strncat, strncat_s

Defined in header <string.h>

`char *strncat( char *dest, const char *src, size_t count );`	(1)	(until C99)
`char *strncat( char *restrict dest, const char *restrict src, size_t count );`		(since C99)
`errno_t strncat_s(char *restrict dest, rsize_t destsz, const char *restrict src, rsize_t count);`	(2)	(since C11)

1) Appends at most count characters from the character array pointed to by src, stopping if the null character is found, to the end of the null-terminated byte string pointed to by dest. The character `src[0]` replaces the null terminator at the end of dest. The terminating null character is always appended in the end (so the maximum number of bytes the function may write is `count+1`).

The behavior is undefined if the destination array does not have enough space for the contents of both dest and the first count characters of src, plus the terminating null character. The behavior is undefined if the source and destination objects overlap. The behavior is undefined if either dest is not a pointer to a null-terminated byte string or src is not a pointer to a character array,

2) Same as (1), except that this function may clobber the remainder of the destination array (from the last byte written to destsz) and that the following errors are detected at runtime and call the currently installed constraint handler function:

- src or dest is a null pointer
- destsz or count is zero or greater than RSIZE_MAX
- there is no null character in the first destsz bytes of dest
- truncation would occur: count or the length of src, whichever is less, exceeds the space available between the null terminator of dest and destsz.
- overlap would occur between the source and the destination strings

The behavior is undefined if the size of the character array pointed to by dest < `strnlen(dest,destsz)+strnlen(src,count)+1` < destsz; in other words, an erroneous value of destsz does not expose the impending buffer overflow. The behavior is undefined if the size of the character array pointed to by src < `strnlen(src,count)` < destsz; in other words, an erroneous value of count does not expose the impending buffer overflow.

As with all bounds-checked functions, strncat_s is only guaranteed to be available if `__STDC_LIB_EXT1__` is defined by the implementation and if the user defines `__STDC_WANT_LIB_EXT1__` to the integer constant 1 before including string.h.

Parameters

dest	-	pointer to the null-terminated byte string to append to
src	-	pointer to the character array to copy from
count	-	maximum number of characters to copy
destsz	-	the size of the destination buffer

Return value

1) returns a copy of dest

2) returns zero on success, returns non-zero on error. Also, on error, writes zero to `dest[0]` (unless dest is a null pointer or destsz is zero or greater than RSIZE_MAX).

Notes

Because strncat needs to seek to the end of dest on each call, it is inefficient to concatenate many strings into one using strncat.

Although truncation to fit the destination buffer is a security risk and therefore a runtime constraints violation for strncat_s, it is possible to get the truncating behavior by specifying count equal to the size of the destination array minus one: it will copy the first count bytes and append the null terminator as always: `strncat_s(dst, sizeof dst, src, (sizeof dst)-strnlen_s(dst, sizeof dst)-1);`

Example

Run this code

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char str[50] = "Hello ";
    char str2[50] = "World!";
    strcat(str, str2);
    strncat(str, " Goodbye World!", 3);
    puts(str);

#ifdef __STDC_LIB_EXT1__
    set_constraint_handler_s(ignore_handler_s);
    char s1[100] = "good";
    char s5[1000] = "bye";
    int r1 = strncat_s(s1, 100, s5, 1000); // r1 is 0, s1 holds "goodbye\0"
    printf("s1 = %s, r1 = %d\n", s1, r1);
    char s2[6] = "hello";
    int r2 = strncat_s(s2, 6, "", 1); // r2 is 0, s2 holds "hello\0"
    printf("s2 = %s, r2 = %d\n", s2, r2);
    char s3[6] = "hello";
    int r3 = strncat_s(s3, 6, "X", 2); // r3 is non-zero, s3 holds "\0"
    printf("s3 = %s, r3 = %d\n", s3, r3);
    // the strncat_s truncation idiom:
    char s4[7] = "abc";
    int r4 = strncat_s(s4, 7, "defghijklmn", 3); // r is 0, s4 holds "abcdef\0"
    printf("s4 = %s, r4 = %d\n", s4, r4);
#endif
}
```

Possible output:

```
Hello World! Go
s1 = goodbye, r1 = 0
s2 = hello, r2 = 0
s3 = , r3 = 22
s4 = abcdef, r4 = 0
```

References

- C11 standard (ISO/IEC 9899:2011):
  - 7.24.3.2 The strncat function (p: 364-365)
  - K.3.7.2.2 The strncat_s function (p: 618-620)
- C99 standard (ISO/IEC 9899:1999):
  - 7.21.3.2 The strncat function (p: 327-328)
- C89/C90 standard (ISO/IEC 9899:1990):
  - 4.11.3.2 The strncat function

See also

strcat **strcat_s** (C11)	concatenates two strings (function)
strcpy **strcpy_s** (C11)	copies one string to another (function)
memccpy (C23)	copies one buffer to another, stopping after the specified delimiter (function)
C++ documentation for **strncat**	

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=c/string/byte/strncat&oldid=124077"