String

# C Strings

<cstring> (string.h)

'string.h' header file defines several functions to manipulate C strings and arrays.

# Macros

NULL Null pointer (macro )

# Types

size\_t Unsigned integral type (type )

# Functions

## Copying

|  |  |
| --- | --- |
| memcpy | Copy block of memory (function )  void \* memcpy  ( void \* destination, const void \* source, size\_t num ); |
| memmove | Move block of memory (function )  void \* memmove  ( void \* destination, const void \* source, size\_t num ); |
| strcpy | Copy string (function )  char \* strcpy  ( char \* destination, const char \* source ); |
| strncpy | Copy characters from string (function )  char \* strncpy  ( char \* destination, const char \* source, size\_t num ); |

## Concatenation

|  |  |
| --- | --- |
| strcat | Concatenate strings (function )  char \* strcat  ( char \* destination, const char \* source ); |
| strncat | Append characters from string (function )  char \* strncat  ( char \* destination, const char \* source, size\_t num ); |

## Comparison

|  |  |
| --- | --- |
| memcmp | Compare two blocks of memory (function )  int memcmp ( const void \* ptr1, const void \* ptr2, size\_t num ); |
| strcmp | Compare two strings (function )  int strcmp ( const char \* str1, const char \* str2 ); |
| strcoll | Compare two strings using locale (function )  int strcoll ( const char \* str1, const char \* str2 ); |
| strncmp | Compare characters of two strings (function )  int strncmp ( const char \* str1, const char \* str2, size\_t num ); |
| strxfrm | Transform string using locale (function )  size\_t strxfrm ( char \* destination, const char \* source, size\_t num ); |

## Searching

|  |  |
| --- | --- |
| memchr | Locate character in block of memory (function ) |
| const void \* memchr ( const void \* ptr, int value, size\_t num ); |
| void \* memchr ( void \* ptr, int value, size\_t num ); |
|  |
| void \* memchr ( const void \*, int, size\_t ); // C |
| strchr | Locate first occurrence of character in string (function ) |
| const char \* strchr ( const char \* str, int character ); |
| char \* strchr ( char \* str, int character ); |
|  |
| char \* strchr ( const char \*, int ); // C |
| strcspn | Get span until character in string (function ) |
| size\_t strcspn ( const char \* str1, const char \* str2 ); |
| strpbrk | Locate characters in string (function ) |
| const char \* strpbrk ( const char \* str1, const char \* str2 ); |
| char \* strpbrk ( char \* str1, const char \* str2 ); |
|  |
| char \* strpbrk ( const char \*, const char \* ); // C |
| strrchr | Locate last occurrence of character in string (function ) |
| const char \* strrchr ( const char \* str, int character ); |
| char \* strrchr ( char \* str, int character ); |
|  |
| char \* strrchr ( const char \*, int ); // C |
| strspn | Get span of character set in string (function ) |
| size\_t strspn ( const char \* str1, const char \* str2 ); |
| strstr | Locate substring (function ) |
| const char \* strstr ( const char \* str1, const char \* str2 ); |
| char \* strstr ( char \* str1, const char \* str2 ); |
|  |
| char \* strstr ( const char \*, const char \* ); // C |
| strtok | Split string into tokens (function ) |
| char \* strtok ( char \* str, const char \* delimiters ); |

## Other:

|  |  |
| --- | --- |
| memset | Fill block of memory (function ) |
|  | void \* memset ( void \* ptr, int value, size\_t num ); |
| strerror | Get pointer to error message string (function ) |
|  | char \* strerror ( int errnum ); |
| strlen | Get string length (function ) |
|  | size\_t strlen ( const char \* str ); |

# Macros

NULL Null pointer (macro )

This macro expands to a null pointer constant.

A null pointer constant can be converted to any pointer type (or pointer-to-member type), which acquires a null pointer value.

This is a special value that indicates that the pointer is not pointing to any object.

* In C, A null-pointer constant is an integral constant expression that evaluates to zero (like 0 or 0L), or the cast of such value to type void\* (like **(void\*)0**).
* In C++, A null-pointer constant is an integral constant expression that evaluates to zero (such as **0 or 0L**).
* In C++11, A null-pointer constant is either an integral constant expression that evaluates to zero (such as **0 or 0L**), or a value of type nullptr\_t (such as **nullptr**).

# Types

size\_t Unsigned integral type (type )

**Unsigned integral type**

Alias of one of the fundamental unsigned integer types

It is a type able to represent the size of any object in bytes:

size\_t is the type returned by the sizeof operator and is widely used in the standard library to represent sizes and counts.

In <cstring>, it is used as the type of the parameter num in the functions memchr, memcmp, memcpy, memmove, memset, strncat, strncmp, strncpy and strxfrm, which in all cases it is used to specify the maximum number of bytes or characters the function has to affect.

It is also used as the return type for strcspn, strlen, strspn and strxfrm to return sizes and lengths.

# Functions - Copying

## memcpy Copy block of memory (function )

**void \* memcpy ( void \* destination, const void \* source, size\_t num );**

Copy block of memory

Copies the values of num bytes from source directly to the memory block pointed to by destination.

The underlying type of the objects pointed to by both the source and destination pointers are irrelevant for this function;

The result is a binary copy of the data.

The function does not check for any terminating null character in source - it always copies exactly num bytes.

To avoid overflows, the size of the arrays pointed to by both the destination and source parameters, shall be at least num bytes, and should not overlap (for overlapping memory blocks, memmove is a safer approach).

**Return**:

destination is returned.

/\* memcpy example \*/

#include <stdio.h>

#include <string.h>

struct {

char name[40];

int age;

} person, person\_copy;

int main ()

{

char myname[] = "Pierre de Fermat";

/\* using memcpy to copy string: \*/

memcpy ( person.name, myname, strlen(myname)+1 );

person.age = 46;

/\* using memcpy to copy structure: \*/

memcpy ( &person\_copy, &person, sizeof(person) );

printf ("person\_copy: %s, %d \n", person\_copy.name, person\_copy.age );

return 0;

}

Output:

person\_copy: Pierre de Fermat, 46

## memmove Move block of memory (function )

**void \* memmove ( void \* destination, const void \* source, size\_t num );**

Move block of memory

Copies the values of num bytes from the location pointed by source to the memory block pointed by destination.

Copying takes place as if an intermediate buffer were used, allowing the destination and source to overlap.

The underlying type of the objects pointed by both the source and destination pointers are irrelevant for this function;

The result is a binary copy of the data.

The function does not check for any terminating null character in source - it always copies exactly num bytes.

To avoid overflows, the size of the arrays pointed by both the destination and source parameters, shall be at least num bytes.

**Return**:

destination is returned.

/\* memmove example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str[] = "memmove can be very useful......";

memmove (str+20,str+15,11);

puts (str);

return 0;

}

Output:

memmove can be very very useful.

## strcpy Copy string (function )

**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.

**Return**:

destination is returned.

/\* 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;

}

Output:

str1: Sample string

str2: Sample string

str3: copy successful

## strncpy Copy characters from string (function )

**char \* strncpy ( char \* destination, const char \* source, size\_t num );**

Copy characters from string

Copies the first num characters of source to destination.

If the end of the source C string (which is signaled by a null-character) is found before num characters have been copied, destination is padded with zeros until a total of num characters have been written to it.

No null-character is implicitly appended at the end of destination if source is longer than num.

Thus, in this case, destination shall not be considered a null terminated C string (reading it as such would overflow).

destination and source shall not overlap (see memmove for a safer alternative when overlapping).

**Return**:

destination is returned.

/\* strncpy example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str1[]= "To be or not to be";

char str2[40];

char str3[40];

/\* copy to sized buffer (overflow safe): \*/

strncpy ( str2, str1, sizeof(str2) );

/\* partial copy (only 5 chars): \*/

strncpy ( str3, str2, 5 );

str3[5] = '\0'; /\* null character manually added \*/

puts (str1);

puts (str2);

puts (str3);

return 0;

}

Output:

To be or not to be

To be or not to be

To be

# Functions - Concatenation

## strcat Concatenate strings (function )

**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.

**Return**:

destination is returned.

/\* 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;

}

Output:

these strings are concatenated.

## strncat Append characters from string (function )

**char \* strncat ( char \* destination, const char \* source, size\_t num );**

Append characters from string

Appends the first num characters of source to destination, plus a terminating null-character.

If the length of the C string in source is less than num, only the content up to the terminating null-character is copied.

**Return**:

destination is returned.

/\* strncat example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str1[20];

char str2[20];

strcpy (str1,"To be ");

strcpy (str2,"or not to be");

strncat (str1, str2, 6);

puts (str1);

return 0;

}

Output:

To be or not

# Functions - Comparison

## memcmp Compare two blocks of memory (function )

**int memcmp ( const void \* ptr1, const void \* ptr2, size\_t num );**

Compare two blocks of memory

Compares the first num bytes of the block of memory pointed by ptr1 to the first num bytes pointed by ptr2, returning zero if they all match or a value different from zero representing which is greater if they do not.

Notice that, **unlike strcmp, the function does not stop comparing after finding a null character**.

**Return**:

an integral value indicating the relationship between the content of the memory blocks:

|  |  |
| --- | --- |
| **Return value** | **Indicates** |
| <0 | the first byte that does not match in both memory blocks has a lower value in ptr1 than in ptr2 (if evaluated as unsigned char values) |
| 0 | the contents of both memory blocks are equal |
| >0 | the first byte that does not match in both memory blocks has a greater value in ptr1 than in ptr2 (if evaluated as unsigned char values) |

/\* memcmp example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char buffer1[] = "DWgaOtP12df0";

char buffer2[] = "DWGAOTP12DF0";

int n;

n=memcmp ( buffer1, buffer2, sizeof(buffer1) );

if (n>0) printf ("'%s' is greater than '%s'.\n",buffer1,buffer2);

else if (n<0) printf ("'%s' is less than '%s'.\n",buffer1,buffer2);

else printf ("'%s' is the same as '%s'.\n",buffer1,buffer2);

return 0;

}

Output:

'DWgaOtP12df0' is greater than 'DWGAOTP12DF0'.

## strcmp Compare two strings (function )

**int strcmp ( const char \* str1, const char \* str2 );**

Compare two strings

Compares the C string str1 to the C string str2.

This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating null-character is reached.

**This function performs a binary comparison of the characters**.

For a function that takes into account locale-specific rules, see strcoll.

**Return**:

an integral value indicating the relationship between the strings:

|  |  |
| --- | --- |
| **Return value** | **Indicates** |
| <0 | the first character that does not match has a lower value in ptr1 than in ptr2 |
| 0 | contents of both strings are equal |
| >0 | the first character that does not match has a greater value in ptr1 than in ptr2 |

#include <stdio.h>

#include <string.h>

int main ()

{

char key[] = "apple";

char buffer[80];

do {

printf ("Guess my favorite fruit? ");

fflush (stdout);

scanf ("%79s",buffer);

} while (strcmp (key,buffer) != 0);

puts ("Correct answer!");

return 0;

}

Output:

Guess my favourite fruit? orange

Guess my favourite fruit? apple

Correct answer!

## strcoll Compare two strings using locale (function )

**int strcoll ( const char \* str1, const char \* str2 );**

Compare two strings using locale

Compares the C string str1 to the C string str2, both interpreted appropriately according to the LC\_COLLATE category of the C locale currently selected.

This function starts comparing the first character of each string.

If they are equal to each other continues with the following pair until the characters differ or until a null-character signaling the end of a string is reached.

The behavior of this function depends on the LC\_COLLATE category of the selected C locale.

**Return**:

an integral value indicating the relationship between the strings:

|  |  |
| --- | --- |
| **Return value** | **Indicates** |
| <0 | the first character that does not match has a lower value in ptr1 than in ptr2 |
| 0 | the contents of both strings are equal |
| >0 | the first character that does not match has a greater value in ptr1 than in ptr2 |

## strncmp Compare characters of two strings (function )

**int strncmp ( const char \* str1, const char \* str2, size\_t num );**

Compare characters of two strings

Compares up to num characters of the C string str1 to those of the C string str2.

This function starts comparing the first character of each string.

If they are equal to each other, it continues with the following pairs until the characters differ, until a terminating null-character is reached, or until num characters match in both strings, whichever happens first.

**Return**:

an integral value indicating the relationship between the strings:

|  |  |
| --- | --- |
| **Return value** | **Indicates** |
| <0 | the first character that does not match has a lower value in ptr1 than in ptr2 |
| 0 | the contents of both strings are equal |
| >0 | the first character that does not match has a greater value in ptr1 than in ptr2 |

/\* strncmp example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str[][5] = { "R2D2" , "C3PO" , "R2A6" };

int n;

puts ("Looking for R2 astromech droids...");

for (n=0 ; n<3 ; n++)

if (strncmp (str[n],"R2xx",2) == 0)

{

printf ("found %s\n",str[n]);

}

return 0;

}

Output:

Looking for R2 astromech droids...

found R2D2

found R2A6

## strxfrm Transform string using locale (function )

**size\_t strxfrm ( char \* destination, const char \* source, size\_t num );**

Transform string using locale

Transforms the C string pointed by source according to the current locale and copies the first num characters of the transformed string to destination, returning its length.

Alternativelly, the function can be used to only retrieve the length, by specifying a null pointer for destination and zero for num.

destination and source shall not overlap.

The behavior of this function depends on the LC\_COLLATE category of the selected C locale.

size\_t is an unsigned integral type.

**Return**:

The length of the transformed string, not including the terminating null-character.

# Functions - Searching

## memchr Locate character in block of memory (function)

|  |
| --- |
| **const void \* memchr ( const void \* ptr, int value, size\_t num );** |
| **void \* memchr ( void \* ptr, int value, size\_t num );** |
|  |
| **void \* memchr ( const void \*, int, size\_t ); // C** |

Locate character in block of memory

Searches within the first num bytes of the block of memory pointed by ptr for the first occurrence of value (interpreted as an unsigned char), and returns a pointer to it.

Both value and each of the bytes checked on the the ptr array are interpreted as unsigned char for the comparison.

**Return**:

A pointer to the first occurrence of value in the block of memory pointed by ptr.

A null pointer, If the value is not found

/\* memchr example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char \* pch;

char str[] = "Example string";

pch = (char\*) memchr (str, 'p', strlen(str));

if (pch!=NULL)

printf ("'p' found at position %d.\n", pch-str+1);

else

printf ("'p' not found.\n");

return 0;

}

Output:

'p' found at position 5.

## strchr Locate first occurrence of character in string (function)

|  |
| --- |
| **const char \* strchr ( const char \* str, int character );** |
| **char \* strchr ( char \* str, int character );** |
|  |
| **char \* strchr ( const char \*, int ); // C** |

Locate first occurrence of character in string

Returns a pointer to the first occurrence of character in the C string str.

The terminating null-character is considered part of the C string.

Therefore, it can also be located in order to retrieve a pointer to the end of a string.

**Return**:

A pointer to the first occurrence of character in str.

A null pointer, If the character is not found

/\* strchr example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str[] = "This is a sample string";

char \* pch;

printf ("Looking for the 's' character in \"%s\"...\n",str);

pch=strchr(str,'s');

while (pch!=NULL)

{

printf ("found at %d\n",pch-str+1);

pch=strchr(pch+1,'s');

}

return 0;

}

Output:

Looking for the 's' character in "This is a sample string"...

found at 4

found at 7

found at 11

found at 18

## strcspn Get span until character in string (function)

|  |
| --- |
| **size\_t strcspn ( const char \* str1, const char \* str2 );** |

Get span until character in string

Scans str1 for the first occurrence of any of the characters that are part of str2, returning the number of characters of str1 read before this first occurrence.

The search includes the terminating null-characters.

Therefore, the function will return the length of str1 if none of the characters of str2 are found in str1.

size\_t is an unsigned integral type.

**Return**:

The length of the initial part of str1 not containing any of the characters that are part of str2.

This is the length of str1 if none of the characters in str2 are found in str1.

/\* strcspn example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str[] = "fcba73";

char keys[] = "1234567890";

int i;

i = strcspn (str,keys);

printf ("The first number in str is at position %d.\n",i+1);

return 0;

}

Output:

The first number in str is at position 5

## strpbrk Locate characters in string (function)

|  |
| --- |
| **const char \* strpbrk ( const char \* str1, const char \* str2 );** |
| **char \* strpbrk ( char \* str1, const char \* str2 );** |
|  |
| **char \* strpbrk ( const char \*, const char \* ); // C** |

Locate characters in string

Returns a pointer to the first occurrence in str1 of any of the characters that are part of str2, or a null pointer if there are no matches.

The search does not include the terminating null-characters of either strings, but ends there.

**Return**:

A pointer to the first occurrence in str1 of any of the characters that are part of str2, or

a null pointer if none of the characters of str2 is found in str1 before the terminating null-character.

If none of the characters of str2 is present in str1, a null pointer is returned.

/\* strpbrk example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str[] = "This is a sample string";

char key[] = "aeiou";

char \* pch;

printf ("Vowels in '%s': ",str);

pch = strpbrk (str, key);

while (pch != NULL)

{

printf ("%c " , \*pch);

pch = strpbrk (pch+1,key);

}

printf ("\n");

return 0;

}

Output:

Vowels in 'This is a sample string': i i a a e i

## strrchr Locate last occurrence of character in string (function)

|  |
| --- |
| **const char \* strrchr ( const char \* str, int character );** |
| **char \* strrchr ( char \* str, int character );** |
|  |
| **char \* strrchr ( const char \*, int ); // C** |

Locate last occurrence of character in string

Returns a pointer to the last occurrence of character in the C string str.

The terminating null-character is considered part of the C string.

Therefore, it can also be located to retrieve a pointer to the end of a string.

**Return**:

A pointer to the last occurrence of character in str.

a null pointer, If the character is not found

/\* strrchr example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str[] = "This is a sample string";

char \* pch;

pch=strrchr(str,'s');

printf ("Last occurence of 's' found at %ld \n",(pch-str)+1);

return 0;

}

Output:

Last occurrence of 's' found at 18

## strspn Get span of character set in string (function)

|  |
| --- |
| **size\_t strspn ( const char \* str1, const char \* str2 );** |

Get span of character set in string

Returns the length of the initial portion of str1 which consists only of characters that are part of str2.

The search does not include the terminating null-characters of either strings, but ends there.

size\_t is an unsigned integral type.

**Return**:

The length of the initial portion of str1 containing only characters that appear in str2.

Therefore, if all of the characters in str1 are in str2, the function returns the **length of the entire str1 string**,

and if the first character in str1 is not in str2, the function returns **zero**.

/\* strspn example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

int i;

char strtext[] = "129th";

char cset[] = "1234567890";

i = strspn (strtext,cset);

printf ("The initial number has %d digits.\n",i);

return 0;

}

Output:

The initial number has 3 digits.

## strstr Locate substring (function)

|  |
| --- |
| **const char \* strstr ( const char \* str1, const char \* str2 );** |
| **char \* strstr ( char \* str1, const char \* str2 );** |
|  |
| **char \* strstr ( const char \*, const char \* ); // C** |

Locate substring

Returns a pointer to the first occurrence of str2 in str1, or a null pointer if str2 is not part of str1.

The matching process does not include the terminating null-characters, but it stops there.

**Return**:

A pointer to the first occurrence in str1 of the entire sequence of characters specified in str2, or

a null pointer if the sequence is not present in str1.

/\* strstr example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str[] ="This is a simple string";

char \* pch;

pch = strstr (str,"simple");

strncpy (pch,"sample",6);

puts (str);

return 0;

}

Output:

This is a sample string

## strtok Split string into tokens (function)

|  |
| --- |
| **char \* strtok ( char \* str, const char \* delimiters );** |

Split string into tokens

A sequence of calls to this function split str into tokens,

which are sequences of contiguous characters separated by any of the characters that are part of delimiters.

On a first call, the function expects a C string as argument for str,

whose first character is used as the starting location to scan for tokens.

In subsequent calls, the function expects a null pointer and uses

the position right after the end of the last token as the new starting location for scanning.

To determine the beginning and the end of a token,

the function first scans from the starting location for

the first character not contained in delimiters (which becomes the beginning of the token).

And then scans starting from this beginning of the token for the first character contained in delimiters,

which becomes the end of the token. The scan also stops if the terminating null character is found.

This end of the token is automatically replaced by a null-character,

and the beginning of the token is returned by the function.

Once the terminating null character of str is found in a call to strtok,

all subsequent calls to this function (with a null pointer as the first argument) return a null pointer.

The point where the last token was found is kept internally by the function to be used on the next call

(particular library implementations are not required to avoid data races).

**Return**:

If a token is found, a pointer to the beginning of the token.

Otherwise, a null pointer.

A null pointer is always returned when the end of the string (i.e., a null character) is reached in the string being scanned.

/\* strtok example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str[] ="- This, a sample string.";

char \* pch;

printf ("Splitting string \"%s\" into tokens:\n",str);

pch = strtok (str," ,.-");

while (pch != NULL)

{

printf ("%s\n",pch);

pch = strtok (NULL, " ,.-");

}

return 0;

}

Output:

Splitting string "- This, a sample string." into tokens:

This

a

sample

string

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Functions - Other:

## memset Fill block of memory (function)

**void \* memset ( void \* ptr, int value, size\_t num );**

Fill block of memory

Sets the first num bytes of the block of memory pointed by ptr to the specified value (interpreted as an unsigned char).

**Return**:

ptr is returned.

/\* memset example \*/

#include <stdio.h>

#include <string.h>

int main ()

{

char str[] = "almost every programmer should know memset!";

memset (str,'-',6);

puts (str);

return 0;

}

Output:

------ every programmer should know memset!

## strerror Get pointer to error message string (function)

**char \* strerror ( int errnum );**

Get pointer to error message string

Interprets the value of errnum, generating a string with a message that describes the error condition as if set to errno by a function of the library.

The returned pointer points to a statically allocated string, which shall not be modified by the program.

Further calls to this function may overwrite its content (particular library implementations are not required to avoid data races).

The error strings produced by strerror may be specific to each system and library implementation.

**Return**:

A pointer to the error string describing error errnum.

/\* strerror example : error list \*/

#include <stdio.h>

#include <string.h>

#include <errno.h>

int main ()

{

FILE \* pFile;

pFile = fopen ("unexist.ent","r");

if (pFile == NULL)

printf ("Error opening file unexist.ent: %s\n",strerror(errno));

return 0;

}

Output:

Error opening file unexist.ent: No such file or directory

## strlen Get string length (function)

**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:

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.

**Return**:

The length of string.

/\* 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;

}

Output:

Enter sentence: just testing

The sentence entered is 12 characters long.