Fondamenti di Informatica II e Lab.

La libreria <string.h>

Pagine estratte dal sito www.cplusplus.com

Ultimo aggiornamento 18/12/2014

function

strlen

```
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.

Parameters

str
C string.

Return Value

The length of string.

function

memcpy

```
void * memcpy ( void * destination, const void * source, size_t num );
```

Copy block of memory

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

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, and should not overlap (for overlapping memory blocks, memmove is a safer approach).

Parameters

destination

Pointer to the destination array where the content is to be copied, type-casted to a pointer of type void*.

source

Pointer to the source of data to be copied, type-casted to a pointer of type const void*.

num

Number of bytes to copy.
size_t is an unsigned integral type.

Return Value

destination is returned.

function

memmove

```
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.

Parameters

destination

Pointer to the destination array where the content is to be copied, type-casted to a pointer of type void*.

source

Pointer to the source of data to be copied, type-casted to a pointer of type const void*.

num

Number of bytes to copy.
size_t is an unsigned integral type.

Return Value

destination is returned.

function

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.

function

strncpy

```
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).

Parameters

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

source
: C string to be copied.

num
: Maximum number of characters to be copied from *source*.
size_t is an unsigned integral type.

Return Value

destination is returned.

function

strcat

```
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.

function

strncat

```
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.

Parameters

destination
: Pointer to the destination array, which should contain a C string, and be large enough to contain the concatenated resulting string, including the additional null-character.

source
: C string to be appended.

num
: Maximum number of characters to be appended.
size_t is an unsigned integral type.

Return Value

destination is returned.

function

memcmp

```
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.

Parameters

ptr1

Pointer to block of memory.

ptr2

Pointer to block of memory.

num

Number of bytes to compare.

Return Value

Returns 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)

function

strcmp

```
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.

Parameters

str1

C string to be compared.

str2

C string to be compared.

Return Value

Returns 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*

function

strncmp

```
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.

Parameters

str1

C string to be compared.

str2

C string to be compared.

num

Maximum number of characters to compare.
size_t is an unsigned integral type.

Return Value

Returns 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*

function

memchr

```
void * memchr ( const void * ptr, int value, size_t num );
```

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.

Parameters

ptr

Pointer to the block of memory where the search is performed.

value

Value to be located. The value is passed as an int, but the function performs a byte per byte search using the*unsigned char* conversion of this value.

num

Number of bytes to be analyzed.
size_t is an unsigned integral type.

Return Value

A pointer to the first occurrence of *value* in the block of memory pointed by *ptr*.
If the *value* is not found, the function returns a null pointer.

function

strchr

```
char * strchr ( const char * str, int character );
```

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.

Parameters

str

C string.

character

Character to be located. It is passed as its int promotion, but it is internally converted back to *char* for the comparison.

Return Value

A pointer to the first occurrence of *character* in *str*.
If the *character* is not found, the function returns a null pointer.

function

strpbrk

```
char * strpbrk ( const char * str1, const char * str2 );
```

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.

Parameters

str1

C string to be scanned.

str2

C string containing the characters to match.

Return Value

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.

function

strrchr

```
char * strrchr ( const char * str, int character );
```

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.

Parameters

str

C string.

character

Character to be located. It is passed as its int promotion, but it is internally converted back to *char*.

Return Value

A pointer to the last occurrence of *character* in *str*.
If the *character* is not found, the function returns a null pointer.

function

strstr

```
char * strstr ( const char * str1, const char * str2 );
```

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.

Parameters

str1

C string to be scanned.

str2

C string containing the sequence of characters to match.

Return Value

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*.

function

strtok

```
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).

Parameters

str

C string to truncate. Notice that this string is modified by being broken into smaller strings (tokens). Alternatively, a null pointer may be specified, in which case the function continues scanning where a previous successful call to the function ended.

delimiters

C string containing the delimiter characters.
These can be different from one call to another.

Return Value

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.