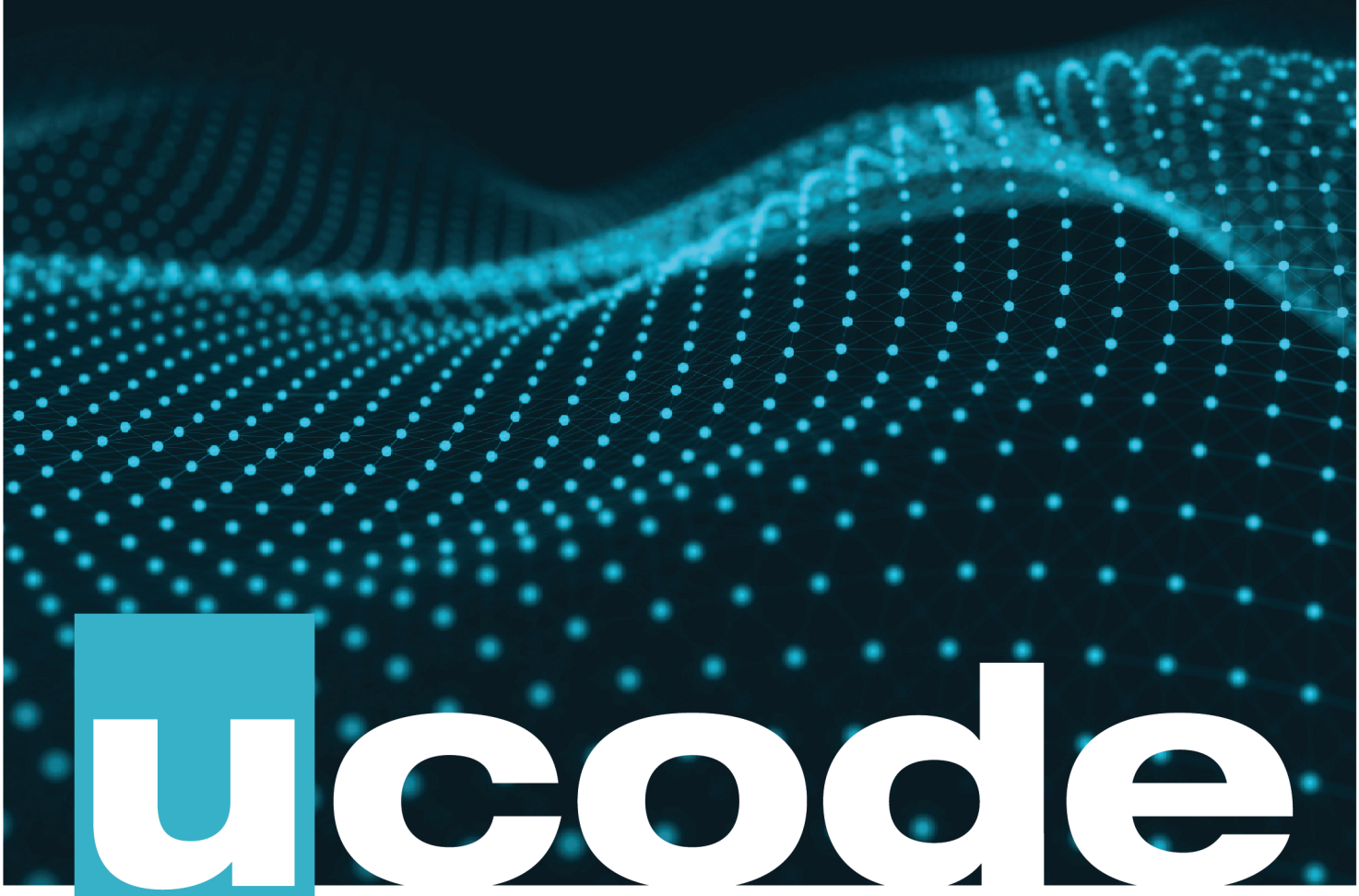
Libmx

Track C

October 28, 2019



ucode

Contents

ucode

Engage

DESCRIPTION

Hey there.

This is the next challenge in Track C - create your own library of functions. Writing this challenge will help you to save a great amount of your time in the future. After all, functions in this library can be reused in the next challenges of Track C.

Furthermore, you have a marvelous opportunity to expand your library with even more useful functions and make it unique.
Of course, we recommend you not to throw all your next challenges fully to the library.
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

Build your own library.

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. This will help you realize why you are here and what you want to get from it. And most importantly, how to move forward.

Ask your neighbor on the right, left, or behind you, and discuss the following questions together. You can walk around the campus and find yourself a comfortable place to do it.

We encourage you to ask as many personal and contextual questions as possible. Note down your discussion.

- Why are developers making their own libraries instead of ready-made solution?
- Have you ever developed any library?
- What do you know about dynamic libraries?
- What is the difference between static and dynamic libraries?
- What is the difference between framework and library?
- What skills do you want to get?
- What is a static variables and their lifecycle?
- What resources do you need to start learning?
- Are you ready to start?

GUIDING ACTIVITIES

These are only a set of example activities and resources. Do not forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

1. Define the terms of the challenge (library, function, solution, product etc).
2. Ask students who have already started this challenge what you need to figure out. Prepare your mind for an explosion.
3. Make a plan where to start. Find out which features you will need in the future, implement them firstly.
4. Think about extra functions in advance.
5. Come seriously to this task, because everything that you are doing now will be needed in the future.

ANALYSIS

Let's get started! And may the odds be ever in your Favor!

1. Carefully read the instructions, as well as the MAN for the functions that have similar behavior as standard libc functions. Understand what to do with it.
2. Ask your friends to test your solution. Everything is OK? Lucky! Repeat this cycle several times, collect feedback.
3. Think about how your solution could be improved. Which useful functions could be added to your library?

Act

SOLUTION DEVELOPMENT

- All content of the challenge files must be written in accordance to Auditor.
- Your challenge must have the next structure:
  - `src` directory contains files with extension `.c`
  - `obj` directory contains files with extension `.o` (you must not push this directory on your repository, only `Makefile` will create it during compilation)
  - `inc` directory contains header `libmx.h`
  - `Makefile` that compiles and builds `libmx.a`
- Makefile must be written in accordance to Auditor.
- 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 to write new ones. Stay DRY. Don't Repeat Yourself.
- You can add some custom functions if you need it.
- All unused memory should be freed when required.

ALLOWED FUNCTIONS

`malloc, malloc_size, free, open, read, write, close, exit`

Utils pack

Here you must add util functions, that will ease your work. Some of that functions you developed during Refresh Marathon C. You can find prototypes for every function below.

NAME

Print character

DESCRIPTION

Create a function that prints single character on the standard output.

SYNOPSIS

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

NAME

Print multibyte characters

DESCRIPTION

Create a function that prints a ASCII and multibyte characters on standard output.

SYNOPSIS

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

NAME

Print string

DESCRIPTION

Create a function that prints string of characters on the standard output.

SYNOPSIS

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

NAME

Print array of strings

DESCRIPTION

This function prints:

- array of strings `arr` on the standard output with delimiter `delim` between elements of array;
- nothing if `arr` or `delim` does not exist;
- the newline at the end of the output.

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

SYNOPSIS

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

NAME

Print integer

DESCRIPTION

Create a function that prints integer values on the standard output.

SYNOPSIS

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

CONSOLE OUTPUT

```
>./mx_printint | cat -e    # n = 25
25%
>./mx_printint | cat -e    # n = 2147483647
2147483647%
>
```

NAME

Exponentiation

DESCRIPTION

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

RETURN

Returns the result of `pow` times multiplying the number `n` by itself.

SYNOPSIS

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

EXAMPLE

```
mx_pow(5, 4); //returns 625
```

NAME

Square root

DESCRIPTION

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

RETURN

It returns the square root of the number `x` if it's 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.

RETURN

Returns number converted into 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 unsigned long number.

SYNOPSIS

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

EXAMPLE

```
hex_to_nbr("C4"); //returns 196
hex_to_nbr("FADE"); //returns 64222
hex_to_nbr("fffffffffff"); //returns 281474976710655
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

Return a number as a string terminating by `\0`.

SYNOPSIS

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

NAME

For each

DESCRIPTION

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

SYNOPSIS

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

NAME

Binary search

DESCRIPTION

Create a function that searches string `s` in array `arr` with the given size. Uses the binary search algorithm assuming that input array has already been sorted in lexicographical order.

RETURN

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

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
mx_binary_search(arr, 6, "aBc", &count);  //returns 2 and count = 1
```

NAME

Bubble sort

DESCRIPTION

Create a function that sorts an array of strings in place in lexicographical order using 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 length in ascending order using algorythm of quick sort. You must not check if `left` and `right` are correct.

RETURN

Return number of shifts or `-1` if `arr` does not exist.

SYNOPSIS

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

EXAMPLE

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

String pack

In this part you need to add the functions to operate with Strings. Some of that functions you developed during Refresh Marathon C. You can find prototypes for every function below.

NAME

String length

DESCRIPTION

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

SYNOPSIS

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

NAME

Swap characters

DESCRIPTION

Create a function which will swap the characters of the 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 which reverses string using pointers. Do nothing if 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

A function that takes a pointer to string, then frees the string memory with `free` and sets string to `NULL`.

SYNOPSIS

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

NAME

Delete array of strings

DESCRIPTION

Create a function that takes pointer to a NULL-terminated array of strings, deletes content of array, frees array memory with `free` and sets pointer to `NULL`.

SYNOPSIS

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

NAME

Get character index

DESCRIPTION

Create a function that finds index of the first occurrence of character `c` in a string `str`.

RETURN

Return index of the first occurrence. `-1` if no occurrence found and `-2` if string does not exist.

SYNOPSIS

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

NAME

Duplicate string

DESCRIPTION

Creat a function that has the same behaviour as 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 behaviour as 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 behaviour as standard libc function `strcpy`.

SYNOPSIS

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

NAME

Copy them all

DESCRIPTION

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

SYNOPSIS

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

NAME

Locate a substring

DESCRIPTION

Create a function which has the same behaviour as standard libc function `strstr`.

SYNOPSIS

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

NAME

Get substring index

DESCRIPTION

Create a function that finds a index of substring.

RETURN

Return

- index of `sub` in `str`;
- `-1` if `sub` isn't found in `str`;
- `-2` if `str` or `sub` does not exist.

and .

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 number of occurrences of substring in a string.

RETURN

Return number of occurrences of `sub` in `str` and `-1` if `str` or `sub` does not exist.

SYNOPSIS

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

EXAMPLE

```
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 which count words in the string. Word is a sequence of characters separated by delimiter.

RETURN

Returns number of words in the string.

SYNOPSIS

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

EXAMPLE

```
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 terminating `'\0'`;
- initializes each character with `'\0'`.

RETURN

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

SYNOPSIS

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

NAME

Trim string

DESCRIPTION

Create a function which creates new string without whitespace characters at the beginning and the end of the string and frees all unused memory with `free`.

RETURN

Returns new trimmed string or `NULL` if string str does not exist or trim of string 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 creates new string without whitespace characters in the beginning and at the end of a string. It puts in the new string exactly one space character between words and frees all unused memory. Word is a sequence of characters separated by whitespaces.

RETURN

Returns new created string in case if success or `NULL` if string str does not exist or creation of string 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 the NULL-terminated array of words and frees all unused memory.

RETURN

Returns NULL-terminated array of strings in case of success or `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 concatenates strings `s1` and `s2` into new string and terminates new string with `'\0'`.

RETURN

Returns a string as result of concatenation of `s1` and `s2` in case of `s1` and `s2` exists or new copy of non-NULL parameter if one and only one of the parameters is `NULL`. Function must return `NULL` if the join 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 takes a filename as a parameter and then reads the data from file into the string.

RETURN

Returns null-terminated string in case of success or `NULL` in case of any errors.

SYNOPSIS

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

NAME

Read lines

DESCRIPTION

Create a function that reads characters into the `lineptr` until at least one of the events occurs:

- read less than `buf_size` characters;
- reached the end of the file;

- reached `delim` character. This character should not be read.

Function must:

- work correct while reading from a file, stdin, stdout, stderr, redirection etc;
- read all text to the end if you call the function with the same parameters in the loop.

`mx_read_line` can have undefined behaviour while reading from binary file.

RETURN

- num of read bytes which was written to `lineptr` using file descriptor `fd`;
- `0` if reading with `fd` is over;
- `-1` if `fd` is invalid or you get error while trying to read with it.

SYNOPSIS

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

NAME

Replace substrings

DESCRIPTION

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

RETURN

Return a new string where sustrings replaced or `NULL` if `sub` or `str` or `replace` doesn't 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"
```

Memory pack

Proper work with memory is an integral and important part of the interaction of your program with a computer. For this we need some functions, here they are:

NAME

Fill memory

DESCRIPTION

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

SYNOPSIS

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

NAME

Locate byte from end

DESCRIPTION

The `mx_memrchr` function is like the `mx_memchr` function, except that it searches backward from the end of the `n` bytes pointed to by `s` instead of forward 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 behaviour as 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 behaviour as 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 behaviour as standard stdlib function `realloc`.

SYNOPSIS

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

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 list of tasks and `s_list` structure below. Your library header must contain structure `s_list` and required includes and prototypes to compile functions successfully.

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

NAME

Create node

DESCRIPTION

Create a function which creates a new node of linked list. Function must assign parameter `data` to the list variable `data`.

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 new 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 allocated for the node memory.

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 allocated for the node memory.

SYNOPSIS

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

NAME

Size of list

DESCRIPTION

Create a function that calculates the number of nodes in the 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 the list's contents in ascending order. Function `cmp` returns true if `a > b` and false in other cases.

SYNOPSIS

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

Share

PUBLISHING

The final important and integral stage of your work is its publishing. This allows you to share your challenges, solutions, and reflections with a local and global audience.

During this stage, you will find how to get a global assessment. You will get representative feedback. As a result, you get the maximum experience from the work you have done.

What you can create to disseminate information

- Text post, summary from reflection.
- Charts, infographics or any other ways to visualize your information.
- Video of your work, reflection video.
- Audio podcast. You can record a story with your experience.
- Photos from Ucode with small post.

Example techniques

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

Example ways to share your experience

- Facebook - create a post that will inspire your friends.
- YouTube - upload a video.
- GitHub - share your solution.
- Telegraph - create a post. This is a good way to share information in a Telegram.
- Instagram - share a photos and stories from ucode. Don't forget to tag us :)

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