libft

Generated by Doxygen 1.8.16

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` 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 `Moulitest`.

1.0.2   Acknowledgements

My thanks go to `yyang42` for making Moulitest 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:

3 File Index

3.1 File List

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

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 34 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 36 of file libft.h.

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 37 of file libft.h.

4.1.2.3 next `s_list::next`

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

Definition at line 38 of file libft.h.

The documentation for this struct was generated from the following file:
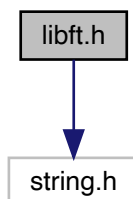
- libft.h

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_atoi (const char ∗str)
- void ft_bzero (void ∗s, size_t n)
- int ft_chrinstr (char c, const char ∗str)
- 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)
- 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_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_putnstr (char ∗s, size_t n)
- void ft_putnstr_fd (char ∗s, size_t n, int fd)
- 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_strcchr (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_strlcat](char *dst, const char *src, size_t dstsize)
- size_t [ft_strlcpy](char *dst, const char *src, size_t maxlen)
- size_t [ft_strlen](const char *s)
- char * [ft_strmap](char const *s, char(*f)(char))
- char * [ft_strmapi](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_atoi() `int ft_atoi (`
`            const char * str )`

Mimic 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 }
```

5.1.1.2 ft_bzero() `void ft_bzero (`
            `void * s,`
            `size_t n )`

Mimic 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 }
```

5.1.1.3 ft_chrinstr() `int ft_chrinstr (`
            `char c,`
            `const char * str )`

Search for a character in a string.

Parameters

c	The character for which to search.
str	The string in which to search.

Returns

1 if the string contains the character and 0 otherwise.

Definition at line 20 of file ft_chrinstr.c.
```
21 {
22     if (!str)
23         return (0);
24     while (*str)
25     {
26         if (*str == c)
27             return (1);
28         str++;
29     }
30     return (0);
31 }
```

5.1.1.4 ft_isalnum() `int ft_isalnum (`
            `int c )`

Mimic 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 )`

Mimic 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 }
```

5.1.1.6 ft_isascii() `int ft_isascii ( int c )`

Mimic 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 )`

Mimic 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 }
```

5.1.1.8 ft_isprint() `int ft_isprint (`
                `int c )`
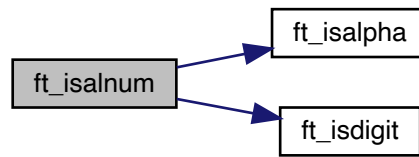
Mimic behaviour of a function of the same name (sans `ft_`) from `libc`.

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 )`

Mimic 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

n	The integer to be transformed into a string.

Returns

The string representing the integer passed as argument.

Definition at line 52 of file ft_itoa.c.
```
53 {
54     char     *str;
55     size_t   len;
56     int      tmp;
57
58     len = ft_len(n);
59     tmp = n;
60     if (n == INT_MIN)
61         return (ft_itoa_intmin(n));
62     if (n < 0)
63     {
64         tmp = -n;
65         len++;
66     }
67     if (!(str = ft_strnew(len)))
68         return (NULL);
69     str[--len] = tmp % 10 + '0';
70     while (tmp /= 10)
71         str[--len] = tmp % 10 + '0';
72     if (n < 0)
73         str[--len] = '-';
74     return (str);
75 }
```

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

alst	The address of a pointer to the first link of a list.
new	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

alst	The address of a pointer to the first link of a list.
new	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

alst	The address of a pointer to the first link of a list that needs to be freed.
del	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

alst	The adress of a pointer to a link that needs to be freed.
del	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

lst	A pointer to the first link of a list.
f	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

lst	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.

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

lst	A pointer's to the first link of a list.
f	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

content	The content to put in the new link.
content_size	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:
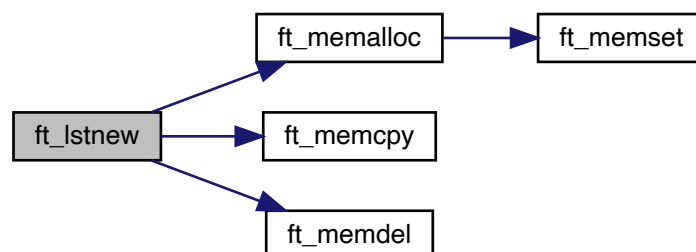
5.1.1.19 ft_memalloc() `void* ft_memalloc (`

`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().

Here is the call graph for this function:

5.1.1.20 ft_memccpy() `void* ft_memccpy (`

`void * dst,`

`const void * src,`

`int c,`

`size_t n )`

Mimic 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 }
```

5.1.1.21 ft_memchr()

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

Mimic 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 }
```

5.1.1.22 ft_memcmp()

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

Mimic 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 }
```

5.1.1.23 ft_memcpy() `void* ft_memcpy (`

   `void * dst,`

   `const void * src,`

   `size_t n )`

Mimic 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  }
```

5.1.1.24 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

ap	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  }
```

5.1.1.25 ft_memmove() `void* ft_memmove (`

   `void * dst,`

   `const void * src,`

   `size_t n )`

Mimic 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 }
```

5.1.1.26 ft_memset() `void* ft_memset (`
            `void * b,`
            `int c,`
            `size_t len )`

Mimic 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 }
```

5.1.1.27 ft_power() `int ft_power (`
            `int num,`
            `unsigned int exponent )`

Raises a number to a given power.

Parameters

num	The base.
exponent	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 }
```

5.1.1.28   ft_putchar() `void ft_putchar (`
`            char c )`

Outputs the character `c` to the standard output.

Parameters

c	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:

5.1.1.29   ft_putchar_fd() `void ft_putchar_fd (`
`            char c,`
`            int fd )`

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

Parameters

c	The character to output.
fd	The file descriptor.

Definition at line 22 of file ft_putchar_fd.c.

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

5.1.1.30   ft_putendl() `void ft_putendl (`
`            char const * s )`

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

Parameters

s	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:
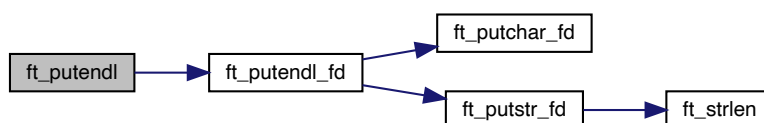
5.1.1.31 ft_putendl_fd() `void ft_putendl_fd (`
`          char const * s,`
`          int fd )`

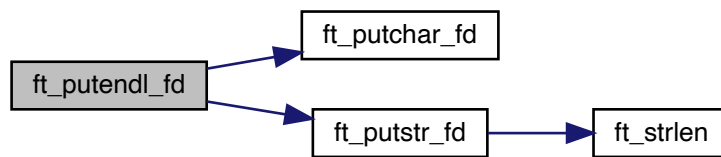Outputs the string `s` to the file descriptor `fd` followed by a `'\n'`.

Parameters

s	The string to output.
fd	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().

Here is the call graph for this function:

5.1.1.32 ft_putnbr() `void ft_putnbr (`
       `int n )`

Outputs the integer `n` to the standard output.
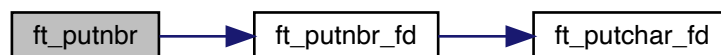
Parameters

n	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:

5.1.1.33 ft_putnbr_fd() `void ft_putnbr_fd (`
       `int n,`
       `int fd )`

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

Parameters

n	The integer to print.
fd	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().

Here is the call graph for this function:

5.1.1.34 ft_putnstr() `void ft_putnstr (`
            `char * s,`
            `size_t n )`

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

Parameters

s	The string, the characters of which to output.
n	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 }
```

5.1.1.35 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

s	The string, the characters of which to output.
n	The number of characters to output.
fd	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 }
```

5.1.1.36 ft_putstr() `void ft_putstr (`
            `char const * s )`

Outputs the string `s` to the standard output.

Parameters

s	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:
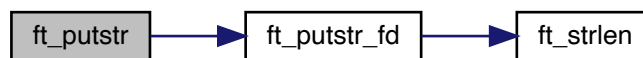
5.1.1.37   ft_putstr_fd() `void ft_putstr_fd (`
         `char const * s,`
         `int fd )`

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

Parameters

s	The string to output.
fd	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().

Here is the call graph for this function:

5.1.1.38   ft_sqrt() `int ft_sqrt (`
         `int num )`

Computes an integer square root of a given number.

Parameters

num	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 }
```

5.1.1.39 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

s1	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:
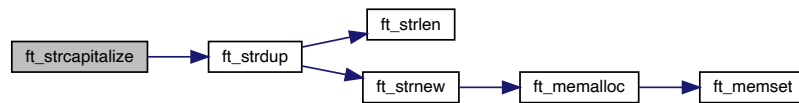
5.1.1.40 ft_strcat() `char* ft_strcat (`
`char * s1,`
`const char * s2 )`

Mimic 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().

Here is the call graph for this function:

5.1.1.41 ft_strcchr() `int ft_strcchr (`
`char const * str,`
`char c )`

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

Parameters

str	The string in which to search.
c	The character for which to search.

Returns

The number of occurences.

Definition at line 22 of file ft_strcchr.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 }
```

5.1.1.42 ft_strchr() `char* ft_strchr (`
`const char * s,`
`int c )`

Mimic 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 }
```

5.1.1.43 ft_strclr() `void ft_strclr (`
`char * s )`

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

Parameters

s	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:
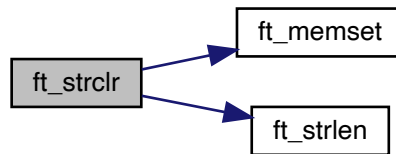
5.1.1.44 ft_strcmp() int ft_strcmp (
                const char * *s1,*
                const char * *s2*)

Mimic 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  }
```

5.1.1.45 ft_strcpy() char* ft_strcpy (
   char * *dst,*
                const char * *src*)

Mimic 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  }
```

5.1.1.46 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

as	The string's address that needs to be freed and its pointer set to `NULL`.

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:

5.1.1.47 ft_strdup() `char* ft_strdup (`
            `const char * s1 )`

Mimic 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().

Here is the call graph for this function:
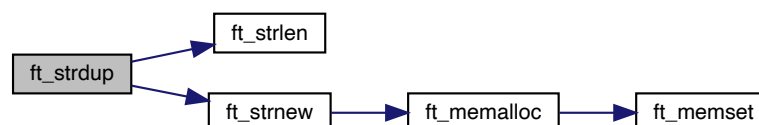
5.1.1.48 ft_strequ() `int ft_strequ (`
`            const char * s1,`
`            const char * s2 )`

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

s1	The first string to be compared.
s2	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 }
```

5.1.1.49 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

str	The string to search for.
tab	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:

5.1.1.50 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

s	The string to iterate.
f	The function to apply to each character of `s`.

Definition at line 22 of file ft_striter.c.

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

5.1.1.51 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

s	The string to iterate.
f	The function to apply to each character of `s` 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 }
```

5.1.1.52 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

s1	The prefix string.
s2	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:
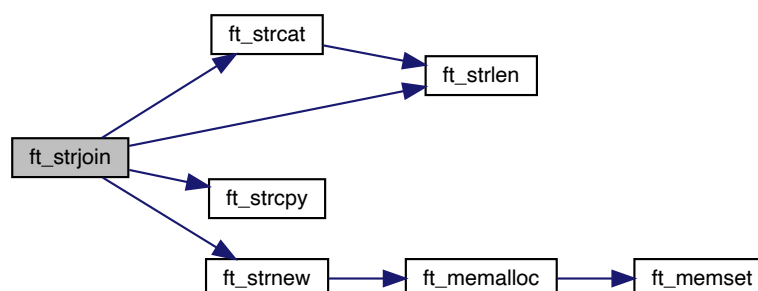
5.1.1.53   **ft_strlast()** `char ft_strlast (`
            `char const ∗ str )`

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

Parameters

str	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 }
```

5.1.1.54   **ft_strlcat()** `size_t ft_strlcat (`
            `char ∗ dst,`
            `const char ∗ src,`
            `size_t dstsize )`

Mimic 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:

5.1.1.55 ft_strlcpy() `size_t ft_strlcpy (`
`           char * dst,`
`           const char * src,`
`           size_t maxlen )`

Mimic 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:
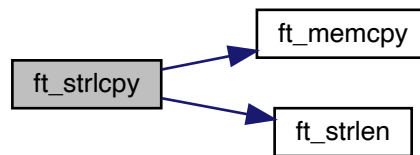
5.1.1.56 ft_strlen() `size_t ft_strlen (`
`           const char * s )`

Mimic 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 }
```

5.1.1.57 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

s	The string to map.
f	The function to apply to each character of `s`.

Returns

The "fresh" string created from the successive applications of `f`.
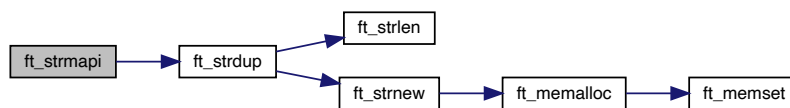
Definition at line 24 of file ft_strmap.c.

```
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:

5.1.1.58 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

s	The string to map.
f	The function to apply to each character and its index of `s`.

Returns

The "fresh" string created from the successive applications of f.

Definition at line 25 of file ft_strmapi.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:

5.1.1.59 ft_strncat() `char* ft_strncat (`
`            char * s1,`
`            const char * s2,`
`            size_t n )`

Mimic 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:

5.1.1.60 ft_strncmp() `int ft_strncmp (`
          `const char * s1,`
          `const char * s2,`
          `size_t n )`

Mimic 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 }
```

5.1.1.61 ft_strncpy() `char* ft_strncpy (`
          `char * dst,`
          `const char * src,`
          `size_t len )`

Mimic 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 }
```

5.1.1.62 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

s1	String to be copied from.
len	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 }
```

5.1.1.63 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

s1	The first string to be compared.
s2	The second string to be compared.
n	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 }
```

5.1.1.64 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

size	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().

Here is the call graph for this function:
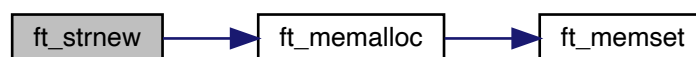
5.1.1.65 ft_strnstr() `char* ft_strnstr (`

```
        const char * haystack,
        const char * needle,
        size_t len )
```

Mimic 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 }
```

5.1.1.66 ft_strrchr() `char* ft_strrchr (`

```
        const char * s,
        int c )
```

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

Definition at line 19 of file ft_strrchr.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 }
```

5.1.1.67 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:
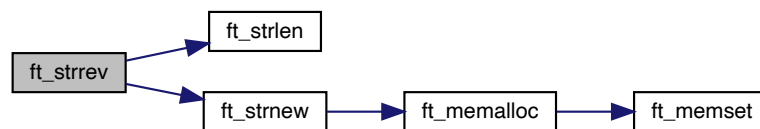
5.1.1.68   ft_strsplit() `char** ft_strsplit (`
            `char const * s,`
            `char delim )`

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

s	The string to split.
delim	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:
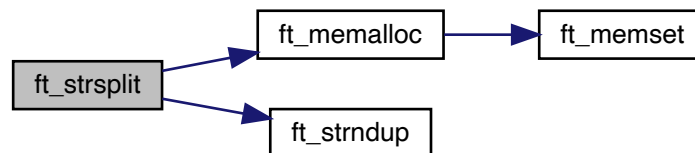
5.1.1.69 ft_strstr() char* ft_strstr (
            const char * *haystack,*
            const char * *needle*)

Mimic 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 }
```

5.1.1.70 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 refering to a valid substring, the behavior is undefined. If the allocation fails, the function returns `NULL`.

Parameters

s	The string from which create the substring.
start	The start index of the substring.
len	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().

Here is the call graph for this function:
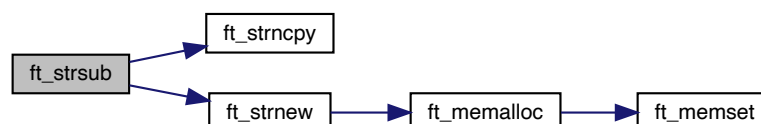
5.1.1.71 ft_strtrim() `char* ft_strtrim (`
`char const * s )`

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:
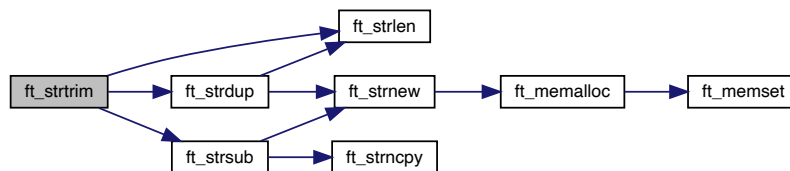
5.1.1.72 ft_tolower() `int ft_tolower (`
`int c )`

Mimic 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 }
```

5.1.1.73 ft_toupper() `int ft_toupper (`
            `int c )`

Mimic 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