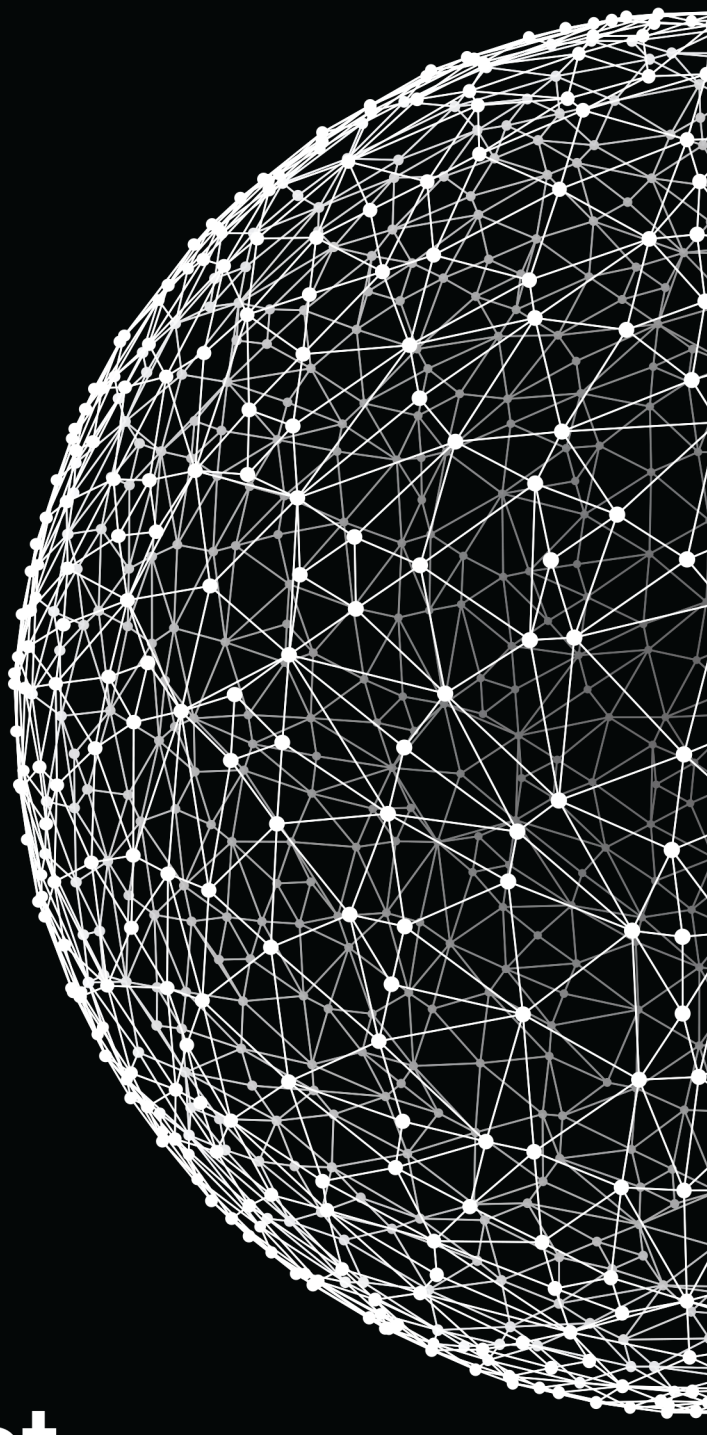
Libmx

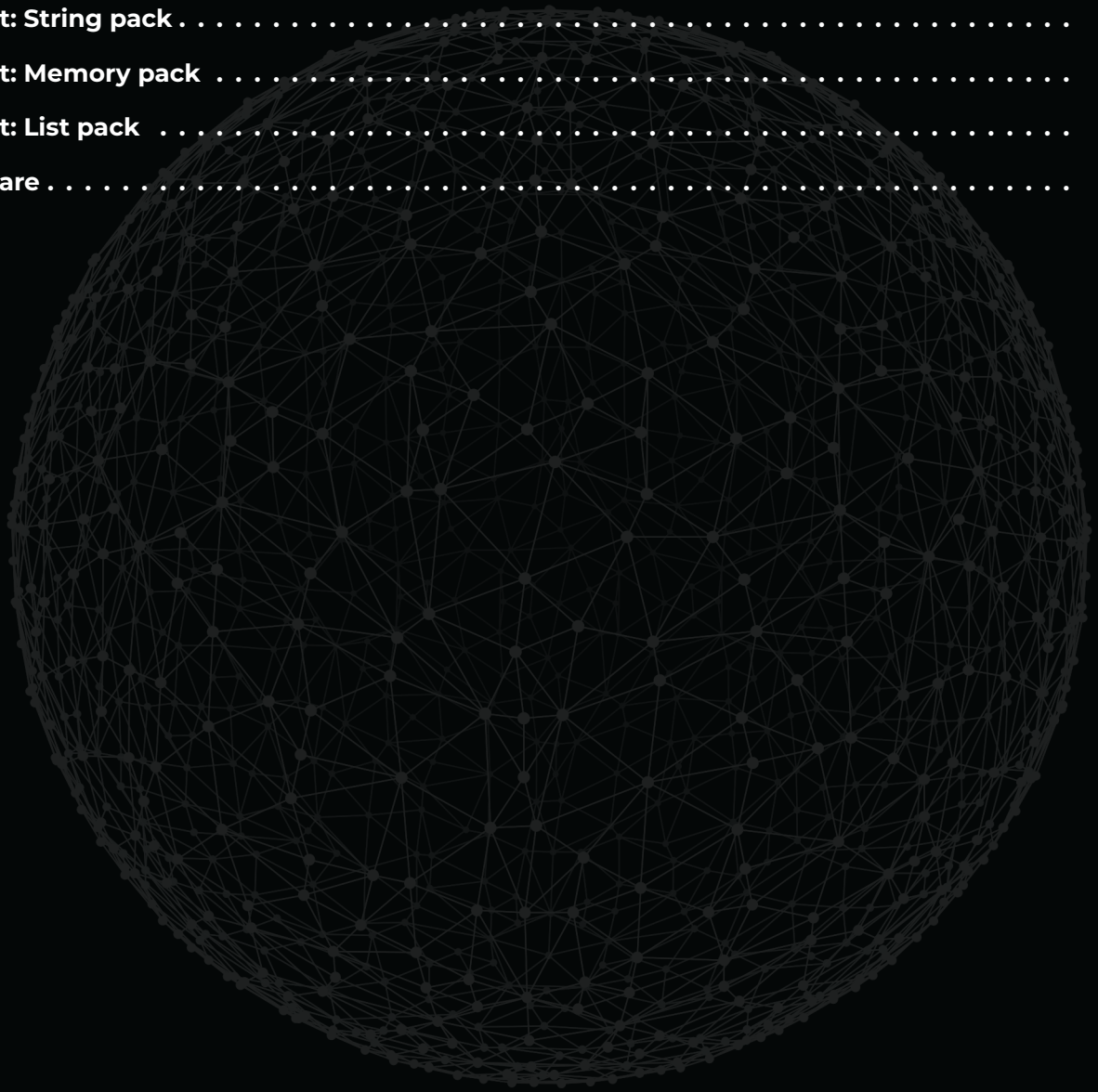
Track C

November 19, 2020



ucode connect

Contents



ucode connect

Engage

DESCRIPTION

Hey there.

The next challenge in Track C is to create your own library of functions.
The implementation of this challenge will help simplify your programming life and save a lot of time in future development. By creating various functions, you can understand even more deeply how they work, why and how they are used, and understand the algorithms of their work.
Of course, you can use the functions from the created library in the next Track C challenges.
In addition, you have a great opportunity to expand your library with even more useful functions and make it unique.

Good luck, The Chosen One!

BIG IDEA

Stay DRY. Don't Repeat Yourself.

ESSENTIAL QUESTION

How can I reuse my code, modules, programs, etc.?

CHALLENGE

Create your own library.

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What are `libraries` in programming?
- What is the difference between a `framework` and a `library`?
- Why is it useful to create your own library?
- Have you ever developed a library before?
- What is the difference between `static` and `dynamic` libraries?
- What do you know about dynamic libraries?
- What are `static variables`? What is their life span?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Define the terms of the challenge (library, function, solution, product, etc.).
- Ask students who have already begun this challenge about what is best to start with and what problems they have encountered.
- Make a plan where to start. Read the tasks, examine all the functions that you must create.
- Think about the extra functions you would like to have in your library.
- Carefully read the instructions, as well as the `man` for the functions that have similar behavior to the standard libc functions.
- Ask other students to test your solutions. Help each other find mistakes.
- Think about how you can improve your functions.
- Clone your git repository that is issued on the challenge page.
- Push your solutions.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Be attentive to all statements of the story.
- The challenge must have the following structure:
  - `src` directory contains files with extension `.c`
  - `obj` directory contains files with extension `.o` (you must not push this directory in your repository, only `Makefile` creates it during compilation)

  - `inc` directory contains header `libmx.h`
  - `Makefile` that compiles and builds `libmx.a`

- Complete the challenge according to the rules specified in the `Auditor`.
- Submit your files using the layout described in the story. Only useful files allowed, garbage shall not pass!
- Compile C-files with clang compiler and use these flags: `clang -std=c11 -Wall -Wextra -Werror -Wpedantic`.
- You are allowed to use such functions: `malloc, malloc_size, free, open, read, write, close, exit`.
- Pay attention to what is allowed. Use of forbidden stuff is considered a cheat and your challenge will be failed.
- A memory that is no longer needed must be freed, otherwise, the function is considered incomplete.
- All functions that are given in all parts of the story must be done in separate files.
- It is recommended to reuse already written functions for writing new ones.
- You can add some custom functions if you need it.
- The solution will be checked and graded by students like you. `Peer-to-Peer learning`.
- Also, the challenge will pass automatic evaluation which is called `Oracle`.
- If you have any questions or don't understand something, ask other students or just Google it.

Act: Utils pack

In this pack, you must create util functions that make your work easier. You have already developed some of these features during the Refresh Marathon C. Find the prototypes for every function below.

NAME

Print character

DESCRIPTION

Create a function that outputs a single character to the standard output.

SYNOPSIS

```
void mx_printchar(char c);
```

NAME

Print multibyte characters

DESCRIPTION

Create a function that outputs ASCII and multibyte characters to the standard output.

SYNOPSIS

```
void mx_print_unicode(wchar_t c);
```

NAME

Print string

DESCRIPTION

Create a function that outputs a string of characters to the standard output.

SYNOPSIS

```
void mx_printstr(const char *s);
```

NAME

Print array of strings

DESCRIPTION

Create a function that outputs:

- an array of strings `arr` to the standard output with a delimiter `delim` between the elements of an array
- nothing if `arr` or `delim` do not exist
- a newline at the end of the output

`arr` must be `NULL`-terminated, in other cases the behavior is undefined.

SYNOPSIS

```c
void mx_print_strarr(char **arr, const char *delim);
```

NAME

Print integer

DESCRIPTION

Create a function that outputs integer values to the standard output.

SYNOPSIS

```c
void mx_printint(int n);
```

EXAMPLE

```c
mx_printint(25); //prints 25
mx_printint(2147483647); //prints 2147483647
```

NAME

Exponentiation

DESCRIPTION

Create a function that computes `n` raised to the power of zero or a positive integer `pow`.

RETURN

Returns the result of `n` to the power of `pow`.

SYNOPSIS

```
double mx_pow(double n, unsigned int pow);
```

EXAMPLE

```
mx_pow(3, 3); //returns 27
mx_pow(2.5, 3); //returns 15.625
mx_pow(2, 0); //returns 1
```

NAME

Square root

DESCRIPTION

Create a function that computes the non-negative square root of `x`.
The function must compute square root in less than 2 seconds.

RETURN

Returns the square root of the number `x` if it is natural, and `0` otherwise.

SYNOPSIS

```
int mx_sqrt(int x);
```

EXAMPLE

```
mx_sqrt(3); //returns 0
mx_sqrt(4); //returns 2
```

NAME

Decimal to hex

DESCRIPTION

Create a function that converts an `unsigned long` number into a hexadecimal string.

RETURN

Returns the number converted to a hexadecimal string.

SYNOPSIS

```
char *mx_nbr_to_hex(unsigned long nbr);
```

EXAMPLE

```
mx_nbr_to_hex(52); //returns "34"
mx_nbr_to_hex(1000); //returns "3e8"
```

NAME

Hex to decimal

DESCRIPTION

Create a function that converts a hexadecimal string into an `unsigned long` number.

RETURN

Returns the `unsigned long` number.

SYNOPSIS

```
unsigned long mx_hex_to_nbr(const char *hex);
```

EXAMPLE

```
mx_hex_to_nbr("C4"); //returns 196
mx_hex_to_nbr("FADE"); //returns 64222
mx_hex_to_nbr("fffffffffffff"); //returns 281474976710655
mx_hex_to_nbr(NULL); //returns 0
```

NAME

Integer to ASCII

DESCRIPTION

Create a function that takes an integer and converts it to a string.

RETURN

Returns the number as a `NULL`-terminated **string.**

SYNOPSIS

```
char *mx_itoa(int number);
```

NAME

For each

DESCRIPTION

Create a function that applies the function `f` for each element of the array `arr` given `size`.

SYNOPSIS

```
void mx_foreach(int *arr, int size, void (*f)(int));
```

NAME

Binary search

DESCRIPTION

Create a function that:

- searches the string `s` in the array `arr` with the given `size` of array
- uses the binary search algorithm assuming that the input array has already been sorted in a lexicographical order

RETURN

- returns the `index` of the found string in the array
- returns `-1` in case of errors or if the string has not been found
- assigns the number of required iterations to the `count` pointer

SYNOPSIS

```
int mx_binary_search(char **arr, int size, const char *s, int *count);
```

EXAMPLE

```
arr = {"222", "Abcd", "aBc", "ab", "az", "z"};
count = 0;
mx_binary_search(arr, 6, "ab", &count); //returns 3 and count = 3
count = 0;
mx_binary_search(arr, 6, "aBc", &count); //returns 2 and count = 1
count = 0;
mx_binary_search(arr, 6, "aBz", &count); //returns -1 and count = 0
```

NAME

Bubble sort

DESCRIPTION

Create a function that:

- sorts an array of strings in place in lexicographical order
- uses the `bubble sort` algorithm

RETURN

Returns the number of swap operations.

SYNOPSIS

```
int mx_bubble_sort(char **arr, int size);
```

EXAMPLE

```
arr = {"abc", "xyz", "ghi", "def"};
mx_bubble_sort(arr, 4); //returns 3

arr = {"abc", "acb", "a"};
mx_bubble_sort(arr, 3); //returns 2
```

NAME

Quick sort

DESCRIPTION

Create a function that:

- sorts an array of strings by their lengths in ascending order
- uses the `quick sort` algorithm. Select the middle element of the array as the center

You must not check the validity of `left` and `right`.

RETURN

- returns the number of swaps
- returns `-1` if `arr` does not exist

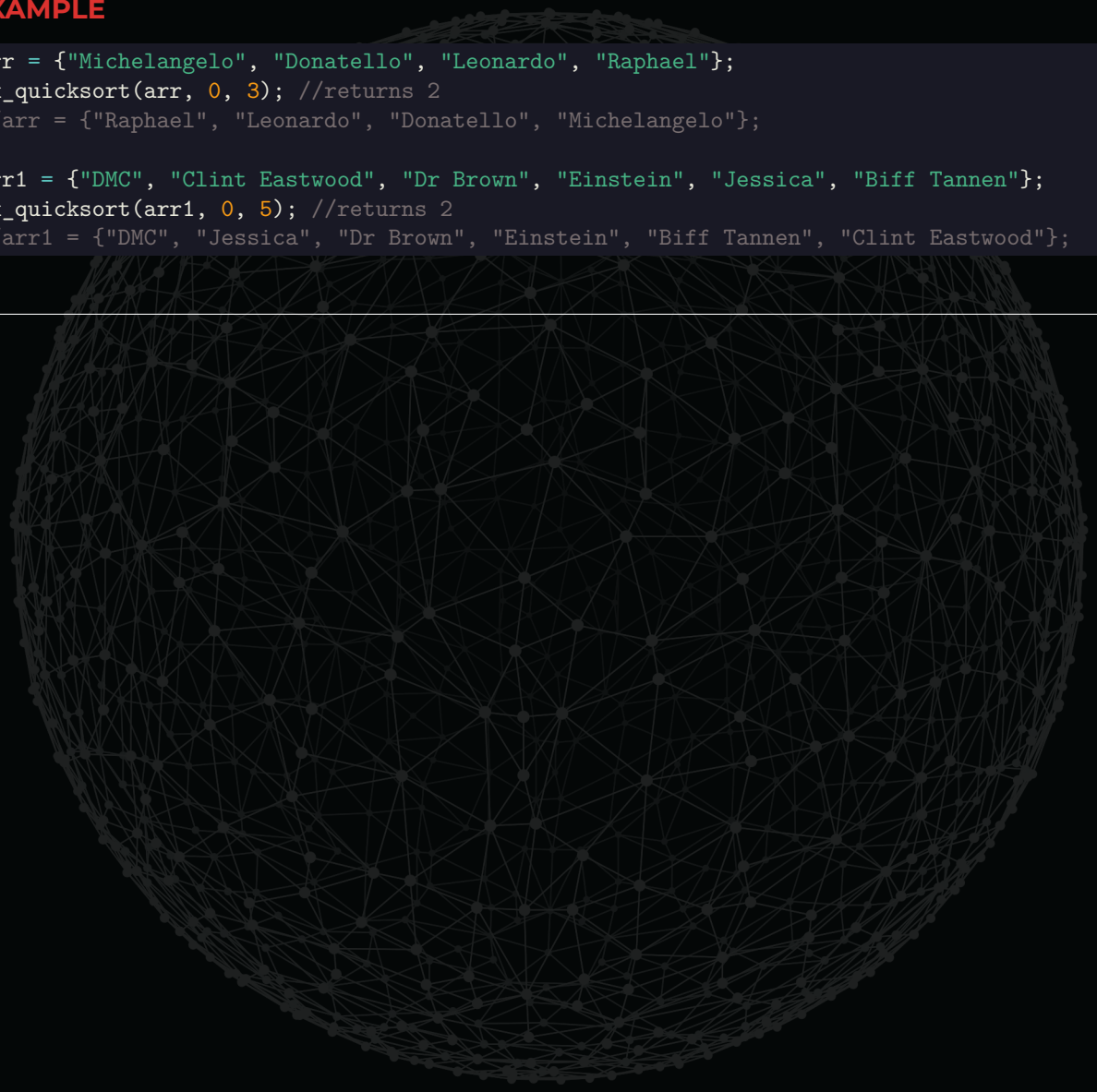
SYNOPSIS

```c
int mx_quicksort(char **arr, int left, int right);
```

EXAMPLE

```c
arr = {"Michelangelo", "Donatello", "Leonardo", "Raphael"};
mx_quicksort(arr, 0, 3); //returns 2
//arr = {"Raphael", "Leonardo", "Donatello", "Michelangelo"};

arr1 = {"DMC", "Clint Eastwood", "Dr Brown", "Einstein", "Jessica", "Biff Tannen"};
mx_quicksort(arr1, 0, 5); //returns 2
//arr1 = {"DMC", "Jessica", "Dr Brown", "Einstein", "Biff Tannen", "Clint Eastwood"};
```



Act: String pack

In this pack, you must create functions to operate with strings. You already developed some of these functions during the Refresh Marathon C. Find prototypes for each function below.

NAME

String length

DESCRIPTION

Create a function that has the same behavior as the corresponding standard libc function `strlen`.

SYNOPSIS

```
int mx_strlen(const char *s);
```

NAME

Swap characters

DESCRIPTION

Create a function that swaps the characters of a string using pointers. Do nothing if `s1` or `s2` does not exist.

SYNOPSIS

```
void mx_swap_char(char *s1, char *s2);
```

EXAMPLE

```
str = "ONE";
mx_swap_char(&str[0], &str[1]); //'str' now is "NOE"
mx_swap_char(&str[1], &str[2]); //'str' now is "NEO"
```

NAME

Reverse string

DESCRIPTION

Create a function that reverses a string using pointers. Do nothing if a string does not exist.

SYNOPSIS

```
void mx_str_reverse(char *s);
```

EXAMPLE

```
str = "game over";
mx_str_reverse(str); //'str' now is "revo emag"
```

NAME

Delete string

DESCRIPTION

Create a function that:

- takes a pointer to a string
- frees string memory with `free`
- sets the string to `NULL`

SYNOPSIS

```
void mx_strdel(char **str);
```

NAME

Delete array of strings

DESCRIPTION

Create a function that:

- takes a pointer to a `NULL`-terminated array of strings
- deletes the contents of the array
- frees array memory with `free`
- sets a pointer to `NULL`

SYNOPSIS

```
void mx_del_strarr(char ***arr);
```

NAME

Get character index

DESCRIPTION

Create a function that finds the index of the first occurrence of the character `c` in a string `str`. A string is a sequence of characters, excluding `NULL` in the end.

RETURN

- returns the index of the first occurrence
- returns `-1` if no occurrence is found
- returns `-2` if the string does not exist

SYNOPSIS

```c
int mx_get_char_index(const char *str, char c);
```

NAME

Duplicate string

DESCRIPTION

Create a function that has the same behavior as the standard libc function `strdup`.

SYNOPSIS

```c
char *mx_strdup(const char *s1);
```

NAME

Duplicate part of string

DESCRIPTION

Create a function that has the same behavior as the standard libc function `strndup`.

SYNOPSIS

```c
char *mx_strndup(const char *s1, size_t n);
```

NAME

Copy string

DESCRIPTION

Create a function that has the same behavior as the standard libc function `strcpy`.

SYNOPSIS

```
char *mx_strcpy(char *dst, const char *src);
```

NAME

Copy them all

DESCRIPTION

Create a function that has the same behavior as the standard libc function `strncpy`.

SYNOPSIS

```
char *mx_strncpy(char *dst, const char *src, int len);
```

NAME

Compare strings

DESCRIPTION

Create a function that has the same behavior as the standard libc function `strcmp`.

SYNOPSIS

```
int mx_strcmp(const char *s1, const char *s2);
```

NAME

Concatenate strings

DESCRIPTION

Create a function that has the same behavior as the standard libc function `strcat`.

SYNOPSIS

```
char *mx_strcat(char *restrict s1, const char *restrict s2);
```

NAME

Locate a substring

DESCRIPTION

Create a function that has the same behavior as the standard libc function `strstr`.

SYNOPSIS

```
char *mx_strstr(const char *haystack, const char *needle);
```

NAME

Get substring index

DESCRIPTION

Create a function that finds the index of a substring.

RETURN

- returns the index of the first character of `sub` in `str`
- returns `-1` if `sub` is not found in `str`
- returns `-2` if `str` or `sub` does not exist

SYNOPSIS

```
int mx_get_substr_index(const char *str, const char *sub);
```

EXAMPLE

```
mx_get_substr_index("McDonalds", "Don"); //returns 2
mx_get_substr_index("McDonalds Donuts", "on"); //returns 3
mx_get_substr_index("McDonalds", "Donatello"); //returns -1
mx_get_substr_index("McDonalds", NULL); //returns -2
mx_get_substr_index(NULL, "Don"); //returns -2
```

NAME

Count substrings

DESCRIPTION

Create a function that counts the substrings `sub` in the string `str`.

RETURN

- returns the count of `sub` in `str`
- returns `0` if `sub` is an empty string
- returns `-1` if `str` and / or `sub` do not exist

SYNOPSIS

```c
int mx_count_substr(const char *str, const char *sub);
```

EXAMPLE

```c
str = "yo, yo, yo Neo";
sub = "yo";
mx_count_substr(str, sub); //returns 3
mx_count_substr(str, NULL); //returns -1
mx_count_substr(NULL, sub); //returns -1
```

NAME

Count words

DESCRIPTION

Create a function that counts words in a string.
Word is a sequence of characters separated by a delimiter.

RETURN

Returns the number of words in the string.

SYNOPSIS

```c
int mx_count_words(const char *str, char c);
```

EXAMPLE

```c
str = "  follow  *   the  white rabbit ";
mx_count_words(str, '*'); //returns 2
mx_count_words(str, ' '); //returns 5
mx_count_words(NULL, ' '); //returns -1
```

NAME

New string

DESCRIPTION

Create a function that:

- allocates memory for a string of a specific `size` and one additional byte for the terminating `'\0'`
- initializes each character with `'\0'`

RETURN

- returns the string of a specific `size` and terminated by `'\0'`
- returns `NULL` if creation fails

SYNOPSIS

```
char *mx_strnew(const int size);
```

NAME

Trim string

DESCRIPTION

Create a function that:

- takes a string, and creates a new one from it without whitespace characters at the beginning and the end of the string
- frees all unused memory

RETURN

- returns a new trimmed string
- returns `NULL` if the string `str` does not exist or string trim fails

SYNOPSIS

```
char *mx_strtrim(const char *str);
```

EXAMPLE

```
name = "\f  My name... is Neo  \t\n ";
mx_strtrim(name); //returns "My name... is Neo"
```

NAME

Clean string

DESCRIPTION

Create a function that:

- takes a string, and creates a new one from it without whitespace characters in the beginning and/or at the end of the string
- separates words in the new string with exactly one space character
- frees all unused memory

A word is a sequence of characters separated by whitespaces.

RETURN

- returns a new created string
- returns `NULL` if the string `str` does not exist or string creation fails

SYNOPSIS

```
char *mx_del_extra_spaces(const char *str);
```

EXAMPLE

```
name = "\f  My name...     is  \r Neo  \t\n ";
mx_del_extra_spaces(name); //returns "My name... is Neo"
```

NAME

Split string

DESCRIPTION

Create a function that:

- converts a string `s` to a `NULL`-terminated array of words
- frees all unused memory

A word is a sequence of characters separated by the character `c` as a delimiter.

RETURN

- returns the `NULL`-terminated array of strings
- returns `NULL` if the string `s` does not exist or conversion fails

SYNOPSIS

```
char **mx_strsplit(const char *s, char c);
```

EXAMPLE

```
s = "**Good bye,**Mr.*Anderson.****";
arr = mx_strsplit(s, '*');  // arr = ["Good bye,", "Mr.", "Anderson."]
s = "     Knock, knock,    Neo.    ";
arr = mx_strsplit(s, ' ');  // arr = ["Knock,", "knock,", "Neo."]
```

NAME

Join strings

DESCRIPTION

Create a function that:

- concatenates strings `s1` and `s2` into a new string
- terminates the new string with `'\0'`

RETURN

- returns the string as a result of concatenation `s1` and `s2`
- returns the new copy of `non-NULL` parameter if one and only one of the parameters is `NULL`
- returns `NULL` if the concatenation fails

SYNOPSIS

```
char *mx_strjoin(const char *s1, const char *s2);
```

EXAMPLE

```
str1 = "this";
str2 = "dodge ";
str3 = NULL;
mx_strjoin(str2, str1); //returns "dodge this"
mx_strjoin(str1, str3); //returns "this"
mx_strjoin(str3, str3); //returns NULL
```

NAME

File to string

DESCRIPTION

Create a function that:

- takes a filename as a parameter
- reads data from the file into a string

RETURN

- returns a `NULL`-terminated string
- returns `NULL` in case of any errors

SYNOPSIS

```
char *mx_file_to_str(const char *file);
```

NAME

Replace substrings

DESCRIPTION

Create a function that replaces all occurrences of `sub` in `str` with `replace`.

RETURN

- returns a new string where substrings are replaced
- returns `NULL` if `sub` or `str` or `replace` does not exist

SYNOPSIS

```
char *mx_replace_substr(const char *str, const char *sub, const char *replace);
```

EXAMPLE

```
mx_replace_substr("McDonalds", "alds", "uts"); //returns "McDonuts"
mx_replace_substr("Ururu turu", "ru", "ta"); //returns "Utata tuta"
```

NAME

Read line a.k.a. Mr. Big

DESCRIPTION

Create a function that reads the line from the given `fd` into the `lineptr` until it:

- reaches a `delim` character. The delimiter must not be returned with `lineptr`
- reaches the End Of File (EOF)

A line is a sequence of characters before a delimiter.

The function:

- works correctly with any file descriptor `fd`
- works correctly with any positive integer `buf_size`. `buf_size` must be passed to the function `read` as a parameter `nbytes`
- can read all data from the given `fd` until the EOF, one line per call
- may contain a single static variable while global variables are still forbidden
- may have undefined behavior while reading from a binary file

RETURN

- returns the number of bytes that are written into `lineptr`
- returns `-1` if EOF is reached and there is nothing to write in `lineptr`
- returns `-2` in case of errors or `fd` is invalid

SYNOPSIS

```c
int mx_read_line(char **lineptr, size_t buf_size, char delim, const int fd);
```

EXAMPLE

```c
/* lets imagine that there is a file 'fragment' and it contains:
FADE IN:

ON COMPUTER SCREEN

so close it has no boundaries.

A blinking cursor...
*/
fd = open("fragment", O_RDONLY);

res = mx_read_line(&str, 7, '\n', fd); //res = 8, str = "FADE IN:"
res = mx_read_line(&str, 35, 'a', fd); //res = 34, str = "
//ON COMPUTER SCREEN
//
//so close it h"
```

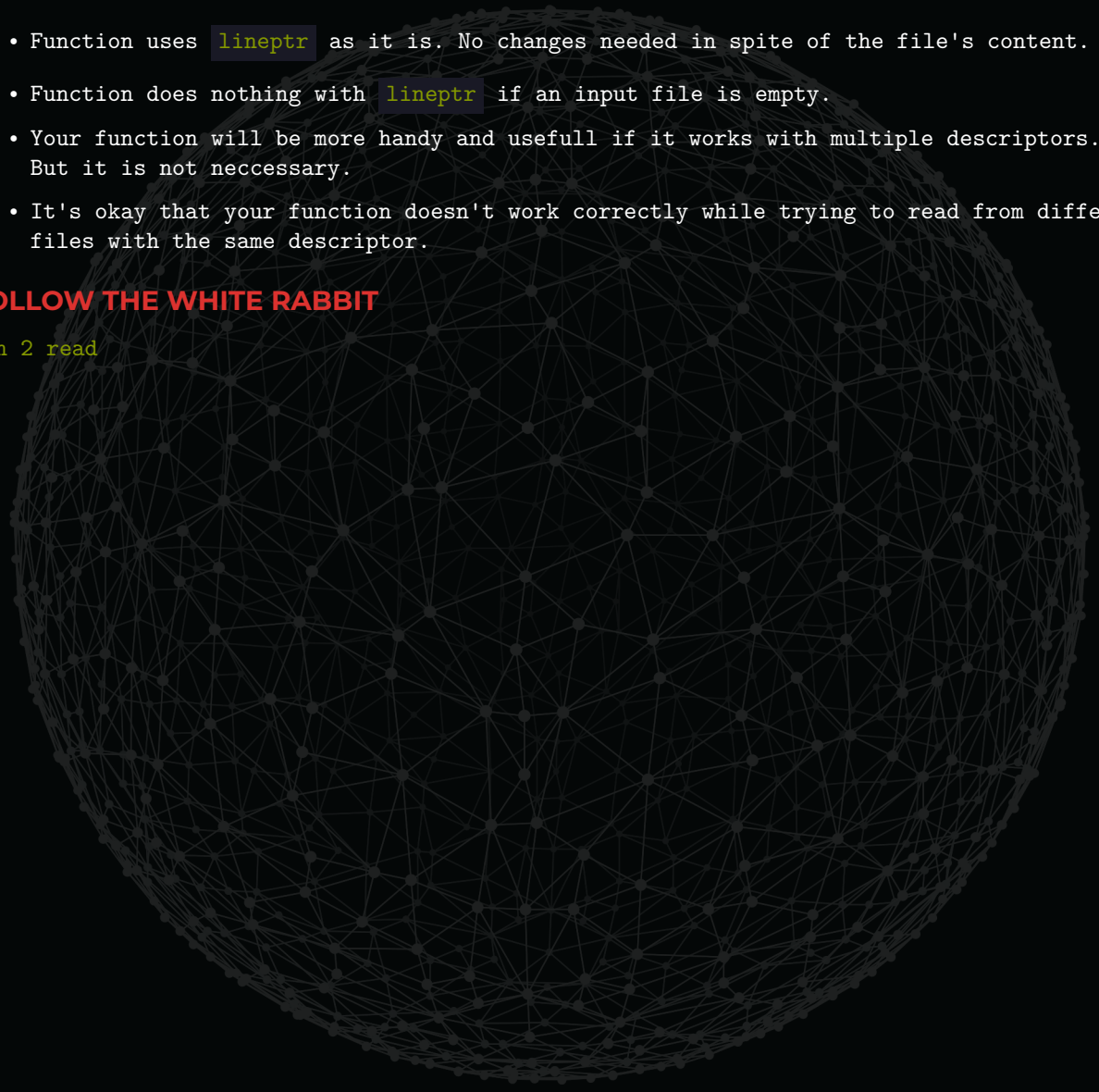
```
res = mx_read_line(&str, 1, '.', fd); //res = 15, str = "s no boundaries"
res = mx_read_line(&str, 10, '\n', fd); //res = 0, str = ""
```

TIPS

- Function uses `lineptr` as it is. No changes needed in spite of the file's content.
- Function does nothing with `lineptr` if an input file is empty.
- Your function will be more handy and usefull if it works with multiple descriptors. But it is not neccessary.
- It's okay that your function doesn't work correctly while trying to read from different files with the same descriptor.

FOLLOW THE WHITE RABBIT

`man 2 read`



Act: Memory pack

Correct work with memory is an integral and important part of the interaction of your program with the computer. To do this, we need the following functions.

NAME

Fill memory

DESCRIPTION

Create a function that has the same behavior as the standard libc function `memset`.

SYNOPSIS

```
void *mx_memset(void *b, int c, size_t len);
```

NAME

Copy memory

DESCRIPTION

Create a function that has the same behavior as the standard libc function `memcpy`.

SYNOPSIS

```
void *mx_memcpy(void *restrict dst, const void *restrict src, size_t n);
```

NAME

Copy memory to ...

DESCRIPTION

Create a function that has the same behavior as the standard stdlib function `memccpy`.

SYNOPSIS

```
void *mx_memccpy(void *restrict dst, const void *restrict src,
                 int c, size_t n);
```

NAME

Compare memory

DESCRIPTION

Create a function that has the same behavior as the standard stdlib function `memcmp`.

SYNOPSIS

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

NAME

Locate byte from start

DESCRIPTION

Create a function that has the same behavior as the standard stdlib function `memchr`.

SYNOPSIS

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

NAME

Locate byte from end

DESCRIPTION

Create a function `mx_memrchr`, which is similar to the function `mx_memchr`, except that it searches in the opposite direction from the end of the bytes `n` points to `s` instead of directly from the beginning.

SYNOPSIS

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

EXAMPLE

```
mx_memrchr("Trinity", 'i', 7); //returns "ity"
mx_memrchr("Trinity", 'M', 7); //returns NULL
```

NAME

Locate block of bytes

DESCRIPTION

Create a function that has the same behavior as the standard libc function `memmem`.

SYNOPSIS

```
void *mx_memmem(const void *big, size_t big_len, const void *little,
                size_t little_len);
```

NAME

Non-overlapping memory copy

DESCRIPTION

Create a function that has the same behavior as the standard libc function `memmove`.

SYNOPSIS

```
void *mx_memmove(void *dst, const void *src, size_t len);
```

NAME

Reallocate memory

DESCRIPTION

Create a function that has the same behavior as the standard stdlib function `realloc`.

SYNOPSIS

```
void *mx_realloc(void *ptr, size_t size);
```

Act: List pack

So now we come to an important and convenient data structure. You will use it in the future. You can find prototypes for every function in the list of tasks and `s_list` structure below. Your library header must contain the structure `s_list` and the required includes and prototypes to compile the functions successfully.

```
typedef struct s_list {
    void *data;
    struct s_list *next;
}              t_list;
```

NAME

Create node

DESCRIPTION

Create a function that creates a new node of a linked list `t_list`. The function assigns a parameter `data` to the list variable `data` and assigns `next` to `NULL`.

SYNOPSIS

```
t_list  *mx_create_node(void *data);
```

NAME

Push front

DESCRIPTION

Create a function that inserts a new node of `t_list` type with the given parameter `data` at the beginning of the linked list.

SYNOPSIS

```
void mx_push_front(t_list **list, void *data);
```

NAME

Push back

DESCRIPTION

Create a function that inserts a node of `t_list` type with the given parameter `data` at the end of the linked list.

SYNOPSIS

```
void mx_push_back(t_list **list, void *data);
```

NAME

Pop front

DESCRIPTION

Create a function that removes the first node of the linked list and frees the memory allocated for the node.

SYNOPSIS

```
void mx_pop_front(t_list **head);
```

NAME

Pop back

DESCRIPTION

Create a function that removes the last node of the linked list and frees the memory allocated for the node.

SYNOPSIS

```
void mx_pop_back(t_list **head);
```

NAME

Size of list

DESCRIPTION

Create a function that calculates the number of nodes in a linked list.

RETURN

Returns the amount of nodes in the linked list.

SYNOPSIS

```
int mx_list_size(t_list *list);
```

NAME

Sort list

DESCRIPTION

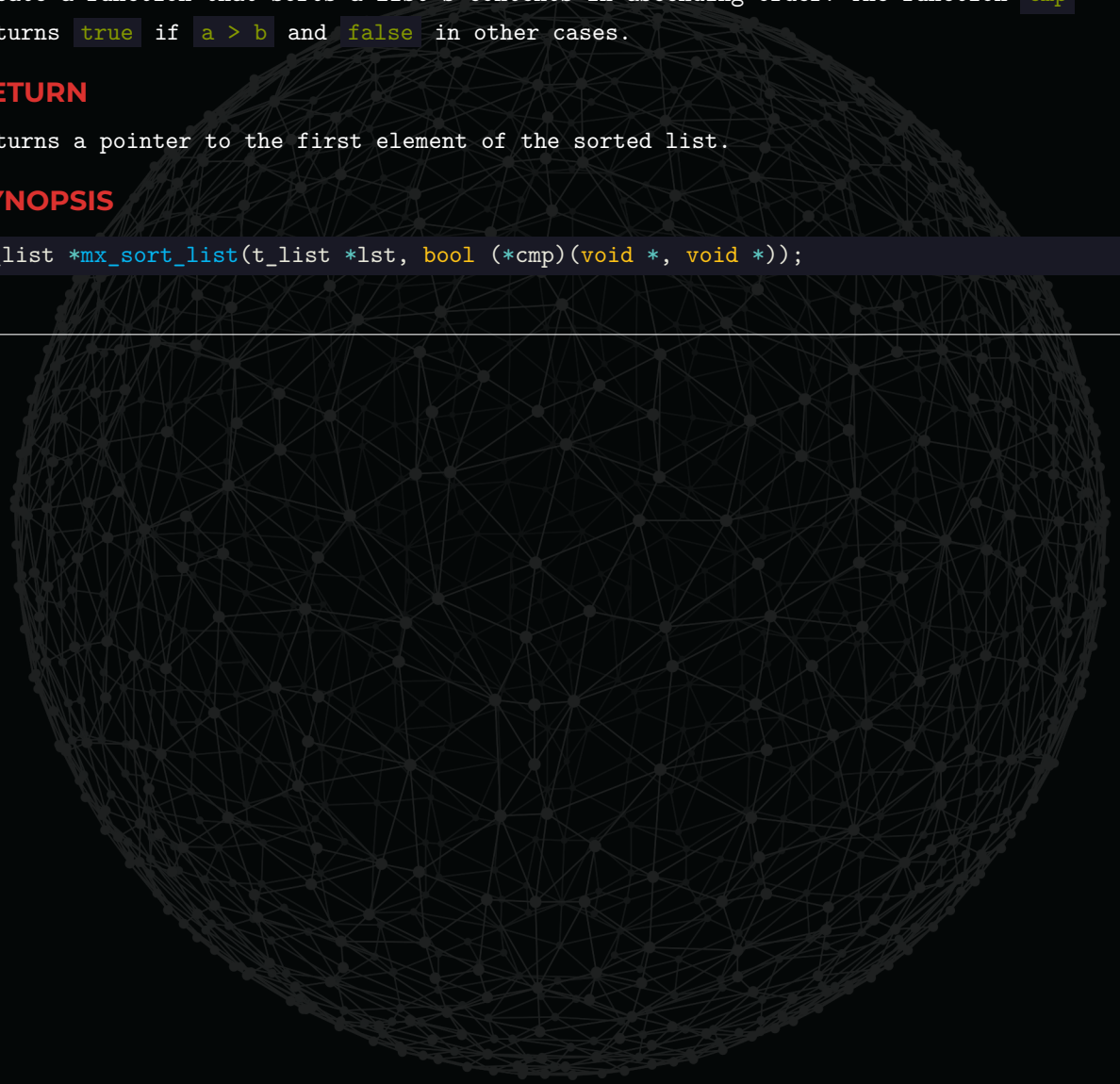
Create a function that sorts a list's contents in ascending order. The function `cmp` returns `true` if `a > b` and `false` in other cases.

RETURN

Returns a pointer to the first element of the sorted list.

SYNOPSIS

```
t_list *mx_sort_list(t_list *lst, bool (*cmp)(void *, void *));
```



Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.