

libft

Generated by Doxygen 1.8.16

<b>1 libft</b>	<b>1</b>
<b>1 libft</b>	<b>1</b>
1.0.1 Overview	1
1.0.2 Acknowledgements	1
<b>2 Data Structure Index</b>	<b>2</b>
2.1 Data Structures	2
<b>3 File Index</b>	<b>2</b>
3.1 File List	2
<b>4 Data Structure Documentation</b>	<b>4</b>
4.1 s_list Struct Reference	4
4.1.1 Detailed Description	5
4.1.2 Field Documentation	5
<b>5 File Documentation</b>	<b>6</b>
5.1 libft.h File Reference	6
5.1.1 Function Documentation	8
<b>Index</b>	<b>49</b>

# 1 libft

*This project is part of the official curriculum at [School 42](#).*

## 1.0.1 Overview

- [Official instructions](#)
- The task is to recreate various standard C library functions, as well as additional useful functions.
- Documentation ( [html](#), [pdf](#)) generated with [Doxygen](#).
- The project is consistent with the [Norme](#), the code standard accepted at *School 42*.
- Use `make` to compile the library and include in projects with `#include "libft.h"`
- This project has been tested with [Moullitest](#).

## 1.0.2 Acknowledgements

My thanks go to [yyang42](#) for making Moullitest freely available online, to the entire team behind School 42 and its [Moscow branch](#), to my fellow students for fruitful discussions, as well as to creators and maintainers of Doxygen.

## 2 Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">s_list</a>	<a href="#">4</a>
------------------------	-------------------

## 3 File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">libft.h</a>	<a href="#">6</a>
<a href="#">src/ft_abs.c</a>	<a href="#">??</a>
<a href="#">src/ft_atoi.c</a>	<a href="#">??</a>
<a href="#">src/ft_bzero.c</a>	<a href="#">??</a>
<a href="#">src/ft_isalnum.c</a>	<a href="#">??</a>
<a href="#">src/ft_isalpha.c</a>	<a href="#">??</a>
<a href="#">src/ft_isascii.c</a>	<a href="#">??</a>
<a href="#">src/ft_isdigit.c</a>	<a href="#">??</a>
<a href="#">src/ft_isprint.c</a>	<a href="#">??</a>
<a href="#">src/ft_isspace.c</a>	<a href="#">??</a>
<a href="#">src/ft_itoa.c</a>	<a href="#">??</a>
<a href="#">src/ft_lstadd.c</a>	<a href="#">??</a>
<a href="#">src/ft_lstappend.c</a>	<a href="#">??</a>
<a href="#">src/ft_lstdel.c</a>	<a href="#">??</a>
<a href="#">src/ft_lstdelone.c</a>	<a href="#">??</a>
<a href="#">src/ft_lstiter.c</a>	<a href="#">??</a>
<a href="#">src/ft_lstlast.c</a>	<a href="#">??</a>
<a href="#">src/ft_lstmap.c</a>	<a href="#">??</a>
<a href="#">src/ft_lstnew.c</a>	<a href="#">??</a>
<a href="#">src/ft_max.c</a>	<a href="#">??</a>
<a href="#">src/ft_memalloc.c</a>	<a href="#">??</a>
<a href="#">src/ft_memccpy.c</a>	<a href="#">??</a>

src/ft_memchr.c	??
src/ft_memcmp.c	??
src/ft_memcpy.c	??
src/ft_memdel.c	??
src/ft_memmove.c	??
src/ft_memset.c	??
src/ft_min.c	??
src/ft_power.c	??
src/ft_putchar.c	??
src/ft_putchar_fd.c	??
src/ft_putendl.c	??
src/ft_putendl_fd.c	??
src/ft_putnbr.c	??
src/ft_putnbr_fd.c	??
src/ft_putstr.c	??
src/ft_putstr_fd.c	??
src/ft_sqrt.c	??
src/ft_strcapitalize.c	??
src/ft_strcat.c	??
src/ft_strchr.c	??
src/ft_strchr.c	??
src/ft_strclr.c	??
src/ft_strcmp.c	??
src/ft_strcpy.c	??
src/ft_strdel.c	??
src/ft_strdup.c	??
src/ft_strequ.c	??
src/ft_strintab.c	??
src/ft_striter.c	??

<code>src/ft_striteri.c</code>	??
<code>src/ft_strjoin.c</code>	??
<code>src/ft_strlast.c</code>	??
<code>src/ft_strlcat.c</code>	??
<code>src/ft_strlcpy.c</code>	??
<code>src/ft_strlen.c</code>	??
<code>src/ft_strmap.c</code>	??
<code>src/ft_strmapi.c</code>	??
<code>src/ft_strncat.c</code>	??
<code>src/ft_strncmp.c</code>	??
<code>src/ft_strncpy.c</code>	??
<code>src/ft_strndup.c</code>	??
<code>src/ft_strnequ.c</code>	??
<code>src/ft_strnew.c</code>	??
<code>src/ft_strnstr.c</code>	??
<code>src/ft_strrchr.c</code>	??
<code>src/ft_strrev.c</code>	??
<code>src/ft_strsplit.c</code>	??
<code>src/ft_strstr.c</code>	??
<code>src/ft_strsub.c</code>	??
<code>src/ft_strtrim.c</code>	??
<code>src/ft_tolower.c</code>	??
<code>src/ft_toupper.c</code>	??

## 4 Data Structure Documentation

### 4.1 s\_list Struct Reference

```
#include <libft.h>
```

#### Data Fields

- void \* [content](#)
- size\_t [content\\_size](#)
- struct [s\\_list](#) \* [next](#)

### 4.1.1 Detailed Description

Represent links of a list.

Definition at line 33 of file libft.h.

### 4.1.2 Field Documentation

#### 4.1.2.1 content `s_list::content`

The data contained in the link. The `void *` allows to store any kind of data.

Definition at line 35 of file libft.h.

Referenced by `ft_lstdel()`, `ft_lstdelone()`, and `ft_lstnew()`.

#### 4.1.2.2 content\_size `s_list::content_size`

The size of the data stored. The `void *` type doesn't allow you to know the size of the pointed data, as a consequence, it is necessary to save its size. For instance, the size of the string "42" is 3 bytes and the 32 bits integer 42 has a size of 4 bytes.

Definition at line 36 of file libft.h.

Referenced by `ft_lstdel()`, `ft_lstdelone()`, and `ft_lstnew()`.

#### 4.1.2.3 next `s_list::next`

The next link's address or `NULL` if it's the last link.

Definition at line 37 of file libft.h.

Referenced by `ft_lstappend()`, `ft_lstdel()`, `ft_lstdelone()`, `ft_lstiter()`, `ft_lstlast()`, `ft_lstmap()`, and `ft_lstnew()`.

The documentation for this struct was generated from the following file:

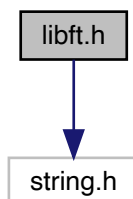
- [libft.h](#)

## 5 File Documentation

### 5.1 libft.h File Reference

```
#include <string.h>
```

Include dependency graph for libft.h:



#### Data Structures

- struct [s\\_list](#)

#### Typedefs

- typedef struct [s\\_list](#) [t\\_list](#)

#### Functions

- int [ft\\_abs](#) (int a)
- int [ft\\_atoi](#) (const char \*str)
- void [ft\\_bzero](#) (void \*s, size\_t n)
- int [ft\\_isalnum](#) (int c)
- int [ft\\_isalpha](#) (int c)
- int [ft\\_isascii](#) (int c)
- int [ft\\_isdigit](#) (int c)
- int [ft\\_isprint](#) (int c)
- int [ft\\_isspace](#) (int c)
- char \* [ft\\_itoa](#) (int n)
- void [ft\\_lstadd](#) ([t\\_list](#) \*\*alst, [t\\_list](#) \*new)
- void [ft\\_lstappend](#) ([t\\_list](#) \*\*alst, [t\\_list](#) \*new)
- void [ft\\_lstdel](#) ([t\\_list](#) \*\*alst, void(\*del)(void \*, size\_t))
- void [ft\\_lstdelone](#) ([t\\_list](#) \*\*alst, void(\*del)(void \*, size\_t))
- void [ft\\_lstiter](#) ([t\\_list](#) \*lst, void(\*f)([t\\_list](#) \*elem))
- [t\\_list](#) \* [ft\\_lstlast](#) ([t\\_list](#) \*head)
- [t\\_list](#) \* [ft\\_lstmap](#) ([t\\_list](#) \*lst, [t\\_list](#) \*(\*f)([t\\_list](#) \*elem))
- [t\\_list](#) \* [ft\\_lstnew](#) (void const \*content, size\_t content\_size)
- int [ft\\_max](#) (int a, int b)
- void \* [ft\\_memalloc](#) (size\_t size)

- void \* [ft\\_memccpy](#) (void \*dst, const void \*src, int c, size\_t n)
- void \* [ft\\_memchr](#) (const void \*s, int c, size\_t n)
- int [ft\\_memcmp](#) (const void \*s1, const void \*s2, size\_t n)
- void \* [ft\\_memcpy](#) (void \*dst, const void \*src, size\_t n)
- void [ft\\_memdel](#) (void \*\*ap)
- void \* [ft\\_memmove](#) (void \*dst, const void \*src, size\_t n)
- void \* [ft\\_memset](#) (void \*b, int c, size\_t len)
- int [ft\\_min](#) (int a, int b)
- int [ft\\_power](#) (int num, unsigned int exponent)
- void [ft\\_putchar](#) (char c)
- void [ft\\_putchar\\_fd](#) (char c, int fd)
- void [ft\\_putendl](#) (char const \*s)
- void [ft\\_putendl\\_fd](#) (char const \*s, int fd)
- void [ft\\_putnbr](#) (int n)
- void [ft\\_putnbr\\_fd](#) (int n, int fd)
- void [ft\\_putstr](#) (char \*s, size\_t n)
- void [ft\\_putstr\\_fd](#) (char \*s, size\_t n, int fd)
- void [ft\\_puts](#) (char const \*s)
- void [ft\\_putstr](#) (char const \*s)
- void [ft\\_putstr\\_fd](#) (char const \*s, int fd)
- int [ft\\_sqrt](#) (int num)
- char \* [ft\\_strcapitalize](#) (const char \*str)
- char \* [ft\\_strcat](#) (char \*s1, const char \*s2)
- int [ft\\_strchr](#) (char const \*str, char c)
- char \* [ft\\_strchr](#) (const char \*s, int c)
- void [ft\\_strclr](#) (char \*s)
- int [ft\\_strcmp](#) (const char \*s1, const char \*s2)
- char \* [ft\\_strcpy](#) (char \*dst, const char \*src)
- void [ft\\_strdel](#) (char \*\*as)
- char \* [ft\\_strdup](#) (const char \*s1)
- int [ft\\_strequ](#) (const char \*s1, const char \*s2)
- int [ft\\_strintab](#) (const char \*str, char \*const tab[])
- void [ft\\_striter](#) (char \*s, void(\*f)(char \*))
- void [ft\\_striteri](#) (char \*s, void(\*f)(unsigned int, char \*))
- char \* [ft\\_strjoin](#) (const char \*s1, const char \*s2)
- char \* [ft\\_strlast](#) (char const \*str)
- size\_t [ft\\_strlen](#) (const char \*s)
- size\_t [ft\\_strlen](#) (const char \*s)
- char \* [ft\\_strmap](#) (char const \*s, char(\*f)(char))
- char \* [ft\\_strmap](#) (char const \*s, char(\*f)(unsigned int, char))
- char \* [ft\\_strncat](#) (char \*s1, const char \*s2, size\_t n)
- int [ft\\_strncmp](#) (const char \*s1, const char \*s2, size\_t n)
- char \* [ft\\_strncpy](#) (char \*dst, const char \*src, size\_t len)
- char \* [ft\\_strndup](#) (const char \*s1, size\_t len)
- int [ft\\_strnequ](#) (char const \*s1, char const \*s2, size\_t n)
- char \* [ft\\_strnew](#) (size\_t size)
- char \* [ft\\_strnstr](#) (const char \*haystack, const char \*needle, size\_t len)
- char \* [ft\\_strrchr](#) (const char \*s, int c)
- char \* [ft\\_strrev](#) (const char \*s)
- char \*\* [ft\\_strsplit](#) (char const \*str, char delim)
- char \* [ft\\_strstr](#) (const char \*haystack, const char \*needle)
- char \* [ft\\_strsub](#) (char const \*s, unsigned int start, size\_t len)
- char \* [ft\\_strtrim](#) (char const \*s)
- int [ft\\_tolower](#) (int c)
- int [ft\\_toupper](#) (int c)



### 5.1.1 Function Documentation

**5.1.1.1 `ft_abs()`** `int ft_abs (`  
`int a )`

Returns the absolute value of the argument.

#### Note

This function is only needed, since "The Norme" (the code standard at School 42) forbids the use of parametrized macros.

#### Parameters

in	a	The integer to take an absolute value of.
----	---	---

Definition at line 20 of file `ft_abs.c`.

```
21 {  
22     return (a < 0 ? -a : a);  
23 }
```

**5.1.1.2 `ft_atoi()`** `int ft_atoi (`  
`const char * str )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 20 of file `ft_atoi.c`.

```
21 {  
22     int i;  
23     int num;  
24     int is_neg;  
25  
26     i = 0;  
27     is_neg = 0;  
28     num = 0;  
29     while (str[i] == ' ' || str[i] == '\t' || str[i] == '\n' || str[i] == '\v'  
30 || str[i] == '\f' || str[i] == '\r')  
31         i++;  
32     if (str[i] == '-')  
33         is_neg = 1;  
34     if (str[i] == '-' || str[i] == '+')  
35         i++;  
36     while (str[i] >= '0' && str[i] <= '9')  
37     {  
38         num *= 10;  
39         num += (str[i] - '0');  
40         i++;  
41     }  
42     return (is_neg ? -num : num);  
43 }
```

**5.1.1.3 ft\_bzero()** void ft\_bzero (  
void \* s,  
size\_t n )

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_bzero.c.

```
20 {  
21     size_t      i;  
22     unsigned char *s1;  
23  
24     i = 0;  
25     s1 = (unsigned char *)s;  
26     while (i < n)  
27         s1[i++] = 0;  
28 }
```

**5.1.1.4 ft\_isalnum()** int ft\_isalnum (  
int c )

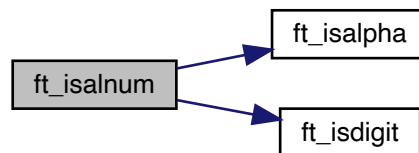
Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_isalnum.c.

```
20 {  
21     return (ft_isalpha(c) || ft_isdigit(c));  
22 }
```

References ft\_isalpha(), and ft\_isdigit().

Here is the call graph for this function:



**5.1.1.5 ft\_isalpha()** int ft\_isalpha (  
int c )

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_isalpha.c.

```
20 {  
21     if (c >= 'a' && c <= 'z')  
22         return (1);  
23     else if (c >= 'A' && c <= 'Z')  
24         return (1);  
25     else  
26         return (0);  
27 }
```

Referenced by ft\_isalnum().

**5.1.1.6 ft\_isascii()** `int ft_isascii (`  
`int c )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_isascii.c`.

```
20 {  
21     return (c >= 0 && c <= 127);  
22 }
```

**5.1.1.7 ft\_isdigit()** `int ft_isdigit (`  
`int c )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_isdigit.c`.

```
20 {  
21     if (c >= '0' && c <= '9')  
22         return (1);  
23     else  
24         return (0);  
25 }
```

Referenced by `ft_isalnum()`.

**5.1.1.8 ft\_isprint()** `int ft_isprint (`  
`int c )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_isprint.c`.

```
20 {  
21     return (c >= 040 && c <= 0176);  
22 }
```

**5.1.1.9 ft\_isspace()** `int ft_isspace (`  
`int c )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_isspace.c`.

```
20 {  
21     return ((c >= 9 && c <= 13) || c == ' ');  
22 }
```

**5.1.1.10 ft\_itoa()** `char* ft_itoa (`  
`int n )`

Allocate (with `malloc`) and returns a “fresh” string ending with `'\0'` representing the integer `n` given as argument. Negative numbers must be supported. If the allocation fails, the function returns `NULL`.

## Parameters

<i>n</i>	The integer to be transformed into a string.
----------	--

## Returns

The string representing the integer passed as argument.

Definition at line 36 of file ft\_itoa.c.

```

37 {
38     char    *str;
39     size_t   len;
40     int      tmp;
41
42     len = ft_int_len(n);
43     tmp = n < 0 ? n : -n;
44     if (n < 0)
45         len++;
46     if (!(str = ft_strnew(len)))
47         return (NULL);
48     while (tmp)
49     {
50         str[--len] = -(tmp % 10) + '0';
51         tmp /= 10;
52     }
53     if (n < 0)
54         str[--len] = '-';
55     return (str);
56 }
```

References ft\_strnew().

Here is the call graph for this function:



**5.1.1.11 ft\_lstadd()** void ft\_lstadd (  
     t\_list \*\* alst,  
     t\_list \* new )

Adds the element new at the beginning of the list.

## Parameters

<i>alst</i>	The address of a pointer to the first link of a list.
<i>new</i>	The link to add at the beginning of the list.

Definition at line 21 of file ft\_lstadd.c.

```

22 {
23     if (!alst || !new)
24         return ;
```

```
25     new->next = *alst;
26     *alst = new;
27 }
```

**5.1.1.12 ft\_lstappend()** void ft\_lstappend (

```
    t_list ** alst,
    t_list * new )
```

Adds the element new at the end of the list.

#### Parameters

<i>alst</i>	The address of a pointer to the first link of a list.
<i>new</i>	The link to add at the beginning of the list.

Definition at line 21 of file ft\_lstappend.c.

```
22 {
23     t_list *last;
24
25     if (!alst || !new)
26         return ;
27     if (!*alst)
28     {
29         *alst = new;
30         return ;
31     }
32     last = ft_lstlast(*alst);
33     last->next = new;
34 }
```

References ft\_lstlast(), and s\_list::next.

Here is the call graph for this function:



**5.1.1.13 ft\_lstdel()** void ft\_lstdel (

```
    t_list ** alst,
    void(*) (void *, size_t) del )
```

Takes as a parameter the address of a pointer to a link and frees the memory of this link and every successors of that link using the functions `del` and `free`. Finally the pointer to the link that was just freed must be set to `NULL` (quite similar to the function `memdel`).

## Parameters

<i>alst</i>	The address of a pointer to the first link of a list that needs to be freed.
<i>del</i>	The address of a function to apply to each link of a list.

Definition at line 27 of file ft\_lstdel.c.

```

28 {
29     t_list *head;
30     t_list *new_head;
31
32     if (!alst || !(*alst) || !del)
33         return ;
34     head = *alst;
35     while (head)
36     {
37         new_head = head->next;
38         del(head->content, head->content_size);
39         free(head);
40         head = new_head;
41     }
42     *alst = NULL;
43 }
```

References `s_list::content`, `s_list::content_size`, and `s_list::next`.

**5.1.1.14 ft\_lstdelone()** void ft\_lstdelone (   
     t\_list \*\* alst,   
     void(\*) (void \*, size\_t) del )

Takes as a parameter a link's pointer address and frees the memory of the link's content using the function `del` given as a parameter, then frees the link's memory using `free`. The memory of next must not be freed under any circumstance. Finally, the pointer to the link that was just freed must be set to `NULL` (quite similar to the function `memdel`).

## Parameters

<i>alst</i>	The address of a pointer to a link that needs to be freed.
<i>del</i>	The address of a function to apply to each link of a list.

Definition at line 27 of file ft\_lstdelone.c.

```

28 {
29     t_list *link;
30
31     if (!alst || !(*alst) || !del || !(*del))
32         return ;
33     link = *alst;
34     link->next = NULL;
35     del(link->content, link->content_size);
36     free(link);
37     *alst = NULL;
38 }
```

References `s_list::content`, `s_list::content_size`, and `s_list::next`.

**5.1.1.15 ft\_lstiter()** void ft\_lstiter (   
     t\_list \* lst,   
     void(\*) (t\_list \*elem) f )

Iterates the list `lst` and applies the function `f` to each link.

**Parameters**

<i>lst</i>	A pointer to the first link of a list.
<i>f</i>	The address of a function to apply to each link of a list.

Definition at line 21 of file `ft_lstiter.c`.

```
22 {
23     if (!f)
24         return ;
25     while (lst)
26     {
27         f(lst);
28         lst = lst->next;
29     }
30 }
```

References `s_list::next`.

**5.1.1.16 `ft_lstlast()`** `t_list*` `ft_lstlast` (  
    `t_list * lst` )

Return last element of the list.

**Parameters**

<i>lst</i>	A pointer's to the first link of a list.
------------	--

**Returns**

The last link of the list.

Definition at line 21 of file `ft_lstlast.c`.

```
22 {
23     if (!lst)
24         return (NULL);
25     while (lst->next)
26         lst = lst->next;
27     return (lst);
28 }
```

References `s_list::next`.

Referenced by `ft_lstappend()`.

**5.1.1.17 `ft_lstmap()`** `t_list*` `ft_lstmap` (  
    `t_list * lst`,  
    `t_list *(*)(t_list *elem) f` )

Iterates a list `lst` and applies the function `f` to each link to create a “fresh” list (using `malloc`) resulting from the successive applications of `f`. If the allocation fails, the function returns `NULL`.

## Parameters

<i>lst</i>	A pointer's to the first link of a list.
<i>f</i>	The address of a function to apply to each link of a list.

## Returns

The new list.

## Remarks

This function fails if *f* returns `NULL`;

Definition at line 26 of file `ft_lstmap.c`.

```
27 {  
28     t_list    *new;  
29     t_list    *tmp;  
30  
31     if (!lst)  
32         return (NULL);  
33     tmp = f(lst);  
34     new = tmp;  
35     while (lst->next)  
36     {  
37         lst = lst->next;  
38         tmp->next = f(lst);  
39         tmp = tmp->next;  
40     }  
41     return (new);  
42 }
```

References `s_list::next`.

**5.1.1.18 ft\_lstnew()** `t_list* ft_lstnew (`  
    `void const * content,`  
    `size_t content_size )`

Allocates (with `malloc`) and returns a “fresh” link. The variables `content` and `content_size` of the new link are initialized by copy of the parameters of the function. If the parameter `content` is `NULL`, the variable `content` is initialized to `NULL` and the variable `content_size` is initialized to 0 even if the parameter `content_size` isn't. The variable `next` is initialized to `NULL`. If the allocation fails, the function returns `NULL`.

## Parameters

<i>content</i>	The content to put in the new link.
<i>content_size</i>	The size of the content of the new link.

## Returns

The new link.

Definition at line 29 of file `ft_lstnew.c`.

```
30 {  
31     t_list    *link;  
32  
33     if (!(link = (t_list *)malloc(sizeof(t_list))))
```



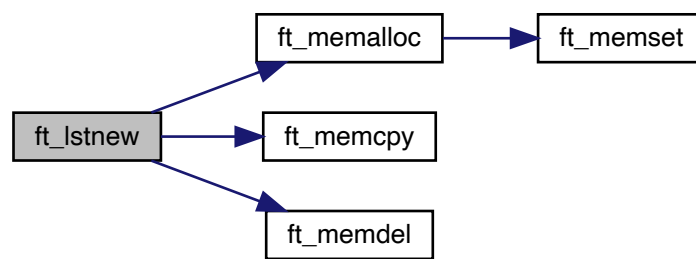
```

34     return (NULL);
35     if (content)
36     {
37         if (!(link->content = ft_memalloc(content_size)))
38         {
39             ft_memdel((void **)&link);
40             return (NULL);
41         }
42         ft_memcpy(link->content, content, content_size);
43         link->content_size = content_size;
44     }
45     else
46     {
47         link->content = NULL;
48         link->content_size = 0;
49     }
50     link->next = NULL;
51     return (link);
52 }

```

References `s_list::content`, `s_list::content_size`, `ft_memalloc()`, `ft_memcpy()`, `ft_memdel()`, and `s_list::next`.

Here is the call graph for this function:



**5.1.1.19 ft\_max()** `int ft_max (`  
`int a,`  
`int b )`

Returns the maximum of two integer arguments.

#### Note

This function is only needed, since "The Norme" (the code standard at School 42) forbids the use of parametrized macros.

#### Parameters

in	<i>a</i>	The first integer to compare.
in	<i>b</i>	The second integer to compare.

### Returns

The larger of the two integers.

Definition at line 22 of file ft\_max.c.

```
23 {  
24     return (a > b ? a : b);  
25 }
```

**5.1.1.20 ft\_memalloc()** void\* ft\_memalloc (  
size\_t size )

Allocates (with malloc) and returns a “fresh” memory area. The memory allocated is initialized to 0. If the allocation fails, the function returns NULL.

### Parameters

size	The size of the memory that needs to be allocated.
------	--

### Returns

The allocated memory area.

Definition at line 24 of file ft\_memalloc.c.

```
25 {  
26     char    *mem;  
27  
28     mem = malloc(size);  
29     if (mem == NULL)  
30         return (NULL);  
31     ft_memset(mem, 0, size);  
32     return (mem);  
33 }
```

References ft\_memset().

Referenced by ft\_lstnew(), ft\_stnew(), and ft\_strsplit().

Here is the call graph for this function:



```
5.1.1.21 ft_memccpy() void* ft_memccpy (
    void * dst,
    const void * src,
    int c,
    size_t n )
```

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_memccpy.c.

```
20 {
21     size_t      i;
22     unsigned char *s1;
23     unsigned char *s2;
24     unsigned char c1;
25
26     s1 = (unsigned char *)dst;
27     s2 = (unsigned char *)src;
28     c1 = (unsigned char)c;
29     i = 0;
30     while (i < n)
31     {
32         s1[i] = s2[i];
33         if (s1[i] == c1)
34             return (s1 + i + 1);
35         i++;
36     }
37     return (NULL);
38 }
```

```
5.1.1.22 ft_memchr() void* ft_memchr (
    const void * s,
    int c,
    size_t n )
```

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_memchr.c.

```
20 {
21     size_t      i;
22     unsigned char *s1;
23     unsigned char c1;
24
25     i = 0;
26     s1 = (unsigned char *)s;
27     c1 = (unsigned char)c;
28     while (i < n)
29     {
30         if (s1[i] == c1)
31             return (s1 + i);
32         i++;
33     }
34     return (NULL);
35 }
```

```
5.1.1.23 ft_memcmp() int ft_memcmp (
    const void * s1,
    const void * s2,
    size_t n )
```

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_memcmp.c.

```
20 {
21     unsigned char *str1;
22     unsigned char *str2;
```

```

23     size_t          i;
24
25     i = 0;
26     str1 = (unsigned char *)s1;
27     str2 = (unsigned char *)s2;
28     while (i < n)
29     {
30         if (str1[i] - str2[i])
31             return (str1[i] - str2[i]);
32         i++;
33     }
34     return (0);
35 }

```

**5.1.1.24 ft\_memcpy()** void\* ft\_memcpy (

```

    void * dst,
    const void * src,
    size_t n )

```

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_memcpy.c.

```

20 {
21     size_t          i;
22     unsigned char   *dst1;
23     unsigned char   *src1;
24
25     if (!dst && !src)
26         return (NULL);
27     i = 0;
28     dst1 = (unsigned char *)dst;
29     src1 = (unsigned char *)src;
30     while (i < n)
31     {
32         dst1[i] = src1[i];
33         i++;
34     }
35     return (dst);
36 }

```

Referenced by ft\_lstnew(), and ft\_strlcpy().

**5.1.1.25 ft\_memdel()** void ft\_memdel (

```

    void ** ap )

```

Takes as a parameter the address of a memory area that needs to be freed with free, then puts the pointer to NULL.

#### Parameters

<i>ap</i>	A pointer's address that needs its memory freed and set to NULL.
-----------	--

Definition at line 22 of file ft\_memdel.c.

```

23 {
24     if (!ap || !(*ap))
25         return ;
26     free(*ap);
27     *ap = 0;
28 }

```

Referenced by ft\_lstnew(), and ft\_strdel().

**5.1.1.26 ft\_memmove()** void\* ft\_memmove (  
    void \* dst,  
    const void \* src,  
    size\_t n )

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_memmove.c.

```
20 {  
21     unsigned char    *s1;  
22     unsigned char    *s2;  
23  
24     if (!dst && !src)  
25         return (NULL);  
26     s1 = (unsigned char *)dst;  
27     s2 = (unsigned char *)src;  
28     while (n > 0)  
29     {  
30         if (s1 < s2)  
31             *(s1++) = *(s2++);  
32         else  
33             s1[n - 1] = s2[n - 1];  
34         n--;  
35     }  
36     return (dst);  
37 }
```

**5.1.1.27 ft\_memset()** void\* ft\_memset (  
    void \* b,  
    int c,  
    size\_t len )

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_memset.c.

```
20 {  
21     unsigned char    *s;  
22     unsigned char    c1;  
23  
24     s = (unsigned char *)b;  
25     c1 = (unsigned char)c;  
26     while (len-- > 0)  
27     {  
28         *s = c;  
29         s++;  
30     }  
31     return (b);  
32 }
```

Referenced by ft\_memalloc(), and ft\_strclr().

**5.1.1.28 ft\_min()** int ft\_min (  
    int a,  
    int b )

Returns the minimum of two integer arguments.

#### Note

This function is only needed, since "The Norme" (the code standard at School 42) forbids the use of parametrized macros.

**Parameters**

in	<i>a</i>	The first integer to compare.
in	<i>b</i>	The second integer to compare.

**Returns**

The smaller of the two integers.

Definition at line 22 of file `ft_min.c`.

```
23 {  
24     return (a < b ? a : b);  
25 }
```

**5.1.1.29 ft\_power()** int ft\_power (  
 int *num*,  
 unsigned int *exponent* )

Raises a number to a given power.

**Parameters**

<i>num</i>	The base.
<i>exponent</i>	The exponent.

**Returns**

The result or 0 if an integer overflow occurred.

Definition at line 23 of file `ft_power.c`.

```
24 {  
25     long long result;  
26  
27     result = 1;  
28     while (exponent-- > 0)  
29         result *= num;  
30     if (result > INT_MAX)  
31         return (0);  
32     else  
33         return ((int)result);  
34 }
```

**5.1.1.30 ft\_putchar()** void ft\_putchar (  
 char *c* )

Outputs the character *c* to the standard output.

**Parameters**

<i>c</i>	The character to output.
----------	--------------------------

Definition at line 21 of file `ft_putchar.c`.

```
22 {  
23     ft_putchar_fd(c, 1);  
24 }
```

References `ft_putchar_fd()`.

Here is the call graph for this function:



**5.1.1.31 ft\_putchar\_fd()** `void ft_putchar_fd (`  
    `char c,`  
    `int fd )`

Outputs the char `c` to the file descriptor `fd`.

#### Parameters

<code>c</code>	The character to output.
<code>fd</code>	The file descriptor.

Definition at line 22 of file `ft_putchar_fd.c`.

```
23 {  
24     write(fd, &c, 1);  
25 }
```

Referenced by `ft_putchar()`, `ft_putendl_fd()`, and `ft_putnbr_fd()`.

**5.1.1.32 ft\_putendl()** `void ft_putendl (`  
    `char const * s )`

Outputs the string `s` to the standard output followed by a `'\n'`.

#### Parameters

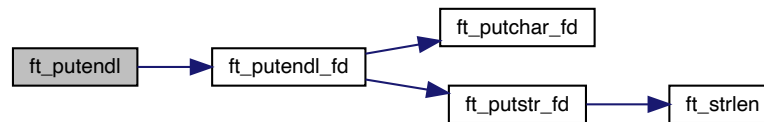
<code>s</code>	The string to output.
----------------	-----------------------

Definition at line 20 of file `ft_putendl.c`.

```
21 {  
22     ft_putendl_fd(s, 1);  
23 }
```

References `ft_putendl_fd()`.

Here is the call graph for this function:



**5.1.1.33 ft\_putendl\_fd()** `void ft_putendl_fd (`  
     `char const * s,`  
     `int fd )`

Outputs the string `s` to the file descriptor `fd` followed by a `'\n'`.

Parameters

<code>s</code>	The string to output.
<code>fd</code>	The file descriptor.

Definition at line 21 of file `ft_putendl_fd.c`.

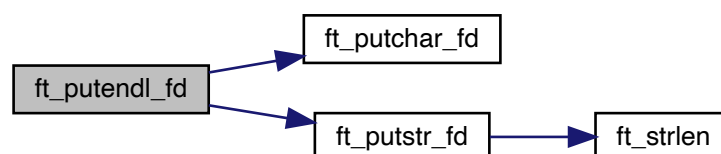
```

22 {
23     ft_putstr_fd(s, fd);
24     ft_putchar_fd('\n', fd);
25 }
  
```

References `ft_putchar_fd()`, and `ft_putstr_fd()`.

Referenced by `ft_putendl()`.

Here is the call graph for this function:





**5.1.1.34 ft\_putnbr()** void ft\_putnbr (  
int n )

Outputs the integer `n` to the standard output.

#### Parameters

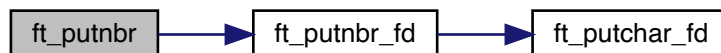
<code>n</code>	The integer to output.
----------------	------------------------

Definition at line 20 of file `ft_putnbr.c`.

```
21 {  
22     ft_putnbr_fd(n, 1);  
23 }
```

References `ft_putnbr_fd()`.

Here is the call graph for this function:



**5.1.1.35 ft\_putnbr\_fd()** void ft\_putnbr\_fd (  
int n,  
int fd )

Outputs the integer `n` to the file descriptor `fd`.

#### Parameters

<code>n</code>	The integer to print.
<code>fd</code>	The file descriptor.

Definition at line 22 of file `ft_putnbr_fd.c`.

```
23 {  
24     if (n == INT_MIN)  
25     {  
26         ft_putnbr_fd(n / 10, fd);  
27         ft_putnbr_fd(-(n % 10), fd);  
28     }  
29     else if (n < 0)  
30     {  
31         ft_putchar_fd('-', fd);  
32         ft_putnbr_fd(-n, fd);  
33     }  
34     else if (n > 9)  
35     {  
36         ft_putnbr_fd(n / 10, fd);  
37         ft_putchar_fd(n % 10 + '0', fd);  
38     }  
39     else
```

```

40         ft_putchar_fd(n % 10 + '0', fd);
41     }

```

References `ft_putchar_fd()`.

Referenced by `ft_putnbr()`.

Here is the call graph for this function:



**5.1.1.36 ft\_putnstr()** void ft\_putnstr (

```

    char * s,
    size_t n )

```

Outputs the first `n` characters of the string `s` to the standard output.

#### Parameters

<code>s</code>	The string, the characters of which to output.
<code>n</code>	The number of characters to output.

#### Remarks

If `s` contains less than `n` characters, behaviour is undefined.

Definition at line 23 of file `ft_putnstr.c`.

```

24 {
25     write(1, s, n);
26 }

```

**5.1.1.37 ft\_putnstr\_fd()** void ft\_putnstr\_fd (

```

    char * s,
    size_t n,
    int fd )

```

Outputs the first `n` characters of the string `s` to the file descriptor `fd`.

#### Parameters

<code>s</code>	The string, the characters of which to output.
<code>n</code>	The number of characters to output.
<code>fd</code>	The file descriptor.

**Remarks**

If `s` contains less than `n` characters, behaviour is undefined.

Definition at line 24 of file `ft_putnstr_fd.c`.

```
25 {  
26     write(fd, s, n);  
27 }
```

**5.1.1.38 ft\_puts()** void ft\_puts (  
const char \* s )

Replicates behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_puts.c`.

```
20 {  
21     return (write(1, s, ft_strlen(s)));  
22 }
```

References `ft_strlen()`.

Here is the call graph for this function:



**5.1.1.39 ft\_putstr()** void ft\_putstr (  
char const \* s )

Outputs the string `s` to the standard output.

**Parameters**

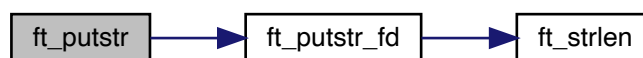
<code>s</code>	The string to output.
----------------	-----------------------

Definition at line 21 of file `ft_putstr.c`.

```
22 {  
23     ft_putstr_fd(s, 1);  
24 }
```

References `ft_putstr_fd()`.

Here is the call graph for this function:



**5.1.1.40 ft\_putstr\_fd()** `void ft_putstr_fd (`  
    `char const * s,`  
    `int fd )`

Outputs the string *s* to the file descriptor *fd*.

#### Parameters

<i>s</i>	The string to output.
<i>fd</i>	The file descriptor.

Definition at line 22 of file `ft_putstr_fd.c`.

```
23 {  
24     write(fd, s, ft_strlen(s));  
25 }
```

References `ft_strlen()`.

Referenced by `ft_putendl_fd()`, and `ft_putstr()`.

Here is the call graph for this function:



**5.1.1.41 ft\_sqrt()** `int ft_sqrt (`  
    `int num )`

Computes an integer square root of a given number.

**Parameters**

<i>num</i>	The number of which to take a square root.
------------	--

**Returns**

The integer square root, or  $-1$  if it doesn't exit.

Definition at line 19 of file `ft_sqrt.c`.

```
20 {
21     int factor;
22
23     if (num < 0 ||
24         (num % 2 == 0 && num % 4 != 0) ||
25         (num % 3 == 0 && num % 9 != 0))
26         return (-1);
27     factor = (num % 2) ? 1 : 0;
28     while (factor < num / 2)
29     {
30         if (factor * factor == num)
31             return (factor);
32         factor += 2;
33     }
34     return (-1);
35 }
```

**5.1.1.42 ft\_strcapitalize()** `char* ft_strcapitalize (`  
    `const char * s1 )`

Capitalizes all words (defined as stretches of alpha-numeric characters) in a `NULL`-terminated string and writes them to a newly allocated string.

Example: `ft_strcapitalize("My word IS 42about%8THEM")` returns `"My Word Is 42about%8them"`

**Parameters**

<i>s1</i>	The string to capitalize.
-----------	---------------------------

**Returns**

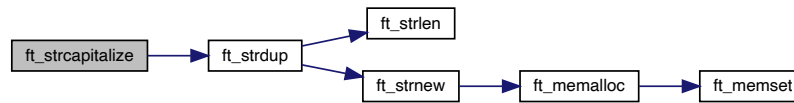
A duplicate of `str` in which all words have been capitalized. If `str` is a `NULL` pointer or allocation fails, `NULL` is returned.

Definition at line 49 of file `ft_strcapitalize.c`.

```
50 {
51     char    *s2;
52
53     if (!s1 || !(s2 = ft_strdup(s1)))
54         return (0);
55     while (scroll_to_word(&s2))
56         capitalize_word(&s2);
57     return (s2);
58 }
```

References `ft_strdup()`.

Here is the call graph for this function:



**5.1.1.43 ft\_strcat()** `char* ft_strcat (`  
     `char * s1,`  
     `const char * s2 )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strcat.c`.

```

20 {
21     size_t i;
22     size_t len;
23
24     len = ft_strlen(s1);
25     i = 0;
26     while (s2[i])
27     {
28         s1[len + i] = s2[i];
29         i++;
30     }
31     s1[len + i] = '\0';
32     return (s1);
33 }
```

References `ft_strlen()`.

Referenced by `ft_strjoin()`.

Here is the call graph for this function:



**5.1.1.44 ft\_strchr()** `int ft_strchr (`  
     `char const * str,`  
     `char c )`

Count the number of occurrences of a character in a string.

**Parameters**

<i>str</i>	The string in which to search.
<i>c</i>	The character for which to search.

**Returns**

The number of occurrences.

Definition at line 22 of file `ft_strchr.c`.

```
23 {
24     int count;
25
26     count = 0;
27     if (!str || !c)
28         return (-1);
29     while (*str)
30     {
31         if (*str == c)
32             count++;
33         str++;
34     }
35     return (count);
36 }
```

**5.1.1.45 ft\_strchr()** `char* ft_strchr (`  
    `const char * s,`  
    `int c )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strchr.c`.

```
20 {
21     size_t i;
22
23     i = 0;
24     while (s[i])
25     {
26         if (s[i] == c)
27             return ((char *) (s + i));
28         i++;
29     }
30     if (c == 0)
31         return ((char *) (s + i));
32     else
33         return (NULL);
34 }
```

**5.1.1.46 ft\_strclr()** `void ft_strclr (`  
    `char * s )`

Sets every character of the string to the value `'\0'`.

**Parameters**

<i>s</i>	The string that needs to be cleared.
----------	--------------------------------------

Definition at line 20 of file `ft_strclr.c`.

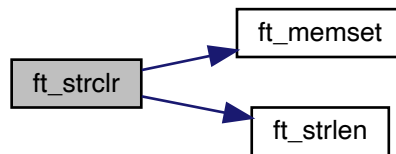
```

21 {
22     if (!s)
23         return ;
24     ft_memset(s, 0, ft_strlen(s));
25 }

```

References `ft_memset()`, and `ft_strlen()`.

Here is the call graph for this function:



**5.1.1.47 ft\_strcmp()** `int ft_strcmp (`  
`const char * s1,`  
`const char * s2 )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strcmp.c`.

```

20 {
21     unsigned char *s1u;
22     unsigned char *s2u;
23
24     s1u = (unsigned char *)s1;
25     s2u = (unsigned char *)s2;
26     while (*s1u || *s2u)
27     {
28         if (*s1u - *s2u)
29             return (*s1u - *s2u);
30         s1u++;
31         s2u++;
32     }
33     return (0);
34 }

```

**5.1.1.48 ft\_strcpy()** `char* ft_strcpy (`  
`char * dst,`  
`const char * src )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strcpy.c`.

```

20 {
21     size_t i;
22
23     i = 0;
24     while (src[i])
25     {
26         dst[i] = src[i];
27         i++;
28     }
29     dst[i] = 0;
30     return (dst);
31 }

```

Referenced by `ft_strjoin()`.



**5.1.1.49 ft\_strdel()** void ft\_strdel (  
char \*\* as )

Takes as a parameter the address of a string that need to be freed with `free`, then sets its pointer to `NULL`.

#### Parameters

<b>as</b>	The string's address that needs to be freed and its pointer set to <code>NULL</code> .
-----------	--

Definition at line 22 of file `ft_strdel.c`.

```
23 {
24     ft_memdel((void **)as);
25 }
```

References `ft_memdel()`.

Here is the call graph for this function:



**5.1.1.50 ft\_strdup()** char\* ft\_strdup (  
const char \* s1 )

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

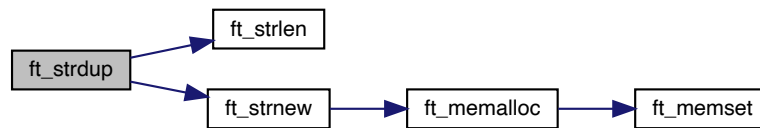
Definition at line 19 of file `ft_strdup.c`.

```
20 {
21     size_t i;
22     char *cpy;
23
24     if (!(cpy = ft_strnew(ft_strlen(s1))))
25         return (NULL);
26     i = 0;
27     while (s1[i])
28     {
29         cpy[i] = s1[i];
30         i++;
31     }
32     cpy[i] = 0;
33     return (cpy);
34 }
```

References `ft_strlen()`, and `ft_strnew()`.

Referenced by `ft_strcapitalize()`, `ft_strmap()`, `ft_strmapi()`, and `ft_strtrim()`.

Here is the call graph for this function:



**5.1.1.51 ft\_strequ()** `int ft_strequ (`  
     `const char * s1,`  
     `const char * s2 )`

Lexicographical comparison between `s1` and `s2` up to `n` characters or until a `'\0'` is reached. If the 2 strings are identical, the function returns 1, or 0 otherwise.

#### Parameters

<code>s1</code>	The first string to be compared.
<code>s2</code>	The second string to be compared.

#### Returns

1 or 0 according to if the 2 strings are identical or not.

Definition at line 24 of file `ft_strequ.c`.

```

25 {
26     if (!s1 && !s2)
27         return (1);
28     else if (!s1 || !s2)
29         return (0);
30     while (*s1 || *s2)
31     {
32         if (*s1 != *s2)
33             return (0);
34         s1++;
35         s2++;
36     }
37     return (1);
38 }
  
```

Referenced by `ft_strintab()`.

**5.1.1.52 ft\_strintab()** `int ft_strintab (`  
     `const char * str,`  
     `char *const tab[] )`

Perform lexicographical comparison between a given string and strings contained in a `NULL`-terminated `tab`. If the `tab` contains an identical string, the function returns 1, or 0 otherwise.

**Parameters**

<i>str</i>	The string to search for.
<i>tab</i>	The NULL-terminated tab to search in.

**Returns**

1 or 0 depending on whether the tab contains an identical string.

Definition at line 25 of file ft\_strintab.c.

```
26 {
27     while (*tab)
28     {
29         if (ft_strequ(str, *tab))
30             return (1);
31         tab++;
32     }
33     return (0);
34 }
```

References ft\_strequ().

Here is the call graph for this function:



**5.1.1.53 ft\_striter()** void ft\_striter (  
char \* *s*,  
void(\*) (char \*) *f* )

Applies the function *f* to each character of the string passed as argument. Each character is passed by address to *f* to be modified if necessary.

**Parameters**

<i>s</i>	The string to iterate.
<i>f</i>	The function to apply to each character of <i>s</i> .

Definition at line 22 of file ft\_striter.c.

```
23 {
24     if (!s || !f)
25         return ;
26     while (*s)
27         f(s++);
28 }
```

**5.1.1.54 ft\_striteri()** void ft\_striteri (  
char \* s,  
void(\*) (unsigned int, char \*) f )

Applies the function `f` to each character of the string passed as argument, and passing its index as first argument. Each character is passed by address to `f` to be modified if necessary.

#### Parameters

<b>s</b>	The string to iterate.
<b>f</b>	The function to apply to each character of <code>s</code> and its index.

Definition at line 23 of file `ft_striteri.c`.

```

24 {
25     size_t i;
26
27     if (!s || !f)
28         return ;
29     i = 0;
30     while (*s)
31         f(i++, s++);
32 }
```

**5.1.1.55 ft\_strjoin()** char\* ft\_strjoin (  
const char \* s1,  
const char \* s2 )

Allocates (with `malloc`) and returns a “fresh” string ending with `'\0'`, result of the concatenation of `s1` and `s2`. If the allocation fails the function returns `NULL`.

#### Parameters

<b>s1</b>	The prefix string.
<b>s2</b>	The suffix string.

#### Returns

The “fresh” string result of the concatenation of the 2 strings.

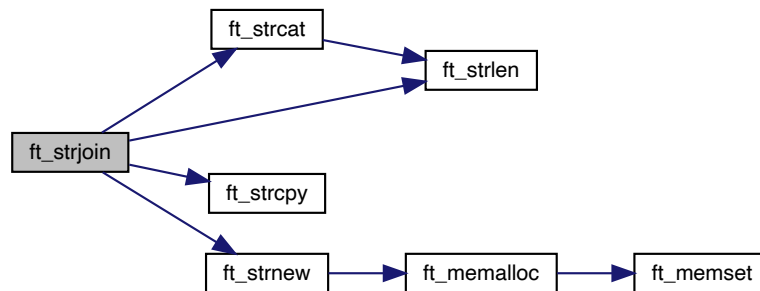
Definition at line 24 of file `ft_strjoin.c`.

```

25 {
26     char *s_joined;
27
28     if (!s1 && !s2)
29         return (NULL);
30     else if (!s1)
31         return (char *)s2;
32     else if (!s2)
33         return (char *)s1;
34     if (!(s_joined = ft_strnew(ft_strlen(s1) + ft_strlen(s2))))
35         return (NULL);
36     s_joined = ft_strcpy(s_joined, s1);
37     s_joined = ft_strcat(s_joined, s2);
38     return (s_joined);
39 }
```

References `ft_strcat()`, `ft_strcpy()`, `ft_strlen()`, and `ft_strnew()`.

Here is the call graph for this function:



**5.1.1.56 ft\_strlast()** `char ft_strlast (char const * str)`

Returns the last characters (excluding NULL-termination) of a string.

#### Parameters

<i>str</i>	The string.
------------	-------------

#### Returns

The last character of the string, or 0 if it is empty.

Definition at line 21 of file `ft_strlast.c`.

```

22 {
23     size_t i;
24
25     i = 0;
26     while (str[i + 1])
27         i++;
28     return (str[i]);
29 }
  
```

**5.1.1.57 ft\_strlcat()** `size_t ft_strlcat (char * dst, const char * src, size_t dstsize)`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strlcat.c`.

```

20 {
21     size_t dstlen;
22     size_t srclen;
  
```

```

23     size_t i;
24
25     dstlen = ft_strlen(dst);
26     srclen = ft_strlen(src);
27     if (dstlen >= dstsize)
28         return (srclen + dstsize);
29     i = 0;
30     while (i < dstsize - dstlen - 1 && src[i])
31     {
32         dst[dstlen + i] = src[i];
33         i++;
34     }
35     dst[dstlen + i] = '\0';
36     return (srclen + dstlen);
37 }

```

References `ft_strlen()`.

Here is the call graph for this function:



**5.1.1.58 ft\_strlcpy()** `size_t ft_strlcpy (`  
`char * dst,`  
`const char * src,`  
`size_t maxlen )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strlcpy.c`.

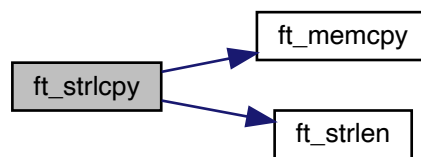
```

20 {
21     size_t srclen;
22
23     srclen = ft_strlen(src);
24     if (srclen + 1 < maxlen)
25     {
26         ft_memcpy(dst, src, srclen + 1);
27     }
28     else if (maxlen != 0)
29     {
30         ft_memcpy(dst, src, maxlen - 1);
31         dst[maxlen - 1] = '\0';
32     }
33     return (srclen);
34 }

```

References `ft_memcpy()`, and `ft_strlen()`.

Here is the call graph for this function:



**5.1.1.59 ft\_strlen()** `size_t ft_strlen (const char * s )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strlen.c`.

```

20 {
21     size_t i;
22
23     i = 0;
24     while (s[i])
25         i++;
26     return (i);
27 }
  
```

Referenced by `ft_puts()`, `ft_putstr_fd()`, `ft_strcat()`, `ft_strclr()`, `ft_strdup()`, `ft_strjoin()`, `ft_strlcat()`, `ft_strlcpy()`, `ft_strncat()`, `ft_strrev()`, and `ft_strtrim()`.

**5.1.1.60 ft\_strmap()** `char* ft_strmap (char const * s, char(*) (char) f )`

Applies the function `f` to each character of the string given as argument to create a “fresh” new string (with `malloc`) resulting from the successive applications of `f`.

#### Parameters

<code>s</code>	The string to map.
<code>f</code>	The function to apply to each character of <code>s</code> .

#### Returns

The “fresh” string created from the successive applications of `f`.

Definition at line 24 of file `ft_strmap.c`.

```

25 {
  
```

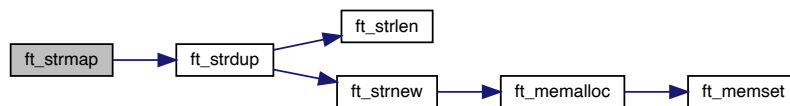
```

26     char    *s_new;
27     size_t   i;
28
29     if (!s || !f)
30         return (NULL);
31     s_new = ft_strdup(s);
32     if (!s_new)
33         return (NULL);
34     i = 0;
35     while (s[i])
36     {
37         s_new[i] = f(s[i]);
38         i++;
39     }
40     return (s_new);
41 }

```

References `ft_strdup()`.

Here is the call graph for this function:



**5.1.1.61 ft\_strmapi()** `char* ft_strmapi (`  
`char const * s,`  
`char(*) (unsigned int, char) f )`

Applies the function `f` to each character of the string passed as argument by giving its index as first argument to create a “fresh” new string (with `malloc`) resulting from the successive applications of `f`.

#### Parameters

<code>s</code>	The string to map.
<code>f</code>	The function to apply to each character and its index of <code>s</code> .

#### Returns

The “fresh” string created from the successive applications of `f`.

Definition at line 25 of file `ft_strmap.c`.

```

26 {
27     char    *s_new;
28     size_t   i;
29
30     if (!s || !f)
31         return (NULL);
32     s_new = ft_strdup(s);
33     if (!s_new)
34         return (NULL);
35     i = 0;
36     while (s[i])
37     {
38         s_new[i] = f(i, s[i]);
39         i++;

```



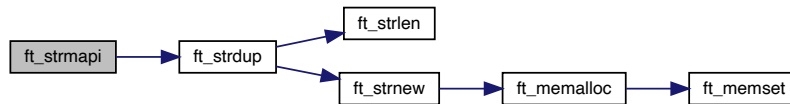
```

40     }
41     return (s_new);
42 }

```

References `ft_strdup()`.

Here is the call graph for this function:



**5.1.1.62 ft\_strncat()** `char* ft_strncat (`  
`char * s1,`  
`const char * s2,`  
`size_t n )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strncat.c`.

```

20 {
21     size_t i;
22     size_t len;
23
24     i = 0;
25     len = ft_strlen(s1);
26     while (i < n && s2[i])
27     {
28         s1[len + i] = s2[i];
29         i++;
30     }
31     s1[len + i] = 0;
32     return (s1);
33 }

```

References `ft_strlen()`.

Here is the call graph for this function:



**5.1.1.63 ft\_strncmp()** int ft\_strncmp (

```

    const char * s1,
    const char * s2,
    size_t n )

```

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_strncmp.c.

```

20 {
21     unsigned char *s1u;
22     unsigned char *s2u;
23
24     s1u = (unsigned char *)s1;
25     s2u = (unsigned char *)s2;
26     while ((*s1u || *s2u) && n-- > 0)
27     {
28         if (*s1u - *s2u)
29             return (*s1u - *s2u);
30         s1u++;
31         s2u++;
32     }
33     return (0);
34 }

```

**5.1.1.64 ft\_strncpy()** char\* ft\_strncpy (

```

    char * dst,
    const char * src,
    size_t len )

```

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_strncpy.c.

```

20 {
21     size_t i;
22
23     i = 0;
24     while (i < len && src[i])
25     {
26         dst[i] = src[i];
27         i++;
28     }
29     while (i < len)
30     {
31         dst[i] = '\0';
32         i++;
33     }
34     return (dst);
35 }

```

Referenced by ft\_strsub().

**5.1.1.65 ft\_strndup()** char\* ft\_strndup (

```

    const char * s1,
    size_t len )

```

Allocate sufficient memory for a string of len characters, do the copy of len characters, NULL terminate the string, and return a pointer to it. The pointer may subsequently be used as an argument to the function free.

#### Parameters

<i>s1</i>	String to be copied from.
<i>len</i>	Number of characters to copy.

**Returns**

String of length `len` with copied characters.

**Remarks**

If `s1` contains less than `len` characters, behaviour is undefined.

Definition at line 27 of file `ft_strndup.c`.

```

28 {
29     size_t  i;
30     char    *cpy;
31
32     if (!(cpy = (char *)malloc(sizeof(char) * (len + 1))))
33         return (NULL);
34     i = 0;
35     while (i < len)
36     {
37         cpy[i] = s1[i];
38         i++;
39     }
40     cpy[i] = '\0';
41     return (cpy);
42 }
```

Referenced by `ft_strsplit()`.

**5.1.1.66 ft\_strnequ()** `int ft_strnequ (`  
     `char const * s1,`  
     `char const * s2,`  
     `size_t n )`

Lexicographical comparison between `s1` and `s2` up to `n` characters or until a `'\0'` is reached. If the 2 strings are identical, the function returns 1, or 0 otherwise.

**Parameters**

<i>s1</i>	The first string to be compared.
<i>s2</i>	The second string to be compared.
<i>n</i>	The maximum number of characters to be compared.

**Returns**

1 or 0 according to if the 2 strings are identical or not.

Definition at line 25 of file `ft_strnequ.c`.

```

26 {
27     if (!s1 && !s2)
28         return (1);
29     else if (!s1 || !s2)
30         return (0);
31     while ((*s1 || *s2) && n-- > 0)
32     {
33         if (*s1 != *s2)
34             return (0);
35         s1++;
36         s2++;
37     }
38     return (1);
39 }
```

**5.1.1.67 ft\_strnew()** `char* ft_strnew (`  
`size_t size )`

Allocates (with `malloc`) and returns a “fresh” string ending with `'\0'`. Each character of the string is initialized at `'\0'`. If the allocation fails the function returns `NULL`.

#### Parameters

<i>size</i>	The size of the string to be allocated.
-------------	---

#### Returns

The string allocated and initialized to 0.

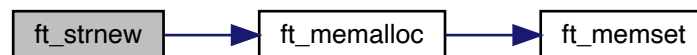
Definition at line 23 of file `ft_strnew.c`.

```
24 {
25     if (size + 1 < size)
26         return (NULL);
27     return (ft_memalloc(sizeof(char) * (size + 1)));
28 }
```

References `ft_memalloc()`.

Referenced by `ft_itoa()`, `ft_strdup()`, `ft_strjoin()`, `ft_strrev()`, and `ft_strsub()`.

Here is the call graph for this function:



**5.1.1.68 ft\_strnstr()** `char* ft_strnstr (`  
`const char * haystack,`  
`const char * needle,`  
`size_t len )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strnstr.c`.

```
20 {
21     size_t i;
22     size_t j;
23
24     i = 0;
25     if (*needle == '\0')
26         return ((char *)haystack);
27     while (haystack[i] && i < len)
28     {
29         if (haystack[i] == needle[0])
30         {
31             j = 0;
32             while (needle[j] &&
```

```
33             i + j < len &&
34             haystack[i + j] == needle[j])
35         j++;
36         if (needle[j] == 0)
37             return ((char *) (haystack + i));
38     }
39     i++;
40 }
41 return (NULL);
42 }
```

**5.1.1.69 ft\_strchr()** char\* ft\_strchr (  
const char \* s,  
int c )

Replicate behaviour of a function of the same name (sans ft\_) from libc.

Definition at line 19 of file ft\_strchr.c.

```
20 {
21     size_t i;
22     size_t last;
23     int found;
24
25     i = 0;
26     last = 0;
27     found = 0;
28     while (s[i])
29     {
30         if (s[i] == c)
31         {
32             last = i;
33             found = 1;
34         }
35         i++;
36     }
37     if (found)
38         return ((char *) (s + last));
39     else if (c == 0)
40         return ((char *) (s + i));
41     else
42         return (NULL);
43 }
```

**5.1.1.70 ft\_strrev()** char\* ft\_strrev (  
const char \* s )

Allocates (with malloc) and returns a “reversed” NULL-terminated string or NULL if allocation fails.

Example: ft\_strrev("0123456789") returns "9876543210"

#### Parameters

s	String to be reversed.
---	------------------------

#### Returns

Reversed string.

Definition at line 24 of file ft\_strrev.c.

```
25 {
26     size_t len;
```

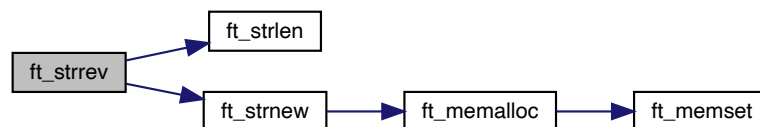
```

27     size_t  i;
28     char    *srev;
29
30     if (!s)
31         return (NULL);
32     len = ft_strlen(s);
33     if (!(srev = ft_strnew(len)))
34         return (NULL);
35     i = 0;
36     while (i < len)
37     {
38         srev[i] = s[len - i - 1];
39         i++;
40     }
41     return (srev);
42 }

```

References `ft_strlen()`, and `ft_strnew()`.

Here is the call graph for this function:



**5.1.1.71 ft\_strsplit()** `char** ft_strsplit (`  
`char const * s,`  
`char delim )`

Allocates (with `malloc`) and returns an array of “fresh” strings (all ending with `'\0'`, including the array itself) obtained by splitting `s` using the character `c` as a delimiter. If the allocation fails the function returns `NULL`.

Example : `ft_strsplit("hello*fellow***students*", '*')` returns the array `["hello", "fellow", "students"]`.

#### Parameters

<i>s</i>	The string to split.
<i>delim</i>	The delimiter character.

#### Returns

The array of “fresh” strings result of the split.

Definition at line 69 of file `ft_strsplit.c`.

```

70 {
71     char    **tab;
72     const char *start;
73     const char *end;
74     size_t  wcount;
75     size_t  i;
76

```

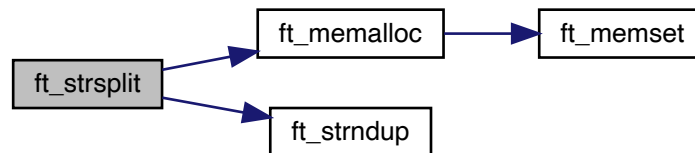
```

77     wcount = count_words(s, delim);
78     if (!(tab = ft_memalloc(sizeof(char *) * (wcount + 1))))
79         return (NULL);
80     i = 0;
81     end = s;
82     while (i < wcount)
83     {
84         start = search_not_delim(end, delim);
85         end = search_delim(start, delim);
86         if (!(tab[i] = ft_strndup(start, end - start)))
87         {
88             free_tab(tab);
89             return (NULL);
90         }
91         i++;
92     }
93     tab[i++] = NULL;
94     return (tab);
95 }

```

References `ft_memalloc()`, and `ft_strndup()`.

Here is the call graph for this function:



**5.1.1.72 ft\_strstr()** `char* ft_strstr (`  
`const char * haystack,`  
`const char * needle )`

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file `ft_strstr.c`.

```

20 {
21     size_t i;
22     size_t j;
23
24     i = 0;
25     if (*needle == 0)
26         return ((char *)haystack);
27     while (haystack[i])
28     {
29         if (haystack[i] == needle[0])
30         {
31             j = 0;
32             while (needle[j] && haystack[i + j] == needle[j])
33                 j++;
34             if (needle[j] == 0)
35                 return ((char *) (haystack + i));
36         }
37         i++;
38     }
39     return (NULL);
40 }

```

**5.1.1.73 ft\_strsub()** `char* ft_strsub (`  
    `char const * s,`  
    `unsigned int start,`  
    `size_t len )`

Allocates (with `malloc`) and returns a “fresh” substring from the string given as argument. The substring begins at index `start` and is of size `len`. If `start` and `len` aren’t referring to a valid substring, the behavior is undefined. If the allocation fails, the function returns `NULL`.

#### Parameters

<i>s</i>	The string from which create the substring.
<i>start</i>	The start index of the substring.
<i>len</i>	The size of the substring.

#### Returns

The substring.

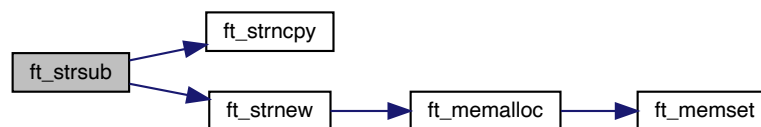
Definition at line 27 of file `ft_strsub.c`.

```
28 {  
29     char    *sub;  
30  
31     if (!s ||  
32         !(sub = ft_strnew(len)))  
33         return (NULL);  
34     ft_strncpy(sub, s + start, len);  
35     return (sub);  
36 }
```

References `ft_strncpy()`, and `ft_strnew()`.

Referenced by `ft_strtrim()`.

Here is the call graph for this function:



**5.1.1.74 ft\_strtrim()** `char* ft_strtrim (`  
    `char const * s )`

Allocates (with `malloc`) and returns a copy of the string given as argument without whitespaces at the beginning or at the end of the string. Will be considered as whitespaces the following characters ' ', '\n' and '\t'. If `s` has no whitespaces at the beginning or at the end, the function returns a copy of `s`. If the allocation fails the function returns `NULL`.



## Parameters

s	The string to be trimmed.
---	---------------------------

## Returns

The “fresh” trimmed string or a copy of s.

Definition at line 31 of file ft\_strtrim.c.

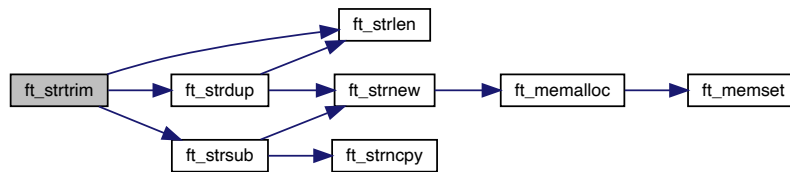
```

32 {
33     size_t i;
34     size_t j;
35
36     i = 0;
37     while (is_trimmable(s[i]))
38         i++;
39     if (s[i] == '\0')
40         return (ft_strdup(""));
41     j = ft_strlen(s) - 1;
42     while (is_trimmable(s[j]))
43         j--;
44     return (ft_strsub(s, i, j - i + 1));
45 }

```

References ft\_strdup(), ft\_strlen(), and ft\_strsub().

Here is the call graph for this function:



**5.1.1.75 ft\_tolower()** int ft\_tolower (  
int c )

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file ft\_tolower.c.

```

20 {
21     if (c >= 'A' && c <= 'Z')
22         return (c + 'a' - 'A');
23     else
24         return (c);
25 }

```

**5.1.1.76 ft\_toupper()** int ft\_toupper (  
int c )

Replicate behaviour of a function of the same name (sans `ft_`) from `libc`.

Definition at line 19 of file ft\_toupper.c.

```

20 {
21     if (c >= 'a' && c <= 'z')
22         return (c + 'A' - 'a');
23     else
24         return (c);
25 }

```

## Index

content  
  s\_list, [5](#)  
content\_size  
  s\_list, [5](#)  
  
ft\_abs  
  libft.h, [8](#)  
ft\_atoi  
  libft.h, [8](#)  
ft\_bzero  
  libft.h, [8](#)  
ft\_isalnum  
  libft.h, [9](#)  
ft\_isalpha  
  libft.h, [9](#)  
ft\_isascii  
  libft.h, [9](#)  
ft\_isdigit  
  libft.h, [10](#)  
ft\_isprint  
  libft.h, [10](#)  
ft\_isspace  
  libft.h, [10](#)  
ft\_itoa  
  libft.h, [10](#)  
ft\_lstadd  
  libft.h, [11](#)  
ft\_lstappend  
  libft.h, [12](#)  
ft\_lstdel  
  libft.h, [12](#)  
ft\_lstdelone  
  libft.h, [13](#)  
ft\_lstiter  
  libft.h, [13](#)  
ft\_lstlast  
  libft.h, [14](#)  
ft\_lstmap  
  libft.h, [14](#)  
ft\_lstnew  
  libft.h, [15](#)  
ft\_max  
  libft.h, [16](#)  
ft\_memalloc  
  libft.h, [17](#)  
ft\_memccpy  
  libft.h, [17](#)  
ft\_memchr  
  libft.h, [18](#)  
ft\_memcmp  
  libft.h, [18](#)  
ft\_memcpy  
  libft.h, [19](#)  
ft\_memdel  
  libft.h, [19](#)  
  
ft\_memmove  
  libft.h, [19](#)  
ft\_memset  
  libft.h, [20](#)  
ft\_min  
  libft.h, [20](#)  
ft\_power  
  libft.h, [21](#)  
ft\_putchar  
  libft.h, [21](#)  
ft\_putchar\_fd  
  libft.h, [22](#)  
ft\_putendl  
  libft.h, [22](#)  
ft\_putendl\_fd  
  libft.h, [23](#)  
ft\_putnbr  
  libft.h, [23](#)  
ft\_putnbr\_fd  
  libft.h, [24](#)  
ft\_putstr  
  libft.h, [25](#)  
ft\_putstr\_fd  
  libft.h, [25](#)  
ft\_puts  
  libft.h, [26](#)  
ft\_putstr  
  libft.h, [26](#)  
ft\_putstr\_fd  
  libft.h, [27](#)  
ft\_sqrt  
  libft.h, [27](#)  
ft\_strcapitalize  
  libft.h, [28](#)  
ft\_strcat  
  libft.h, [29](#)  
ft\_strchr  
  libft.h, [29](#)  
ft\_strchr  
  libft.h, [30](#)  
ft\_strclr  
  libft.h, [30](#)  
ft\_strcmp  
  libft.h, [31](#)  
ft\_strcpy  
  libft.h, [31](#)  
ft\_strdel  
  libft.h, [31](#)  
ft\_strdup  
  libft.h, [32](#)  
ft\_strequ  
  libft.h, [33](#)  
ft\_strintab  
  libft.h, [33](#)  
ft\_striter

libft.h, 34  
ft\_striteri  
libft.h, 34  
ft\_strjoin  
libft.h, 35  
ft\_strlast  
libft.h, 36  
ft\_strlcat  
libft.h, 36  
ft\_strlcpy  
libft.h, 37  
ft\_strlen  
libft.h, 38  
ft\_strmap  
libft.h, 38  
ft\_strmapi  
libft.h, 39  
ft\_strncat  
libft.h, 40  
ft\_strncmp  
libft.h, 40  
ft\_strncpy  
libft.h, 41  
ft\_strndup  
libft.h, 41  
ft\_strnequ  
libft.h, 42  
ft\_strnew  
libft.h, 42  
ft\_strnstr  
libft.h, 43  
ft\_strchr  
libft.h, 44  
ft\_strev  
libft.h, 44  
ft\_strsplit  
libft.h, 45  
ft\_strstr  
libft.h, 46  
ft\_strsub  
libft.h, 46  
ft\_strtrim  
libft.h, 47  
ft\_tolower  
libft.h, 48  
ft\_toupper  
libft.h, 48  
libft.h, 6  
ft\_abs, 8  
ft\_atoi, 8  
ft\_bzero, 8  
ft\_isalnum, 9  
ft\_isalpha, 9  
ft\_isascii, 9  
ft\_isdigit, 10  
ft\_isprint, 10  
ft\_isspace, 10  
ft\_itoa, 10  
ft\_lstadd, 11  
ft\_lstappend, 12  
ft\_lstdel, 12  
ft\_lstdelone, 13  
ft\_lstiter, 13  
ft\_lstlast, 14  
ft\_lstmap, 14  
ft\_lstnew, 15  
ft\_max, 16  
ft\_memalloc, 17  
ft\_memccpy, 17  
ft\_memchr, 18  
ft\_memcmp, 18  
ft\_memcpy, 19  
ft\_memdel, 19  
ft\_memmove, 19  
ft\_memset, 20  
ft\_min, 20  
ft\_power, 21  
ft\_putchar, 21  
ft\_putchar\_fd, 22  
ft\_putendl, 22  
ft\_putendl\_fd, 23  
ft\_putnbr, 23  
ft\_putnbr\_fd, 24  
ft\_putstr, 25  
ft\_putstr\_fd, 25  
ft\_puts, 26  
ft\_putstr, 26  
ft\_putstr\_fd, 27  
ft\_sqrt, 27  
ft\_strcapitalize, 28  
ft\_strcat, 29  
ft\_strcchr, 29  
ft\_strchr, 30  
ft\_strclr, 30  
ft\_strcmp, 31  
ft\_strcpy, 31  
ft\_strdel, 31  
ft\_strdup, 32  
ft\_strequ, 33  
ft\_strintab, 33  
ft\_striter, 34  
ft\_striteri, 34  
ft\_strjoin, 35  
ft\_strlast, 36  
ft\_strlcat, 36  
ft\_strlcpy, 37  
ft\_strlen, 38  
ft\_strmap, 38  
ft\_strmapi, 39  
ft\_strncat, 40  
ft\_strncmp, 40  
ft\_strncpy, 41  
ft\_strndup, 41  
ft\_strnequ, 42  
ft\_strnew, 42  
ft\_strnstr, 43

- [ft\\_strchr](#), [44](#)
- [ft\\_strrev](#), [44](#)
- [ft\\_strsplit](#), [45](#)
- [ft\\_strstr](#), [46](#)
- [ft\\_strsub](#), [46](#)
- [ft\\_strtrim](#), [47](#)
- [ft\\_tolower](#), [48](#)
- [ft\\_toupper](#), [48](#)

next

- [s\\_list](#), [5](#)

[s\\_list](#), [4](#)

- [content](#), [5](#)
- [content\\_size](#), [5](#)
- [next](#), [5](#)