ) returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a null pointer if no character from *s2* exists in *s1*.

strstr( ) locates the first occurrence of the string *s2* (excluding the terminating null character) in string *s1*. **strstr( )** returns a pointer to the located string, or a null pointer if the string is not found. If *s2* points to a string with zero length (that is, the string **""**), the function returns *s1*.

strtok( ) can be used to break the string pointed to by *s1* into a sequence of tokens, each of which is delimited by one or more characters from the string pointed to by *s2*. **strtok( )** considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and will have written a null character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that subsequent calls (which must be made with the first argument being a null pointer) will work through the string *s1* immediately following that token. In this way subsequent calls will work through the string *s1* until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a null pointer is returned.

strtok_r( ) has the same functionality as **strtok( )** except that a pointer to a string placeholder *lasts* must be supplied by the caller. The **lasts** pointer is to keep track of the next substring in which to search for the next token.

SEE ALSO

malloc(3C), **setlocale**(3C), **strxfrm**(3C)

NOTES

The **strtok_r( )** interface is as proposed in the POSIX.4a Draft #6 document, and is subject to change to be compliant to the standard when it is accepted.

When compiling multi-thread applications, the **_REENTRANT** flag must be defined on the compile line. This flag should only be used in multi-thread applications.

All of these functions assume the default locale ''C.'' For some locales, **strxfrm( )** should be applied to the strings before they are passed to the functions.

strtok( ) is unsafe in multi-thread applications. **strtok_r( )** should be used instead.

string( ), **strcasecmp( )**, **strcat( )**, **strchr( )**, **strcmp( )**, **strcpy( )**, **strcspn( )**, **strdup( )**, **strlen( ), strncasecmp( ), strncat( ), strncmp( ), strncpy( ), strpbrk( ), strrchr( )**, **strspn( )**, and **strstr( )**, are MT-Safe in multi-thread applications.