C 문자 배열 <cstring>

다른거임

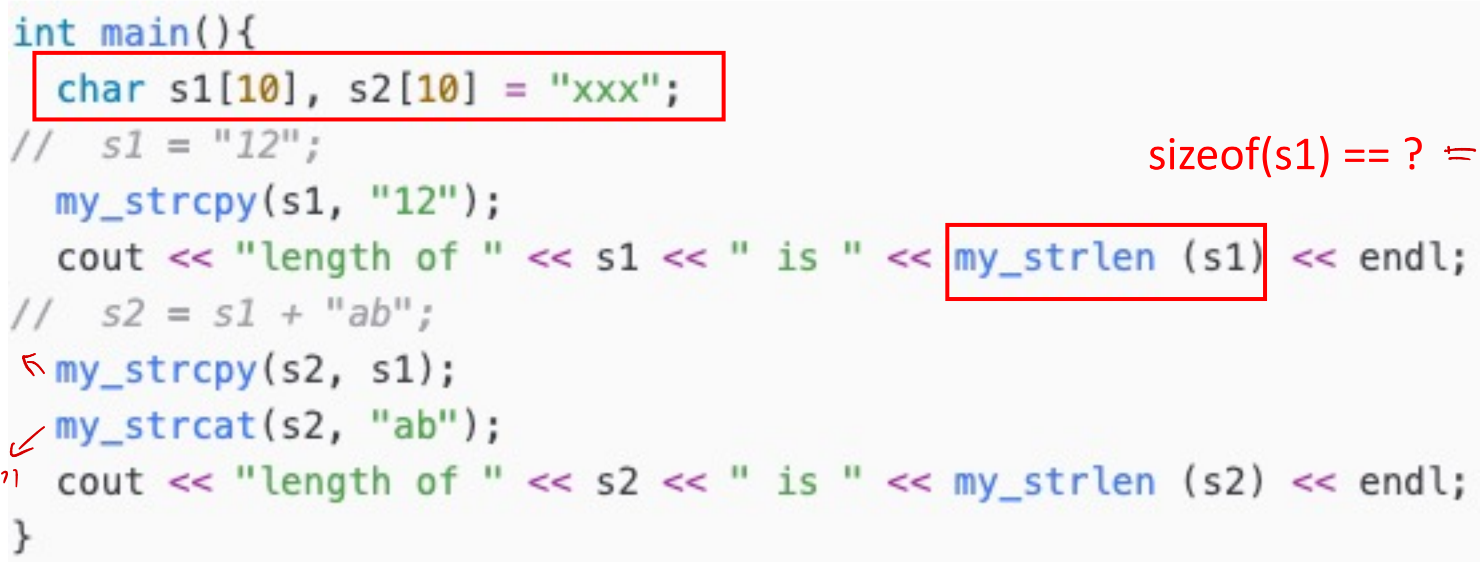
vs.

C++ string class <string>

C++ 에서 사용

국민대학교 임은진

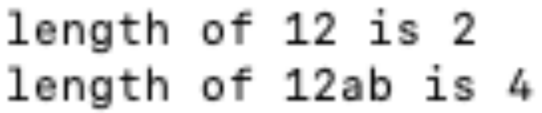
문자열은 마지막 문자가 '\0' 인 문자배열이다.



```
int main(){
  char s1[10], s2[10] = "xxx";
//  s1 = "12";
  my_strcpy(s1, "12");
  cout << "length of " << s1 << " is " << my_strlen (s1) << endl;
//  s2 = s1 + "ab";
  my_strcpy(s2, s1);
  my_strcat(s2, "ab");
  cout << "length of " << s2 << " is " << my_strlen (s2) << endl;
}
```

복사하기

붙여쓰기

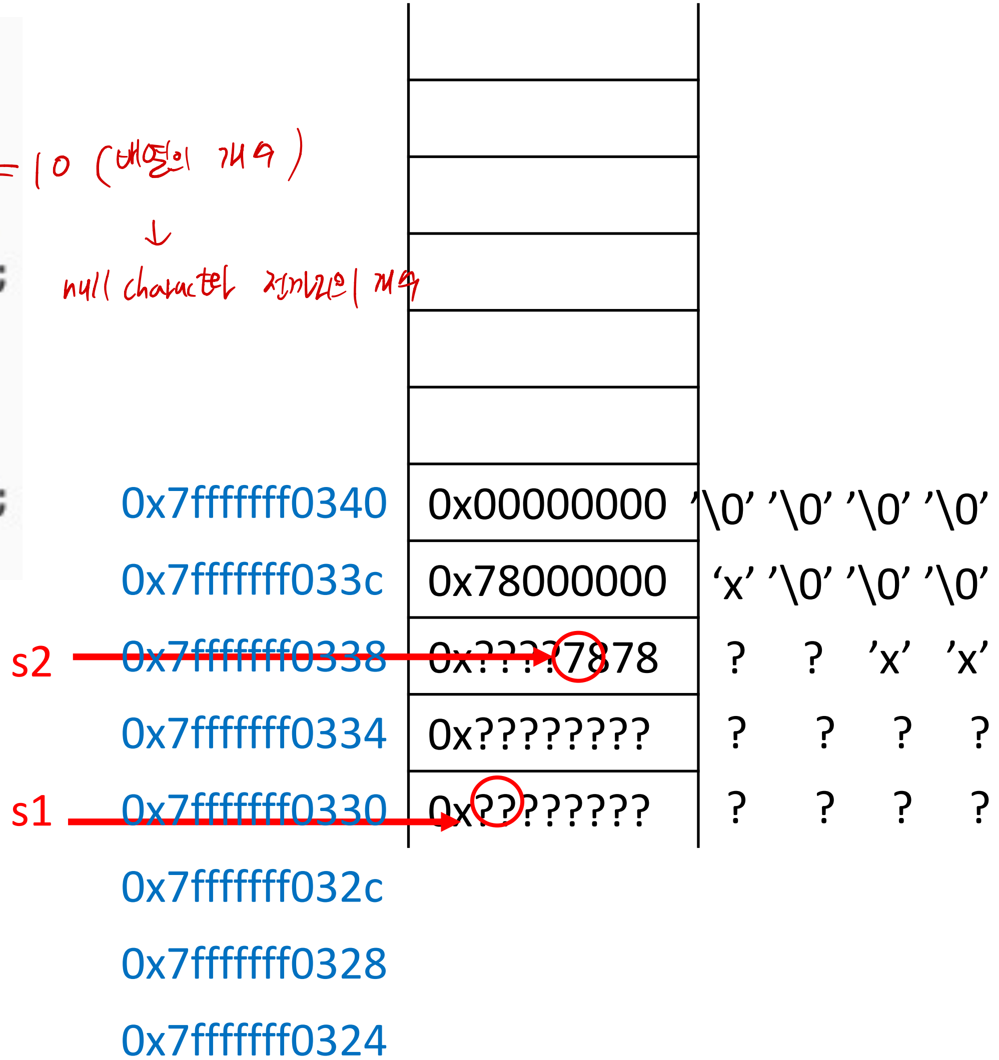


```
length of 12 is 2
length of 12ab is 4
```

sizeof(s1) == ? = 10 (배열의 개수)

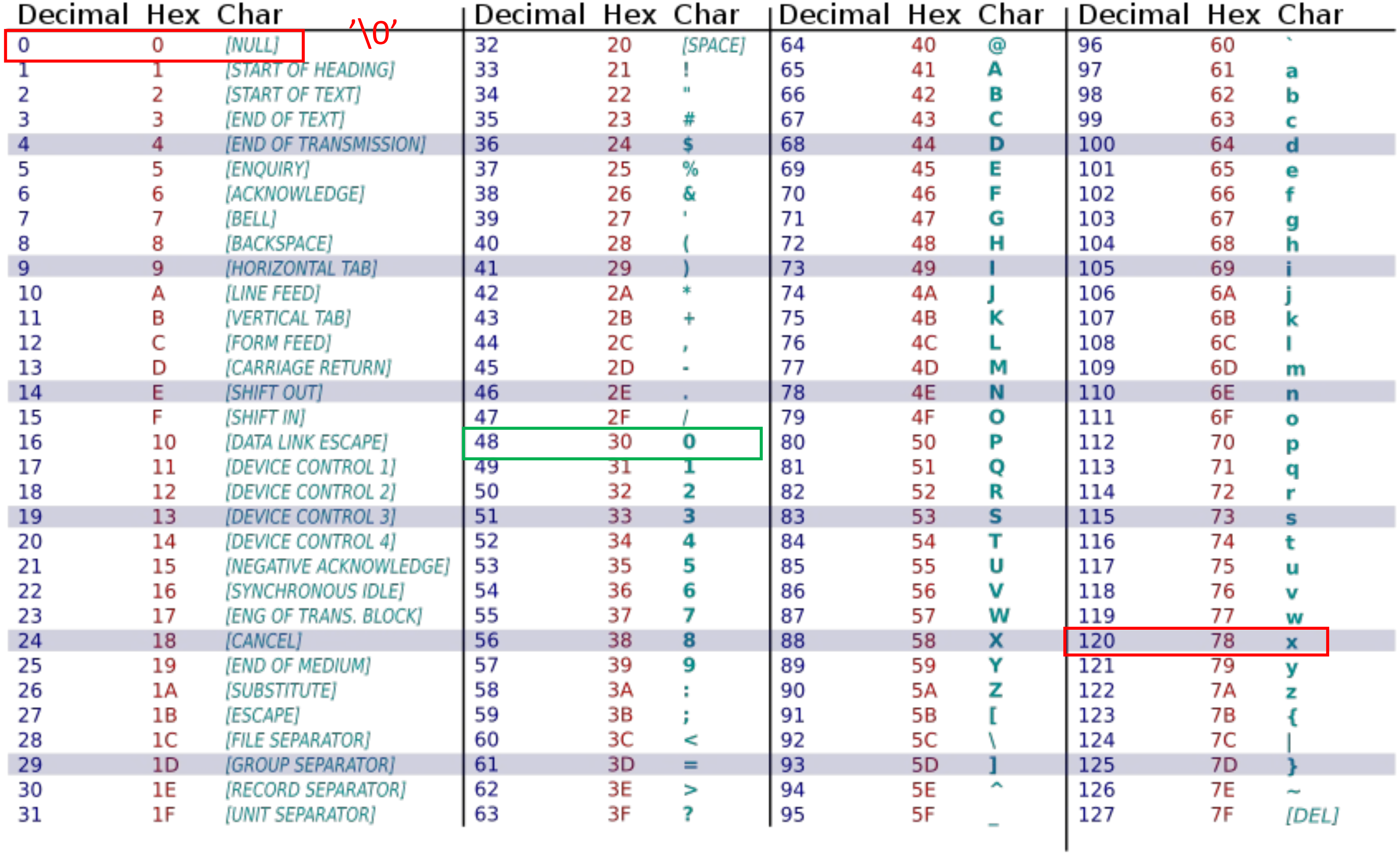
↓

null character 전까지의 개수



0x7fffffff0340	0x00000000	'\0' '\0' '\0' '\0'
0x7fffffff033c	0x78000000	'x' '\0' '\0' '\0'
0x7fffffff0338	0x????7878	? ? 'x' 'x'
0x7fffffff0334	0x????????	? ? ? ?
0x7fffffff0330	0x????????	? ? ? ?
0x7fffffff032c		
0x7fffffff0328		
0x7fffffff0324		

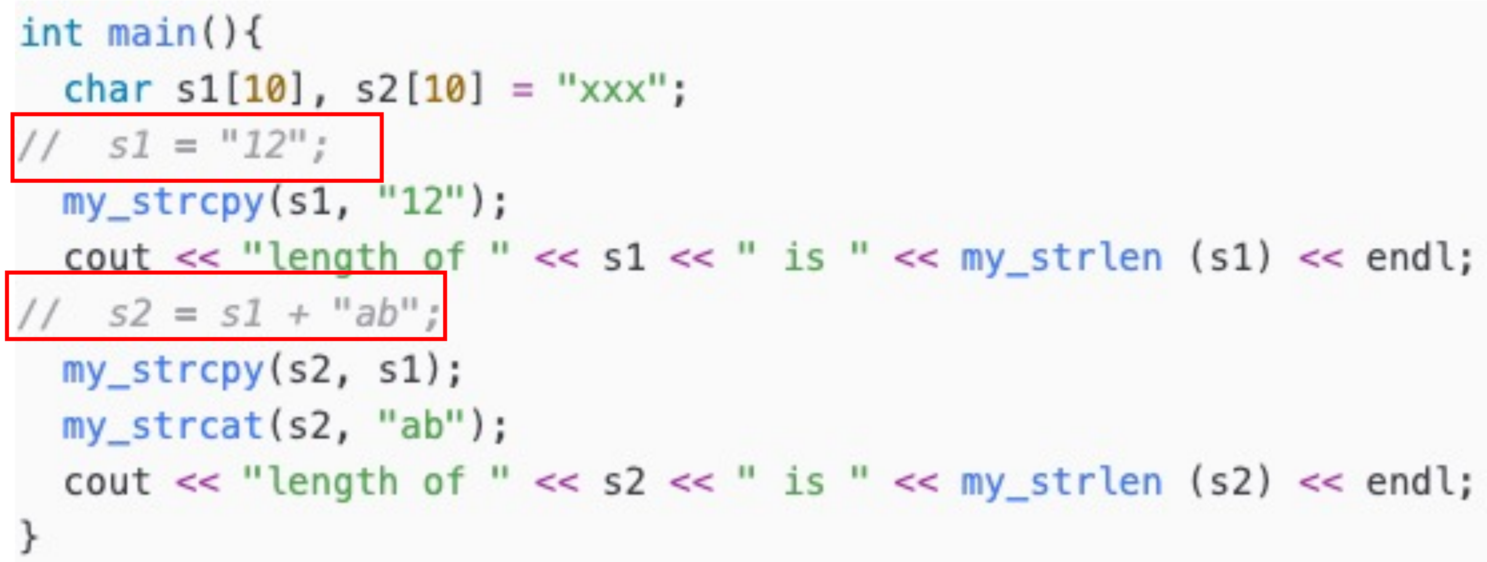
ASCII TABLE



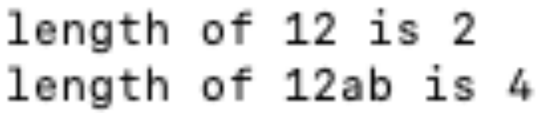
’\0’

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

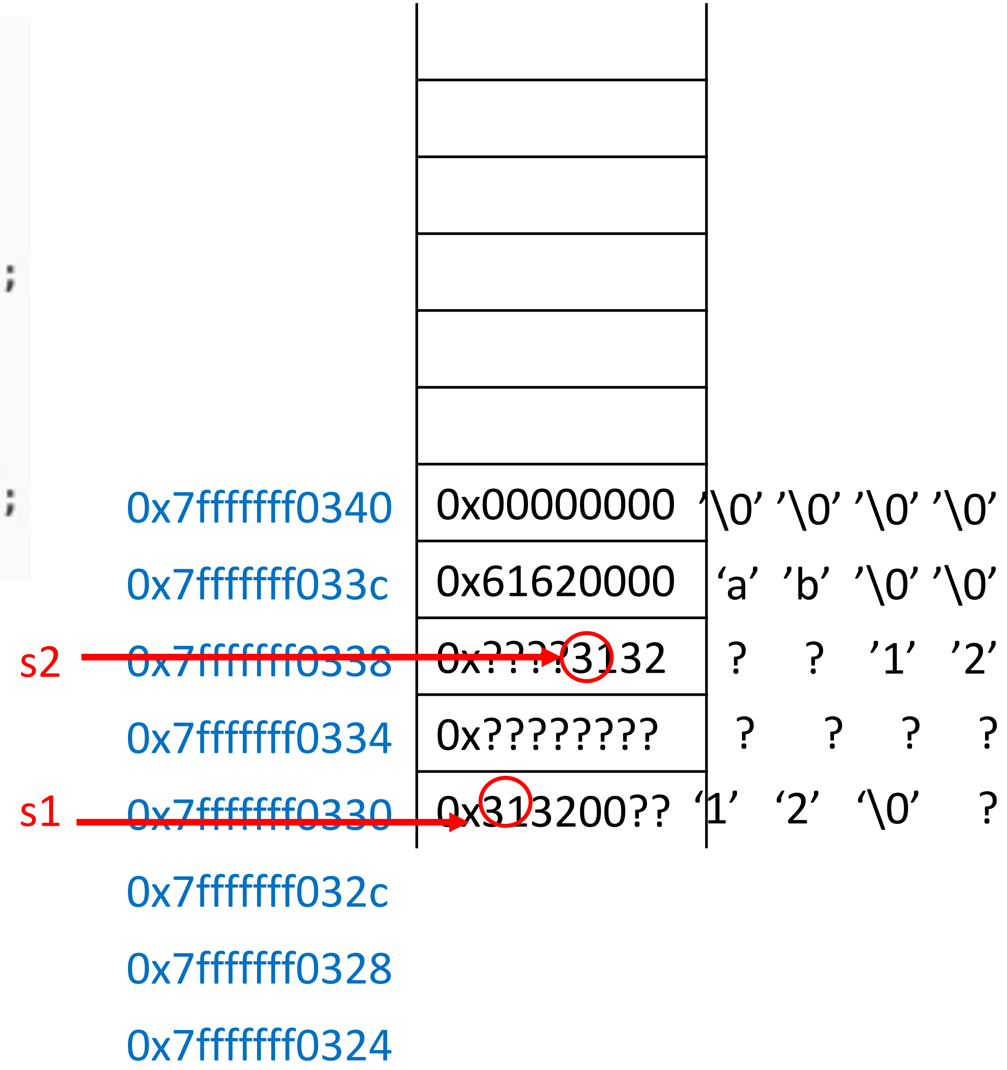
문자열의 연산은 불가능하므로 함수로 구현해야 한다.



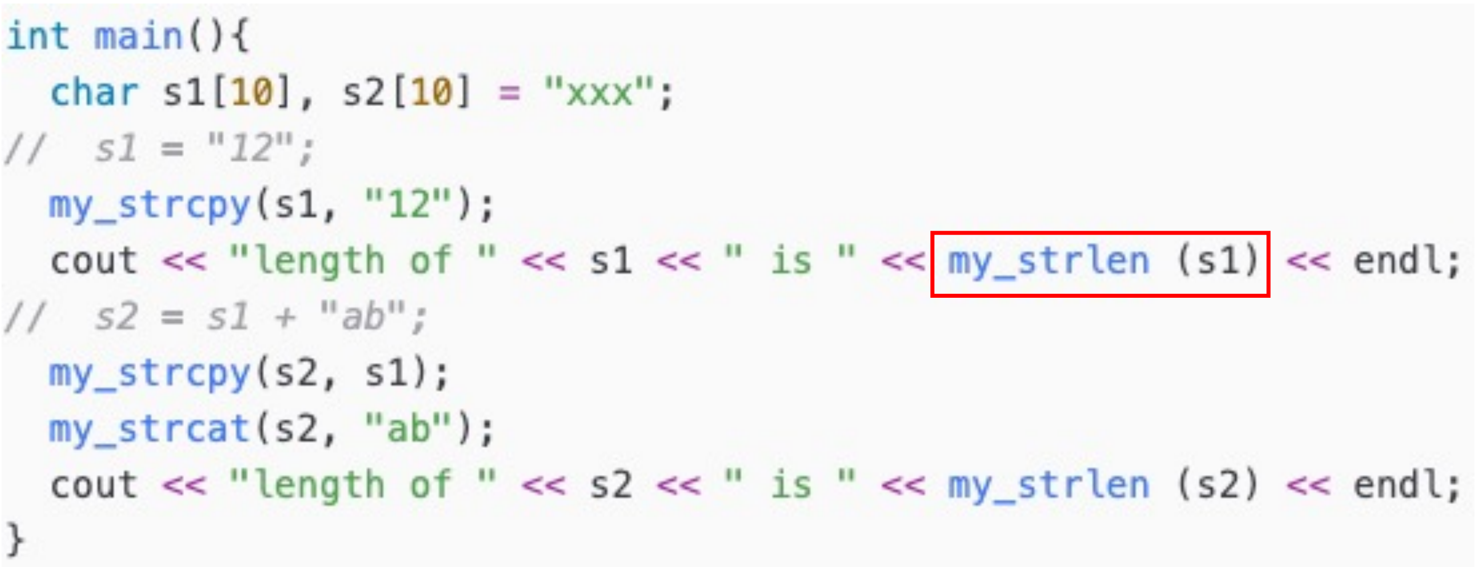
```
int main(){
  char s1[10], s2[10] = "xxx";
//  s1 = "12";
  my_strcpy(s1, "12");
  cout << "length of " << s1 << " is " << my_strlen (s1) << endl;
//  s2 = s1 + "ab";
  my_strcpy(s2, s1);
  my_strcat(s2, "ab");
  cout << "length of " << s2 << " is " << my_strlen (s2) << endl;
}
```



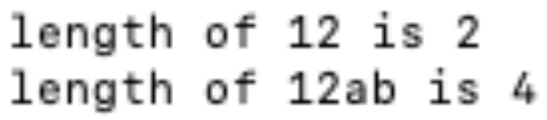
```
length of 12 is 2
length of 12ab is 4
```



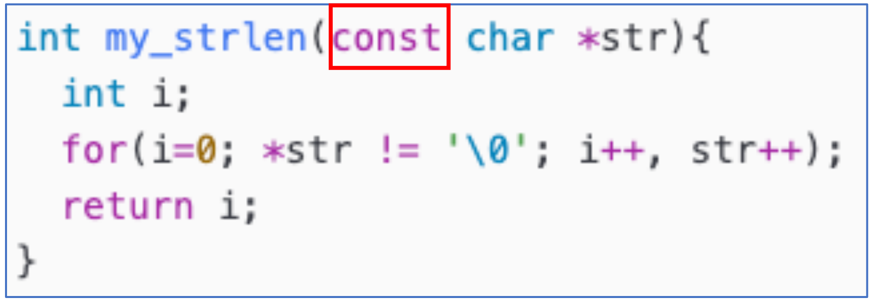
	0x7ffffff0340	0x00000000	'\0' '\0' '\0' '\0'
	0x7ffffff033c	0x61620000	'a' 'b' '\0' '\0'
s2	0x7ffffff0338	0x????3132	? ? '1' '2'
	0x7ffffff0334	0x????????	? ? ? ?
s1	0x7ffffff0330	0x313200??	'1' '2' '\0' ?
	0x7ffffff032c		
	0x7ffffff0328		
	0x7ffffff0324		



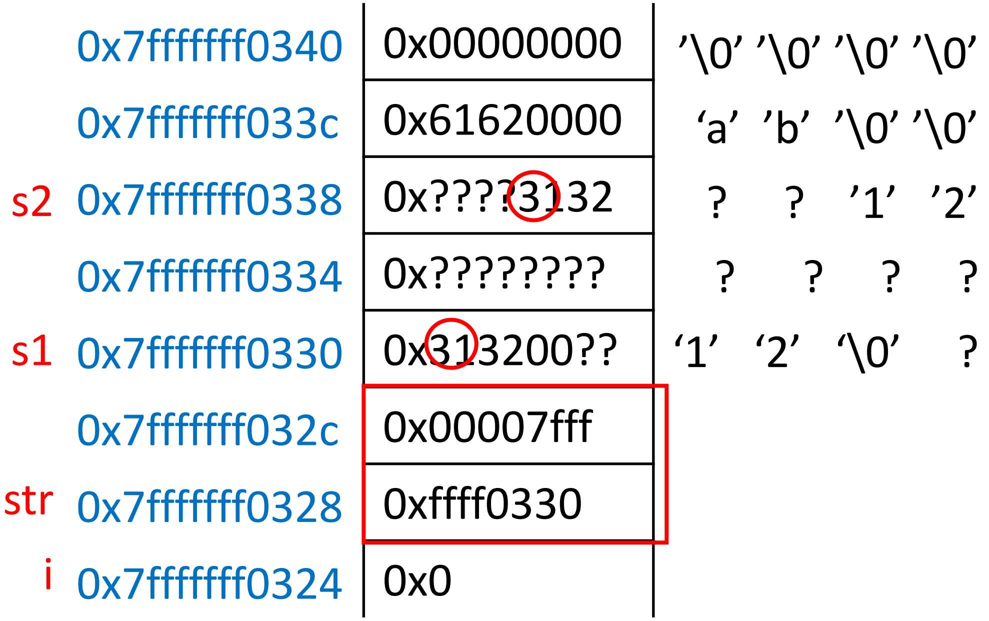
```
int main(){
  char s1[10], s2[10] = "xxx";
//  s1 = "12";
  my_strcpy(s1, "12");
  cout << "length of " << s1 << " is " << my_strlen (s1) << endl;
//  s2 = s1 + "ab";
  my_strcpy(s2, s1);
  my_strcat(s2, "ab");
  cout << "length of " << s2 << " is " << my_strlen (s2) << endl;
}
```



```
length of 12 is 2
length of 12ab is 4
```



```
int my_strlen(const char *str){
  int i;
  for(i=0; *str != '\0'; i++, str++);
  return i;
}
```

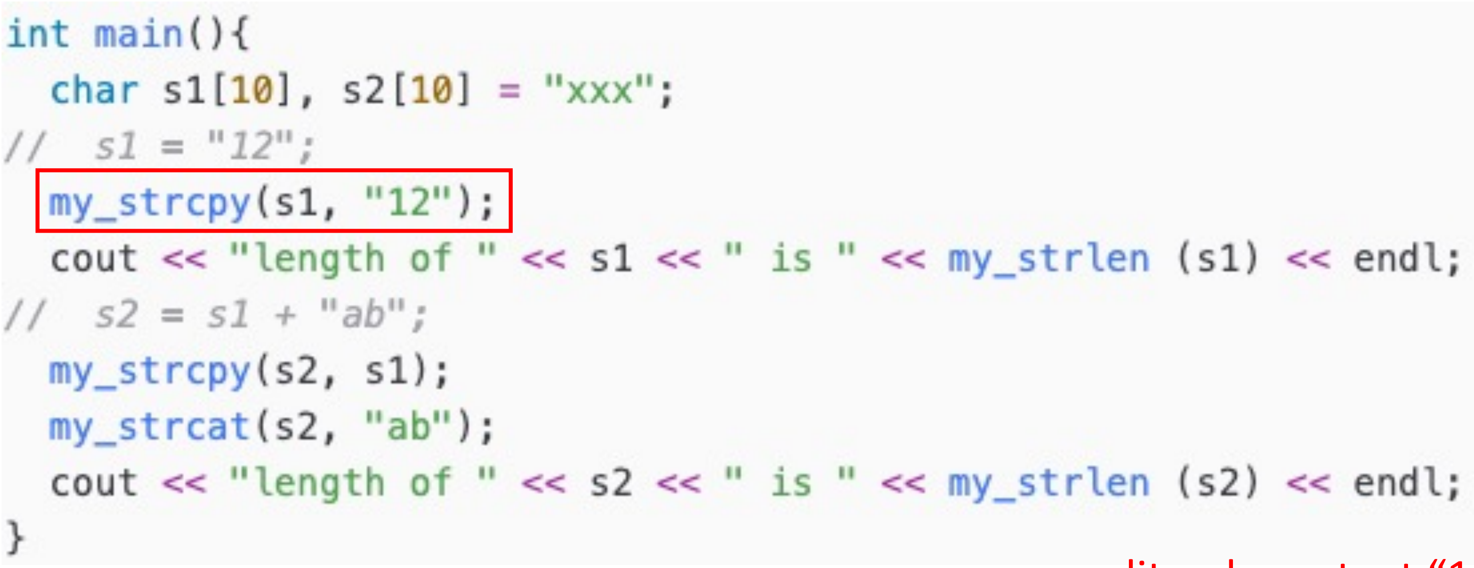


	0x7fffffff0340	0x00000000	’\0’ ’\0’ ’\0’ ’\0’
	0x7fffffff033c	0x61620000	‘a’ ’b’ ’\0’ ’\0’
s2	0x7fffffff0338	0x????3132	? ? ’1’ ’2’
	0x7fffffff0334	0x????????	? ? ? ?
s1	0x7fffffff0330	0x313200??	‘1’ ‘2’ ‘\0’ ?
	0x7fffffff032c	0x00007fff	
str	0x7fffffff0328	0xffff0330	
i	0x7fffffff0324	0x0	

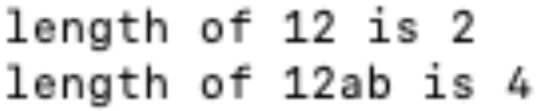
const 포인터

- const int *p1;
- p1은 const int에 대한 포인터이다. 즉 p1이 가리키는 내용이 상수가 된다.
- *p1 = 100;(X)

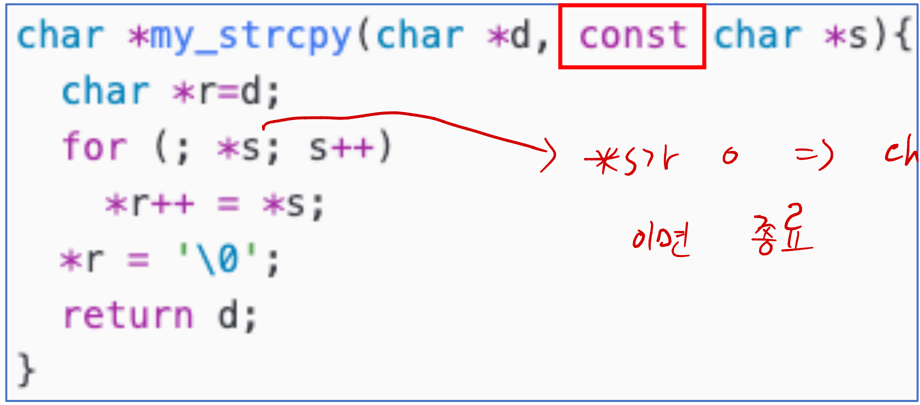
- int * const p2;
- 이번에는 정수를 가리키는 p2가 상수라는 의미이다. 즉 p2의 내용이 변경될 수 없다.
- p2 = p1; (X)



```
int main(){
  char s1[10], s2[10] = "xxx";
//  s1 = "12";
  my_strcpy(s1, "12");
  cout << "length of " << s1 << " is " << my_strlen (s1) << endl;
//  s2 = s1 + "ab";
  my_strcpy(s2, s1);
  my_strcat(s2, "ab");
  cout << "length of " << s2 << " is " << my_strlen (s2) << endl;
}
```



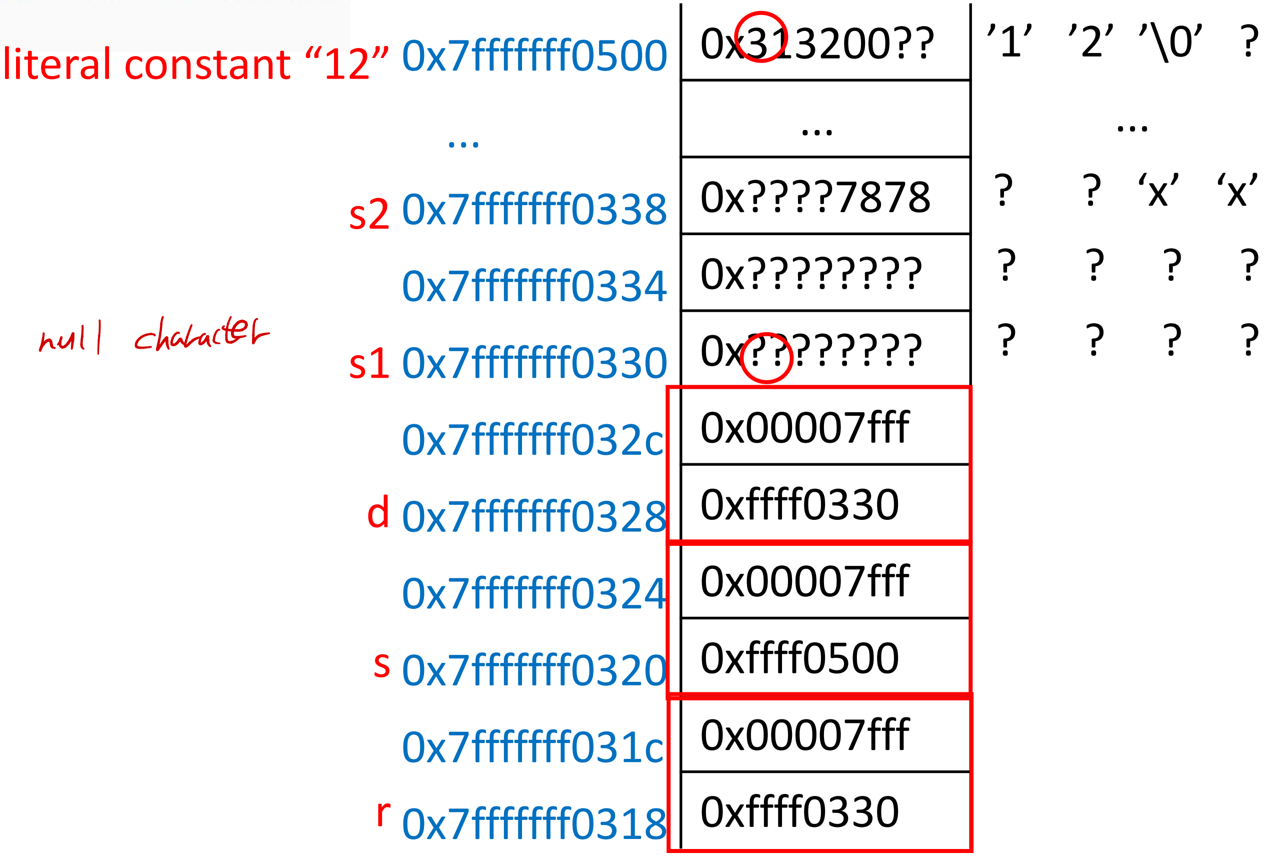
```
length of 12 is 2
length of 12ab is 4
```



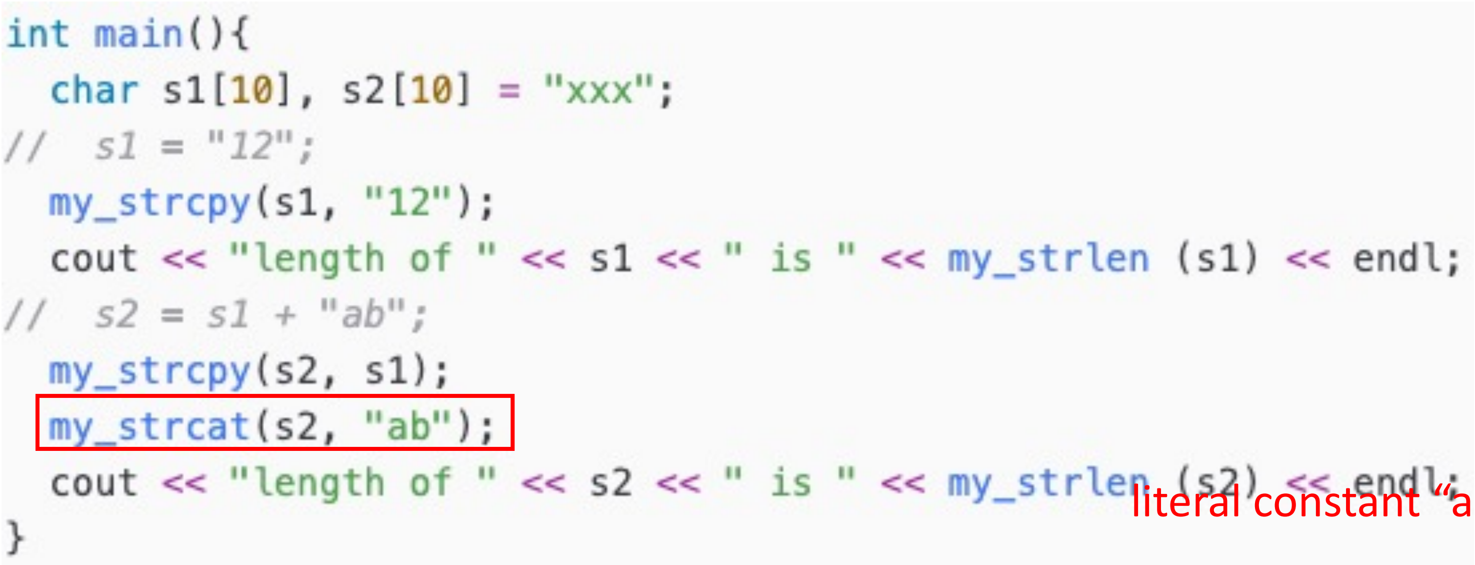
```
char *my_strcpy(char *d, const char *s){
  char *r=d;
  for (; *s; s++)
    *r++ = *s;
  *r = '\0';
  return d;
}
```

*s가 0 => char이 0 => null character

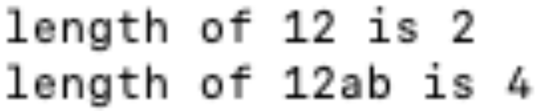
이면 종료



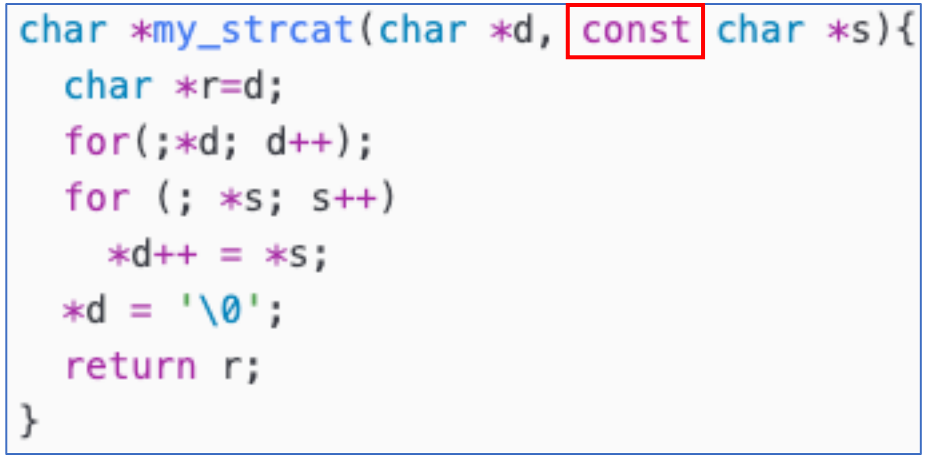
literal constant “12”	0x7fffffff0500	0x313200??	’1’ ’2’ ’\0’ ?
	…	…	…
s2	0x7fffffff0338	0x????7878	? ? ‘x’ ‘x’
	0x7fffffff0334	0x????????	? ? ? ?
s1	0x7fffffff0330	0x????????	? ? ? ?
	0x7fffffff032c	0x00007fff	
d	0x7fffffff0328	0xffff0330	
	0x7fffffff0324	0x00007fff	
s	0x7fffffff0320	0xffff0500	
	0x7fffffff031c	0x00007fff	
r	0x7fffffff0318	0xffff0330	



```
int main(){
  char s1[10], s2[10] = "xxx";
//  s1 = "12";
  my_strcpy(s1, "12");
  cout << "length of " << s1 << " is " << my_strlen (s1) << endl;
//  s2 = s1 + "ab";
  my_strcpy(s2, s1);
  my_strcat(s2, "ab");
  cout << "length of " << s2 << " is " << my_strlen (s2) << endl;
}
```

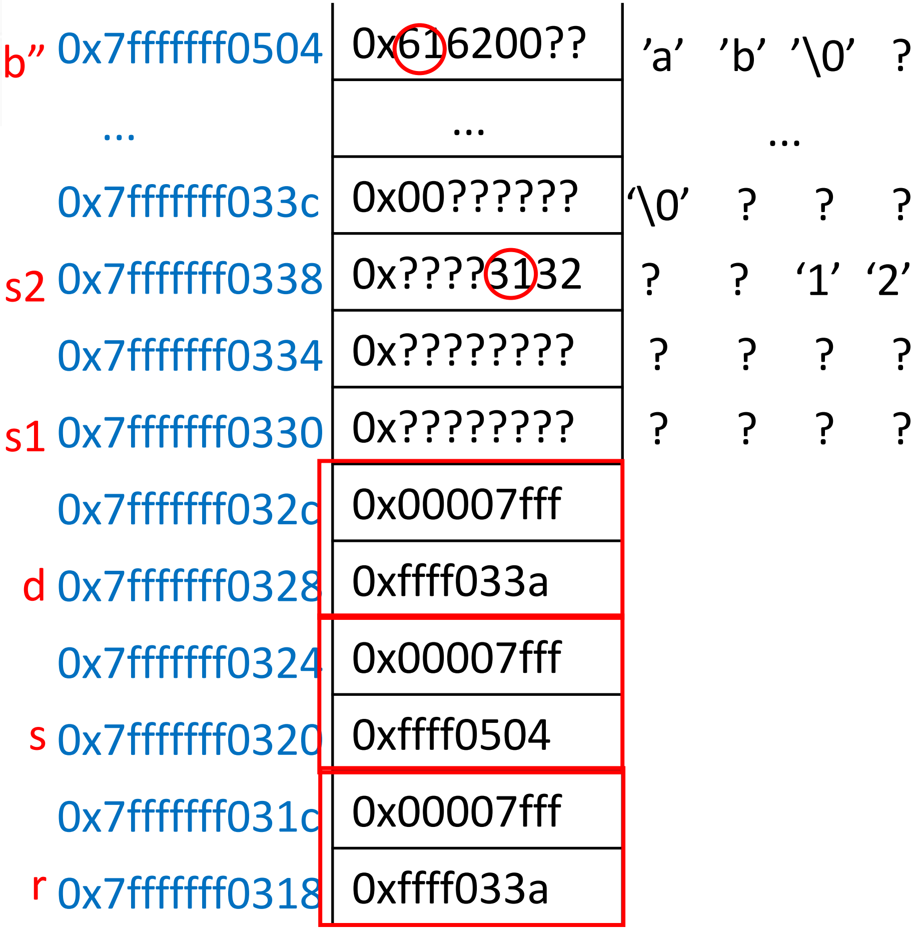


```
length of 12 is 2
length of 12ab is 4
```

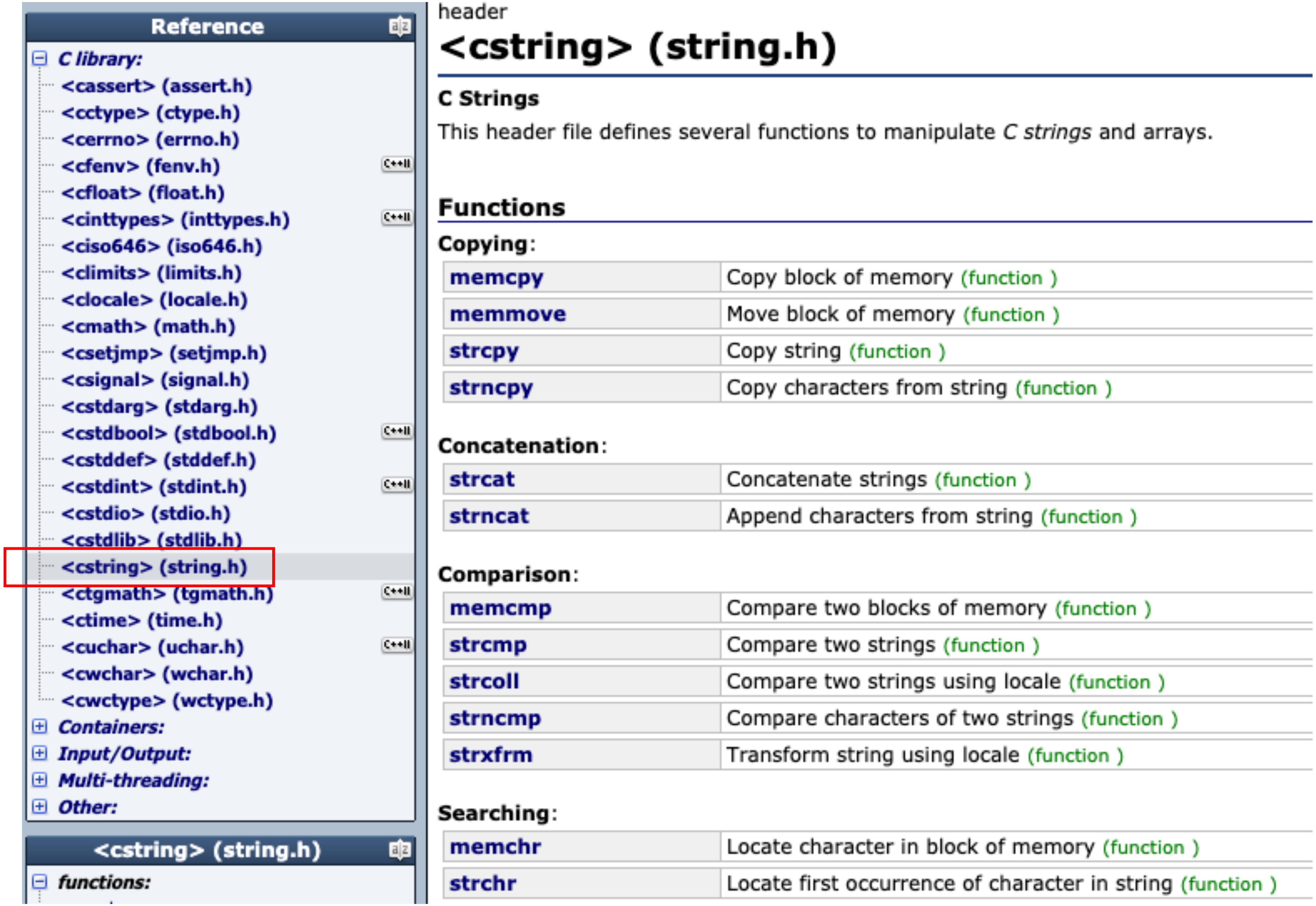


```
char *my_strcat(char *d, const char *s){
  char *r=d;
  for(;*d; d++);
  for (; *s; s++)
    *d++ = *s;
  *d = '\0';
  return r;
}
```

literal constant “ab”



“ab”	0x7fffffff0504	0x616200??	’a’	’b’	’\0’	?
	…	…		…		
	0x7fffffff033c	0x00??????	‘\0’	?	?	?
s2	0x7fffffff0338	0x????3132	?	?	‘1’	‘2’
	0x7fffffff0334	0x????????	?	?	?	?
s1	0x7fffffff0330	0x????????	?	?	?	?
	0x7fffffff032c	0x00007fff				
d	0x7fffffff0328	0xffff033a				
	0x7fffffff0324	0x00007fff				
s	0x7fffffff0320	0xffff0504				
	0x7fffffff031c	0x00007fff				
r	0x7fffffff0318	0xffff033a				



header

<cstring> (string.h)

C Strings

This header file defines several functions to manipulate *C strings* and arrays.

Functions

Copying:

memcpy	Copy block of memory (function)
memmove	Move block of memory (function)
strcpy	Copy string (function)
strncpy	Copy characters from string (function)

Concatenation:

strcat	Concatenate strings (function)
strncat	Append characters from string (function)

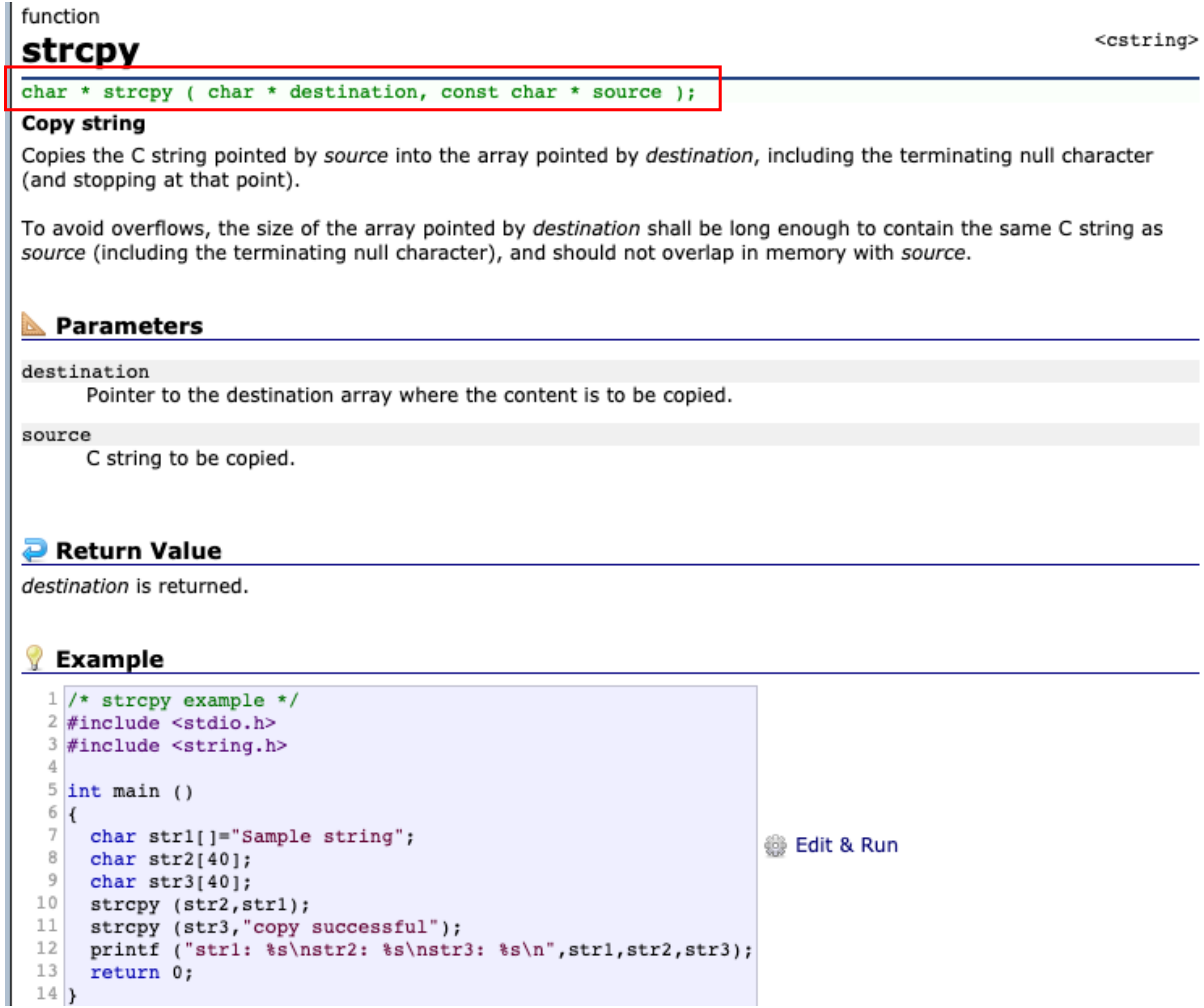
Comparison:

memcmp	Compare two blocks of memory (function)
strcmp	Compare two strings (function)
strcoll	Compare two strings using locale (function)
strncmp	Compare characters of two strings (function)
strxfrm	Transform string using locale (function)

Searching:

memchr	Locate character in block of memory (function)
strchr	Locate first occurrence of character in string (function)

```c
char *my_strcpy(char *d, const char *s){
  char *r=d;
  for (; *s; s++)
    *r++ = *s;
  *r = '\0';
  return d;
}
```



function

<cstring>

strcpy

```
char * strcpy ( char * destination, const char * source );
```

Copy string

Copies the C string pointed by *source* into the array pointed by *destination*, including the terminating null character (and stopping at that point).

To avoid overflows, the size of the array pointed by *destination* shall be long enough to contain the same C string as *source* (including the terminating null character), and should not overlap in memory with *source*.

Parameters

destination
: Pointer to the destination array where the content is to be copied.

source
: C string to be copied.

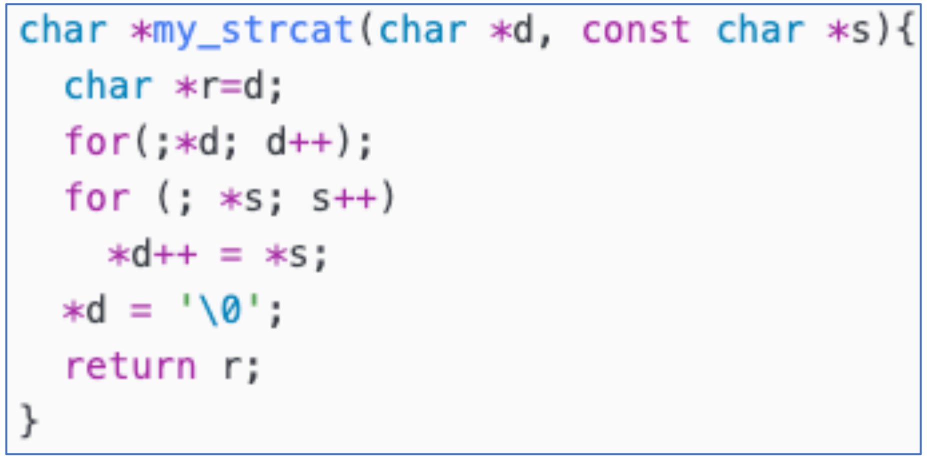
Return Value

destination is returned.

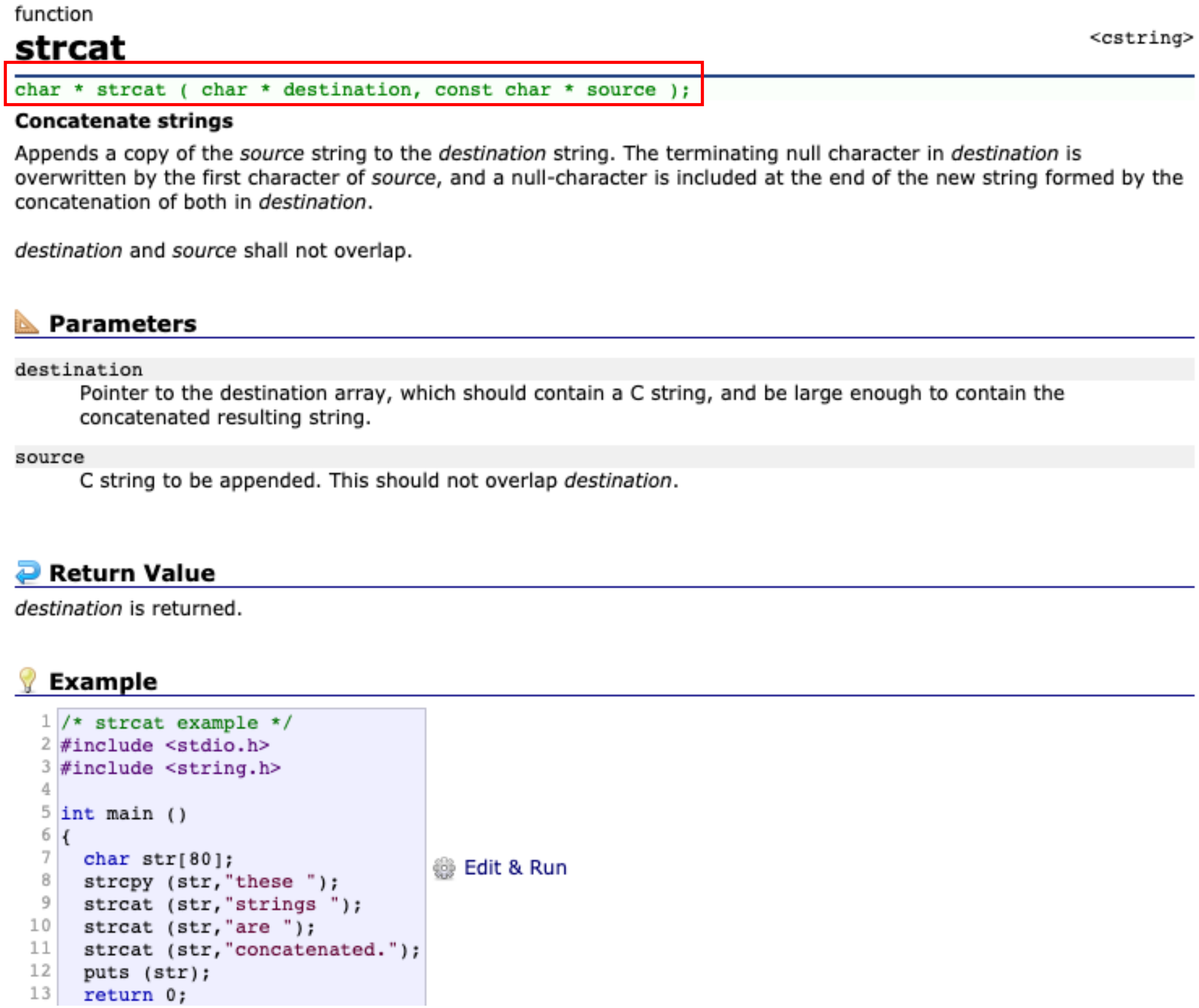
Example

```c
/* strcpy example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str1[]="Sample string";
  char str2[40];
  char str3[40];
  strcpy (str2,str1);
  strcpy (str3,"copy successful");
  printf ("str1: %s\nstr2: %s\nstr3: %s\n",str1,str2,str3);
  return 0;
}
```



```c
char *my_strcat(char *d, const char *s){
  char *r=d;
  for(;*d; d++);
  for (; *s; s++)
    *d++ = *s;
  *d = '\0';
  return r;
}
```



function

strcat

<cstring>

```c
char * strcat ( char * destination, const char * source );
```

Concatenate strings

Appends a copy of the *source* string to the *destination* string. The terminating null character in *destination* is overwritten by the first character of *source*, and a null-character is included at the end of the new string formed by the concatenation of both in *destination*.

destination and *source* shall not overlap.

Parameters

`destination`
: Pointer to the destination array, which should contain a C string, and be large enough to contain the concatenated resulting string.

`source`
: C string to be appended. This should not overlap *destination*.

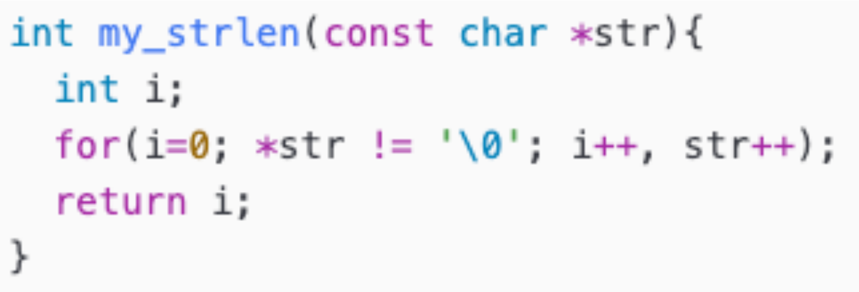
Return Value

destination is returned.

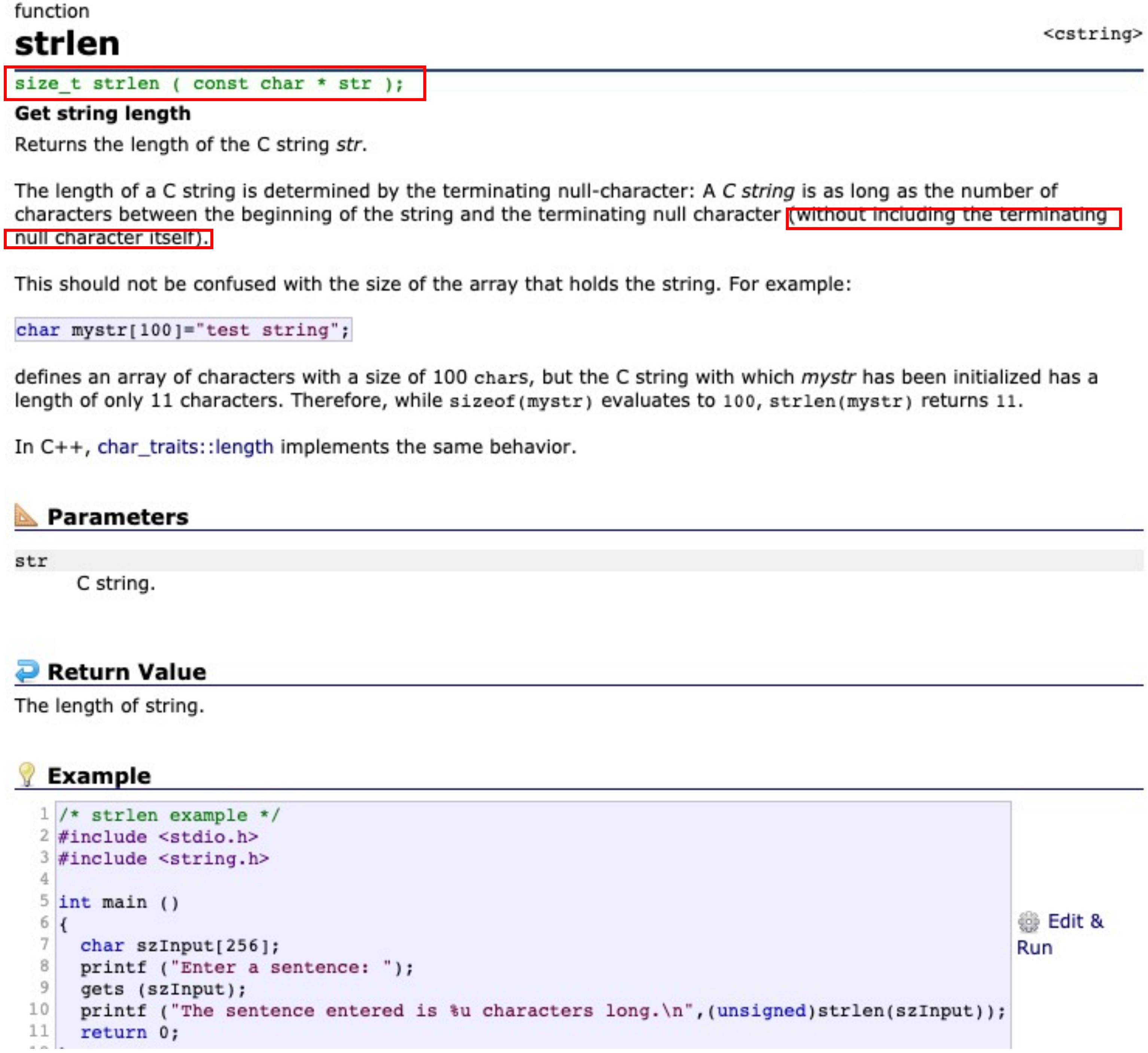
Example

```c
/* strcat example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[80];
  strcpy (str,"these ");
  strcat (str,"strings ");
  strcat (str,"are ");
  strcat (str,"concatenated.");
  puts (str);
  return 0;
```



```c
int my_strlen(const char *str){
  int i;
  for(i=0; *str != '\0'; i++, str++);
  return i;
}
```



function

strlen

<cstring>

```
size_t strlen ( const char * str );
```

Get string length

Returns the length of the C string *str*.

The length of a C string is determined by the terminating null-character: A *C string* is as long as the number of characters between the beginning of the string and the terminating null character (without including the terminating null character itself).

This should not be confused with the size of the array that holds the string. For example:

```c
char mystr[100]="test string";
```

defines an array of characters with a size of 100 `chars`, but the C string with which *mystr* has been initialized has a length of only 11 characters. Therefore, while `sizeof(mystr)` evaluates to `100`, `strlen(mystr)` returns `11`.

In C++, char_traits::length implements the same behavior.

Parameters

`str`

C string.

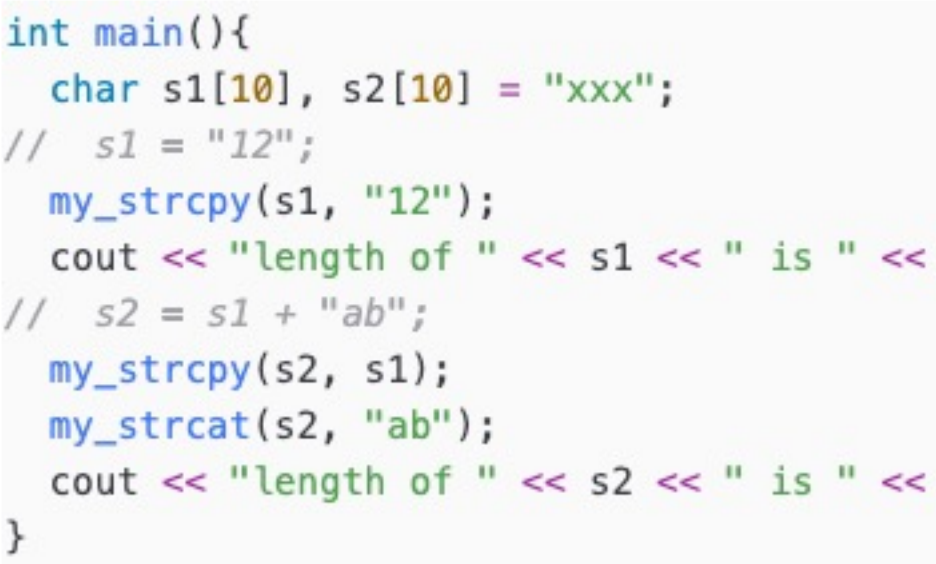
Return Value

The length of string.

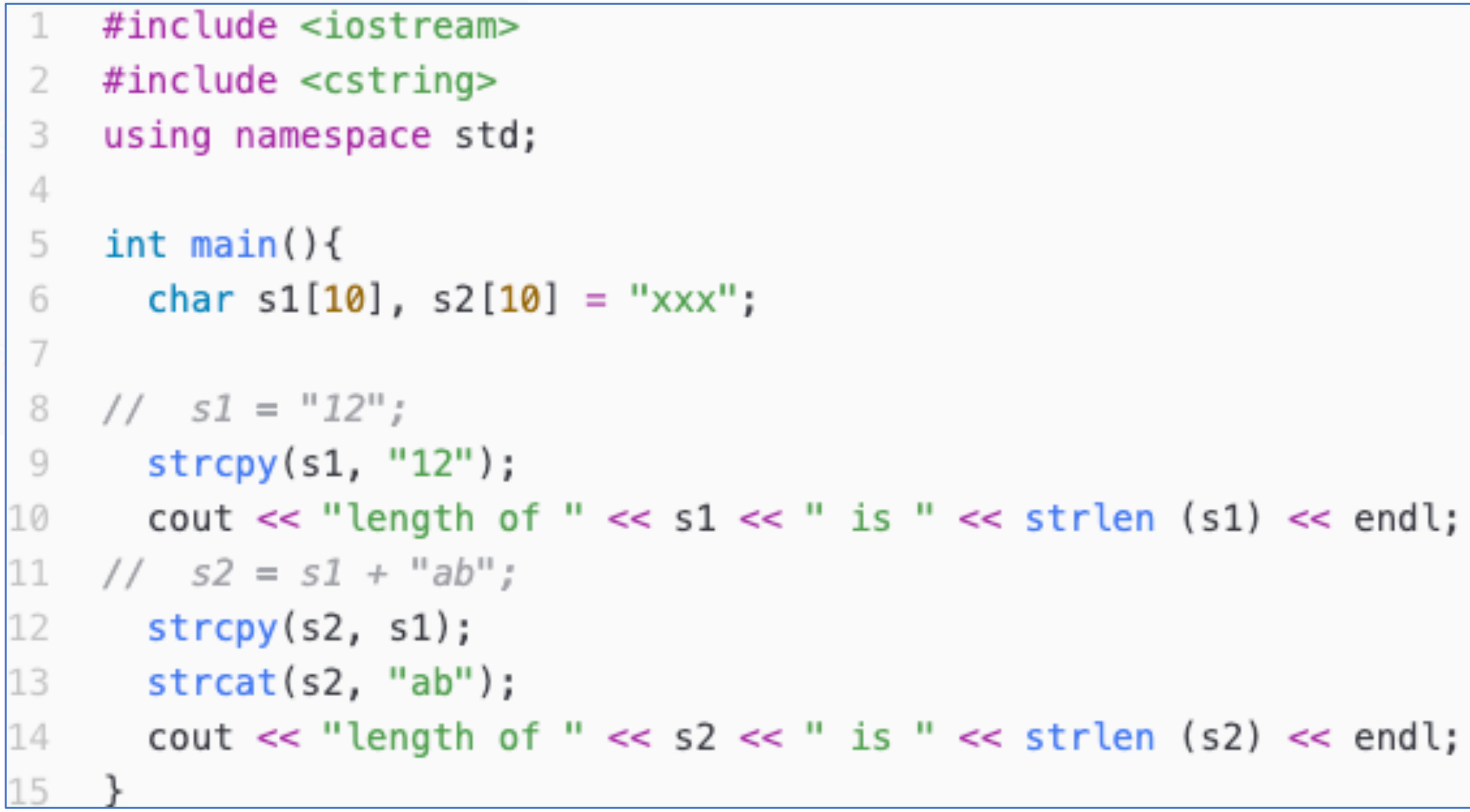
Example

```c
/* strlen example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char szInput[256];
  printf ("Enter a sentence: ");
  gets (szInput);
  printf ("The sentence entered is %u characters long.\n",(unsigned)strlen(szInput));
  return 0;
```



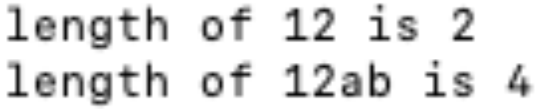
```cpp
int main(){
  char s1[10], s2[10] = "xxx";
//  s1 = "12";
  my_strcpy(s1, "12");
  cout << "length of " << s1 << " is " <<
//  s2 = s1 + "ab";
  my_strcpy(s2, s1);
  my_strcat(s2, "ab");
  cout << "length of " << s2 << " is " <<
}
```



```cpp
#include <iostream>
#include <cstring>
using namespace std;

int main(){
  char s1[10], s2[10] = "xxx";

//  s1 = "12";
  strcpy(s1, "12");
  cout << "length of " << s1 << " is " << strlen (s1) << endl;
//  s2 = s1 + "ab";
  strcpy(s2, s1);
  strcat(s2, "ab");
  cout << "length of " << s2 << " is " << strlen (s2) << endl;
}
```

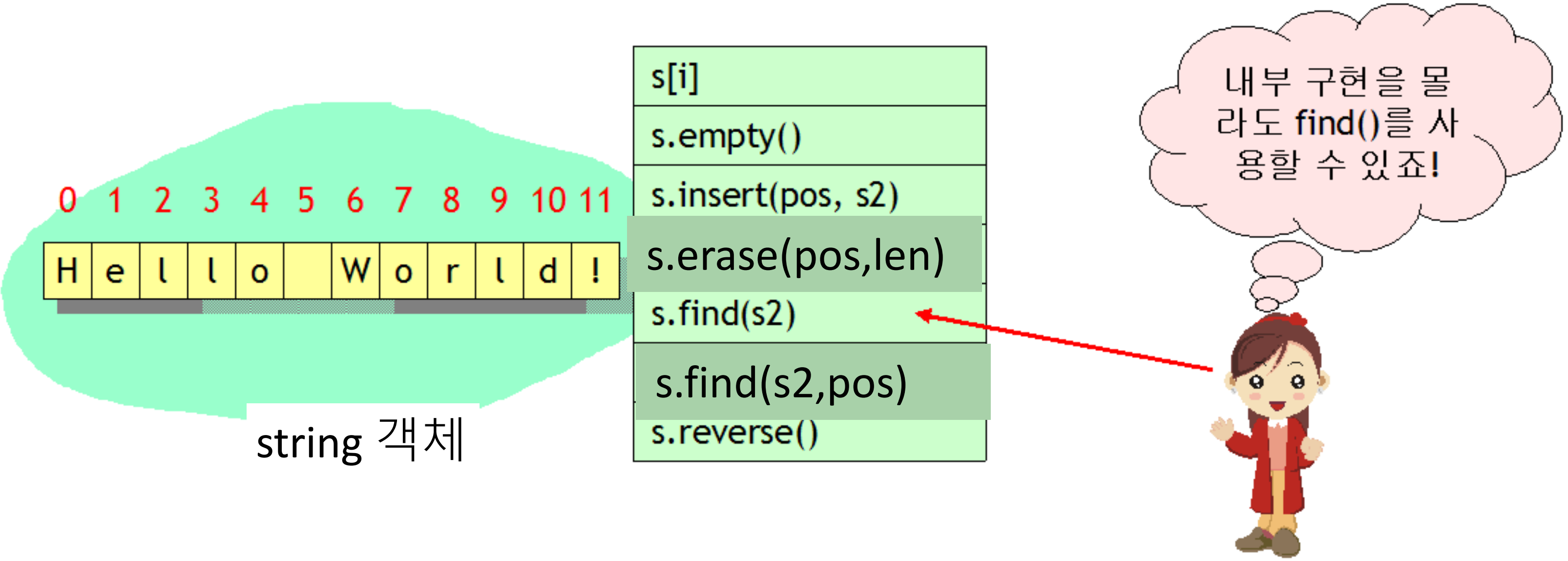


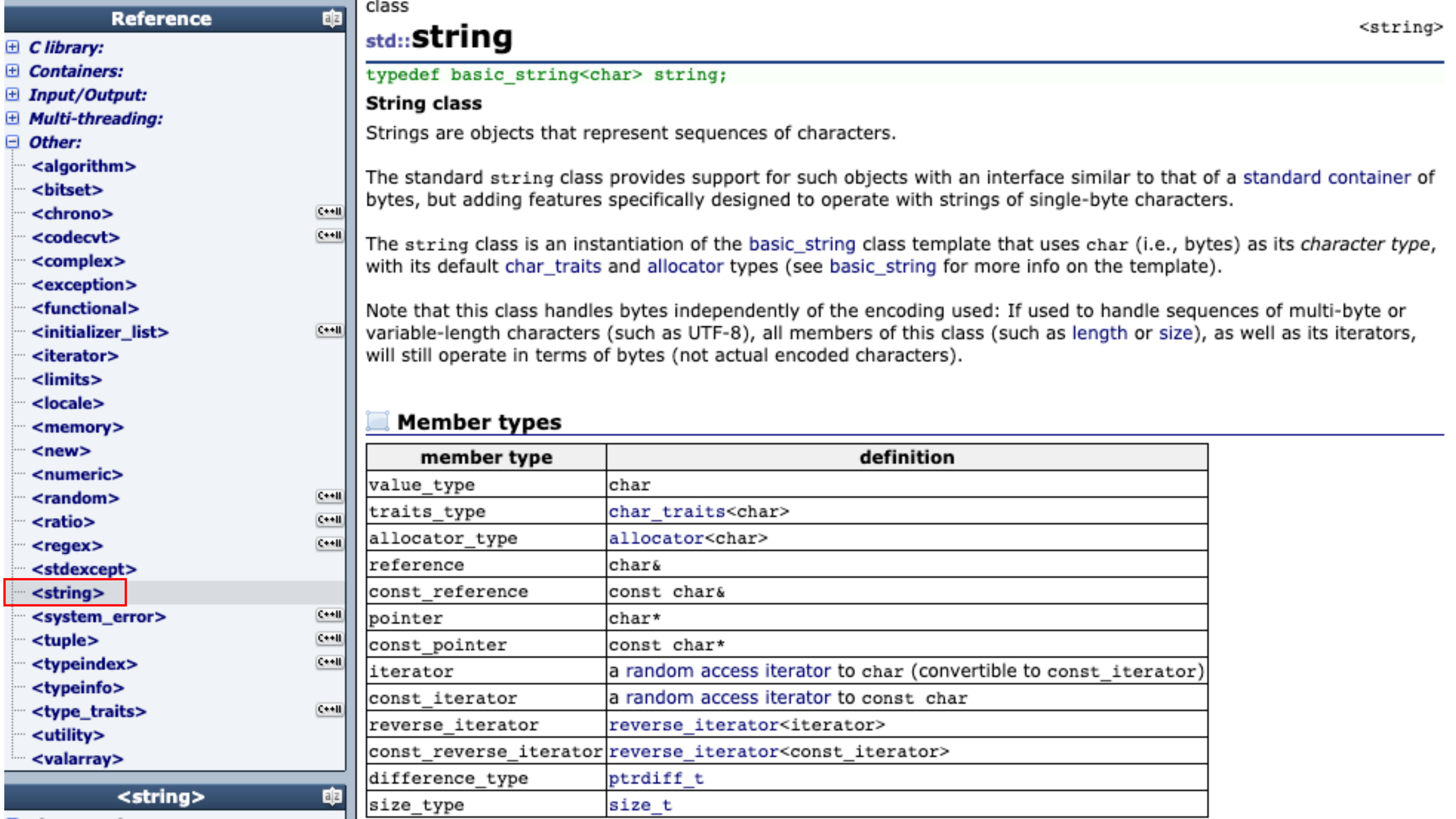
```
length of 12 is 2
length of 12ab is 4
```

클래스 사용의 예 : string 클래스

- C++에서는 문자열을 나타내는 클래스 string을 제공한다.

#include <string>





class

std::string

<string>

```
typedef basic_string<char> string;
```

String class

Strings are objects that represent sequences of characters.

The standard `string` class provides support for such objects with an interface similar to that of a standard container of bytes, but adding features specifically designed to operate with strings of single-byte characters.

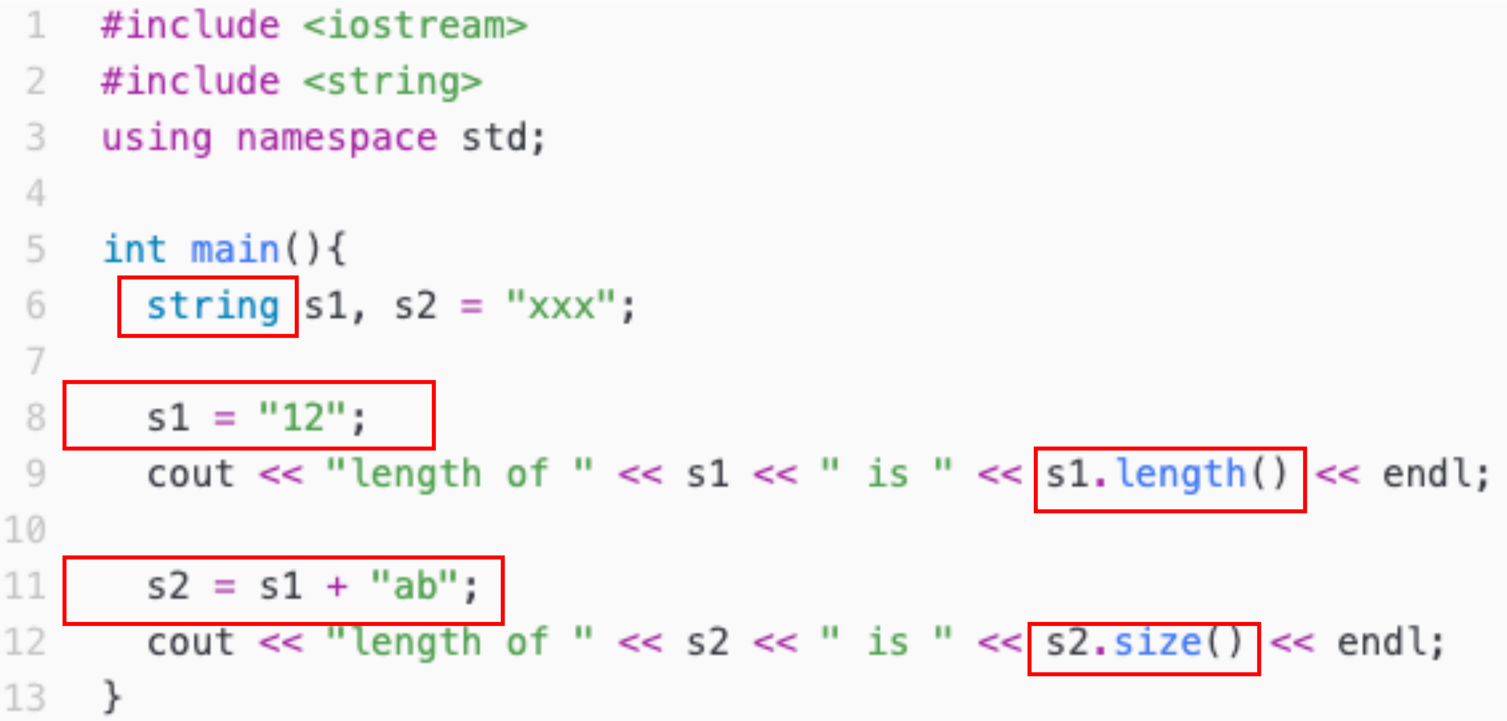
The `string` class is an instantiation of the basic_string class template that uses `char` (i.e., bytes) as its *character type*, with its default char_traits and allocator types (see basic_string for more info on the template).

Note that this class handles bytes independently of the encoding used: If used to handle sequences of multi-byte or variable-length characters (such as UTF-8), all members of this class (such as length or size), as well as its iterators, will still operate in terms of bytes (not actual encoded characters).

Member types

member type	definition
`value_type`	`char`
`traits_type`	`char_traits<char>`
`allocator_type`	`allocator<char>`
`reference`	`char&`
`const_reference`	`const char&`
`pointer`	`char*`
`const_pointer`	`const char*`
`iterator`	a random access iterator to `char` (convertible to `const_iterator`)
`const_iterator`	a random access iterator to `const char`
`reverse_iterator`	`reverse_iterator<iterator>`
`const_reverse_iterator`	`reverse_iterator<const_iterator>`
`difference_type`	`ptrdiff_t`
`size_type`	`size_t`

string class in <string>

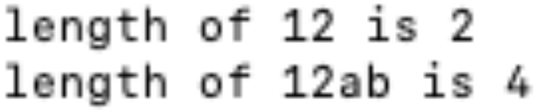


```cpp
#include <iostream>
#include <string>
using namespace std;

int main(){
  string s1, s2 = "xxx";

    s1 = "12";
    cout << "length of " << s1 << " is " << s1.length() << endl;

    s2 = s1 + "ab";
    cout << "length of " << s2 << " is " << s2.size() << endl;
}
```



```
length of 12 is 2
length of 12ab is 4
```