strncpy, strncpy_s

Defined in header `<string.h>`

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

1) Copies at most count characters of the character array pointed to by src (including the terminating null character, but not any of the characters that follow the null character) to character array pointed to by dest.

If count is reached before the entire array src was copied, the resulting character array is not null-terminated.

If, after copying the terminating null character from src, count is not reached, additional null characters are written to dest until the total of count characters have been written.

The behavior is undefined if the character arrays overlap, if either dest or src is not a pointer to a character array (including if dest or src is a null pointer), if the size of the array pointed to by dest is less than count, or if the size of the array pointed to by src is less than count and it does not contain a null character.

2) Same as (1), except that the function does not continue writing zeroes into the destination array to pad up to count, it stops after writing the terminating null character (if there was no null in the source, it writes one at `dest[count]` and then stops). Also, the following errors are detected at runtime and call the currently installed constraint handler function:

- src or dest is a null pointer
- destsz is zero or greater than RSIZE_MAX
- count is greater than RSIZE_MAX
- count is greater or equal destsz, but destsz is less or equal `strnlen_s(src, count)`, in other words, truncation would occur
- 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_s(src, destsz)` <= 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_s(src, count)` < destsz; in other words, an erroneous value of count does not expose the impending buffer overflow.

As with all bounds-checked functions, strncpy_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 character array to copy 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) and may clobber the rest of the destination array with unspecified values.

Notes

As corrected by the post-C11 DR 468, strncpy_s, unlike strcpy_s, is only allowed to clobber the remainder of the destination array if an error occurs.

Unlike strncpy, strncpy_s does not pad the destination array with zeroes, This is a common source of errors when converting existing code to the bounds-checked version.

Although truncation to fit the destination buffer is a security risk and therefore a runtime constraints violation for strncpy_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: `strncpy_s(dst, sizeof dst, src, (sizeof dst)-1);`

Example

Run this code

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

int main(void)
{
    char src[] = "hi";
    char dest[6] = "abcdef"; // no null terminator
    strncpy(dest, src, 5); // writes five characters 'h', 'i', '\0', '\0', '\0' to dest
    printf("strncpy(dest, src, 5) to a 6-byte dest gives : ");
    for (size_t n = 0; n < sizeof dest; ++n) {
        char c = dest[n];
        c ? printf("'%c' ", c) : printf("'\\0' ");
    }

    printf("\nstrncpy(dest2, src, 2) to a 2-byte dst gives : ");
    char dest2[2];
    strncpy(dest2, src, 2); // truncation: writes two characters 'h', 'i', to dest2
    for (size_t n = 0; n < sizeof dest2; ++n) {
        char c = dest2[n];
        c ? printf("'%c' ", c) : printf("'\\0' ");
    }
    printf("\n");

#ifdef __STDC_LIB_EXT1__
    set_constraint_handler_s(ignore_handler_s);
    char dst1[6], src1[100] = "hello";
    errno_t r1 = strncpy_s(dst1, 6, src1, 100);  // writes 0 to r1, 6 characters to dst1
    printf("dst1 = \"%s\", r1 = %d\n", dst1,r1); // 'h','e','l','l','o','\0' to dst1

    char dst2[5], src2[7] = {'g','o','o','d','b','y','e'};
    errno_t r2 = strncpy_s(dst2, 5, src2, 7);    // copy overflows the destination array
    printf("dst2 = \"%s\", r2 = %d\n", dst2,r2); // writes nonzero to r2,'\0' to dst2[0]

    char dst3[5];
    errno_t r3 = strncpy_s(dst3, 5, src2, 4);    // writes 0 to r3, 5 characters to dst3
    printf("dst3 = \"%s\", r3 = %d\n", dst3,r3); // 'g', 'o', 'o', 'd', '\0' to dst3
#endif
}
```

Possible output:

```
strncpy(dest, src, 5) to a 6-byte dst gives : 'h' 'i' '\0' '\0' '\0' 'f'
strncpy(dest2, src, 2) to a 2-byte dst gives : 'h' 'i'
dst1 = "hello", r1 = 0
dst2 = "", r2 = 22
dst3 = "good", r3 = 0
```

References

- C11 standard (ISO/IEC 9899:2011):
  - 7.24.2.4 The strncpy function (p: 363-364)
  - K.3.7.1.4 The strncpy_s function (p: 616-617)
- C99 standard (ISO/IEC 9899:1999):
  - 7.21.2.4 The strncpy function (p: 326-327)
- C89/C90 standard (ISO/IEC 9899:1990):

- 4.11.2.4 The strncpy function

See also

strcpy **strcpy_s** (C11)	copies one string to another (function)
memcpy **memcpy_s** (C11)	copies one buffer to another (function)
strndup (dynamic memory TR)	allocate a copy of a string up to specified size (function)
C++ documentation for **strncpy**	