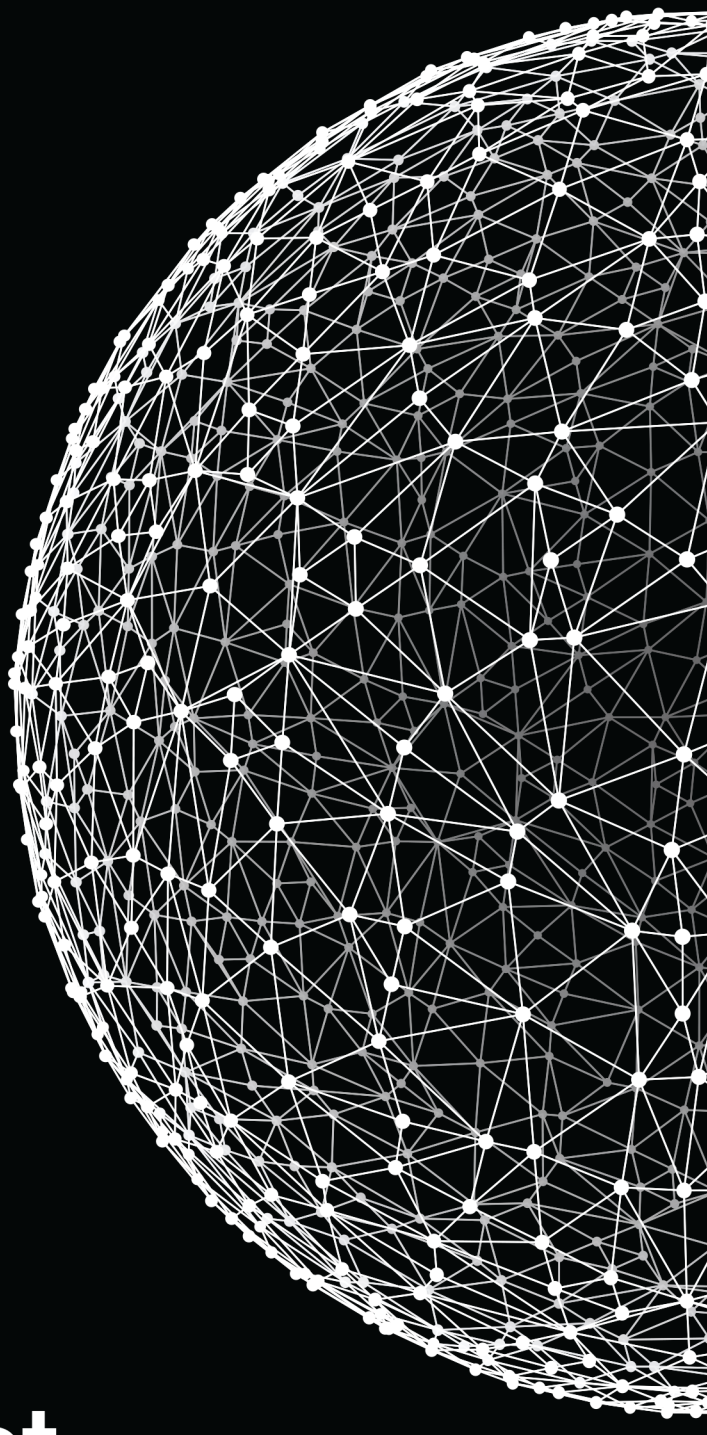
Sprint 10

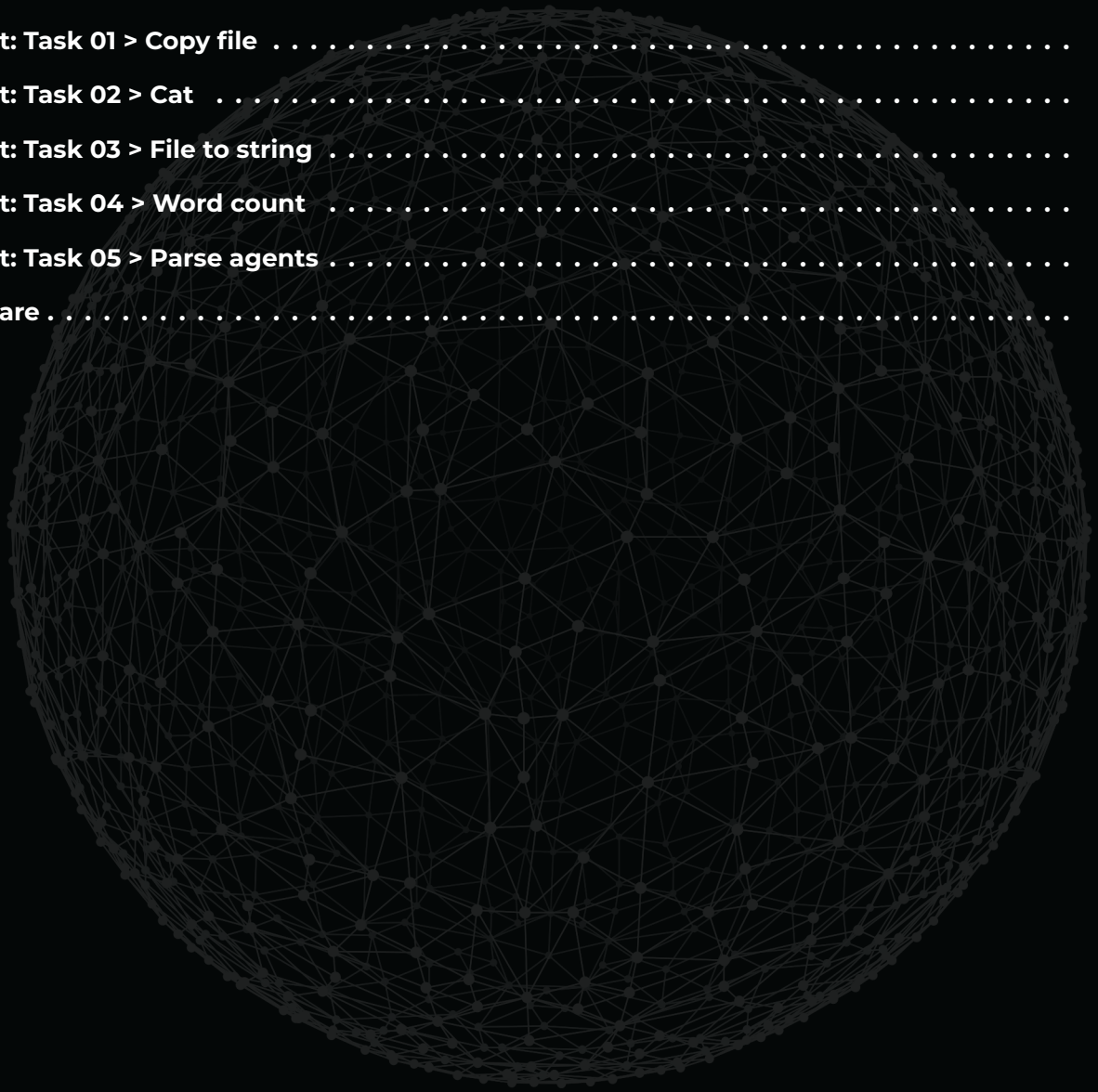
Marathon C

October 15, 2020



ucode connect

Contents



ucode connect

Engage

DESCRIPTION

Hey, guys and gals!

While using a personal computer we always use a filesystem, even if we don't realize it. Storing, copying, removing, cataloging files in directory trees with the help of cute GUI apps are common actions for daily PC use. Have you ever wondered what is hidden under the hood of those apps?

`C` comes with very powerful functions for file handling operations, such as opening, closing, reading, and writing. During this Sprint, you will learn file input/output streams and implement several system utilities.

Therefore, treat this challenge with special attention. This knowledge is extremely important!

BIG IDEA

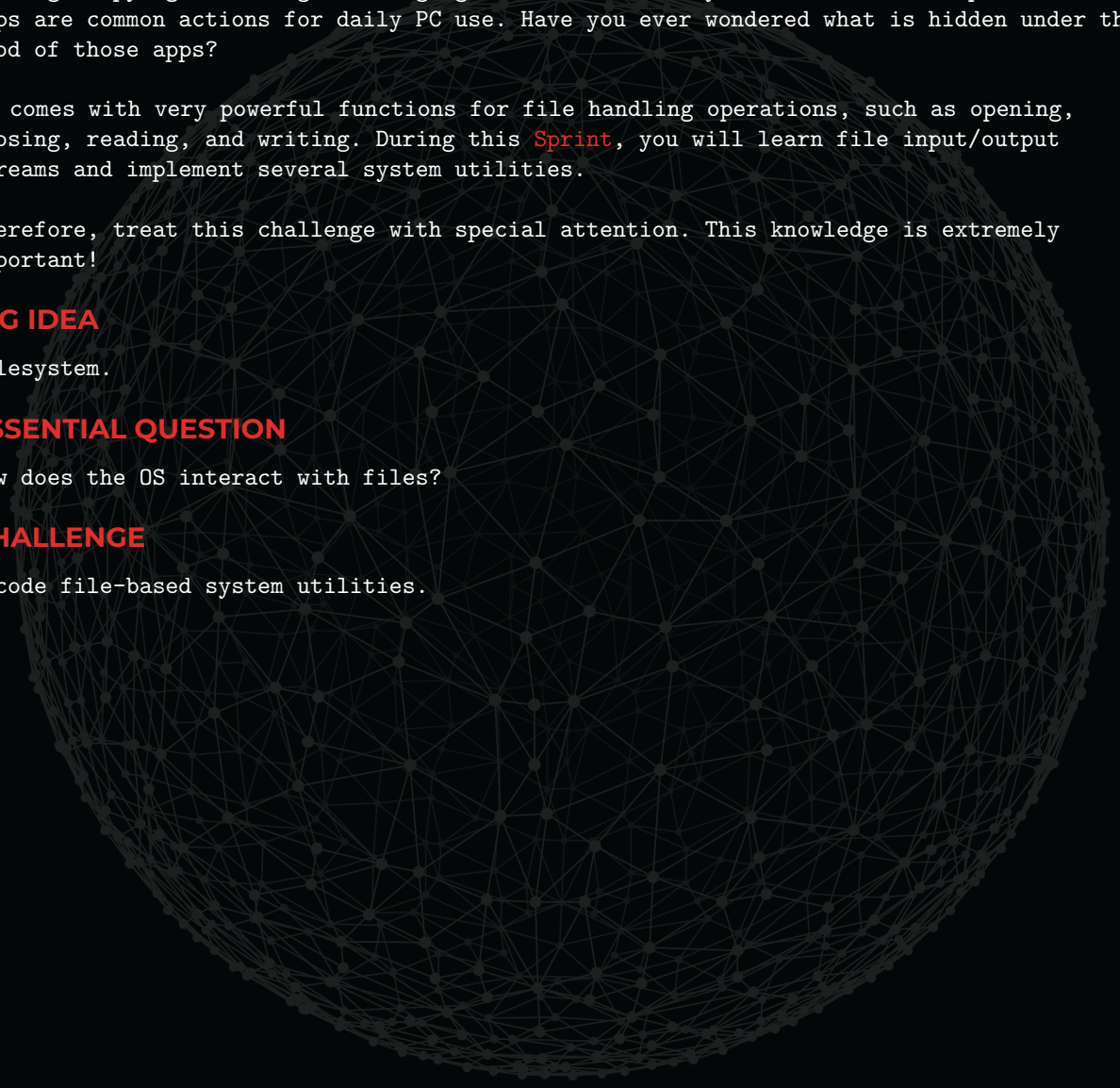
Filesystem.

ESSENTIAL QUESTION

How does the OS interact with files?

CHALLENGE

Recode file-based system utilities.



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 is a file?
- Which Unix utilities do you know to manipulate files?
- How many methods do you know to create a file from the command-line?
- How can you get/put the necessary data from/in the file?
- What system streams exist?
- What is a file descriptor?
- How to open a file?
- Why is it necessary to close a file after input/output operations?
- What happens if the file can not be read?
- Is it possible to read a directory instead of a file?
- What types of files exist in the macOS filesystem?
- What is a buffer?
- Have you heard about symbolic links? What is their purpose? Where can you apply them?
- Do you need to read the Auditor once again? Are you sure that you've read the Makefile chapter carefully?

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.

- Start by figuring out what a file is on Unix filesystems.
- Learn in detail the operation of such utilities: `cat`, `cp`, `wc`. It will definitely come in handy.
- Learn how to work with standard streams in C.
- Explore how you can read information from a file.
- Find out how to compose a Makefile correctly.
- Clone your git repository that is issued on the challenge page in the LMS.
- Start to develop the solution. Suggest improvements. Test your code.
- Explore new things for you.
- Communicate with students and share information.

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.
- Analyze all information you have collected during the preparation stages. Try to define the order of your actions.
- Submit your files using the format described in the story. Only useful files allowed, garbage shall not pass!
- Pay attention to what is allowed. Use of forbidden stuff is considered a cheat and your challenge will be failed.
- The solution will be checked and graded by students like you. Use Peer-to-Peer learning.
- If you have any questions or don't understand something, ask other students or just Google it.
- Be attentive to all statements of the story. Examine the given examples carefully. They may contain details that are not mentioned in the task.
- Analyze all information you have collected during the preparation stages.
- Perform only those tasks that are given in this document.
- 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`.
- Your program must manage memory allocations correctly. A memory that is no longer needed must be freed, otherwise, the task is considered incomplete.
- Pay attention to what is allowed in a certain task. Use of forbidden stuff is considered a cheat and your tasks will be failed.
- Complete tasks according to the rules specified in the `Auditor`.
- 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.
- Use your brain and follow the white rabbit to prove that you are the Chosen one!

Act: Task 00

NAME

Read file

DIRECTORY

`t00/`

SUBMIT

`Makefile, inc/*.[h], src/*.[c]`

ALLOWED FUNCTIONS

`open, read, close, write`

BINARY

`read_file`

DESCRIPTION

Create a program that prints:

- the contents of a file given as an argument to the standard output
- `error` to the `stderr` followed by a newline in case of any errors
- `usage: ./read_file [file_path]` to the `stderr` followed by a newline if no or too many arguments have been given

CONSOLE OUTPUT

```
>./read_file | cat -e
usage: ./read_file [file_path]
>./read_file file.txt
#the contents of the file
>./read_file unknown_file | cat -e
error
>
```

FOLLOW THE WHITE RABBIT

`man stderr`

Act: Task 01

NAME

Copy file

DIRECTORY

`t01/`

SUBMIT

`Makefile, inc/*.[h], src/*.[c]`

ALLOWED FUNCTIONS

`open, read, close, write, strerror, exit`

BINARY

`mx_cp`

DESCRIPTION

Create a program that:

- copies the contents of a source file to a new file
- does nothing if a destination file already exists
- prints the respective message to the `stderr` followed by a newline if a source file does not exist

Tips for this task

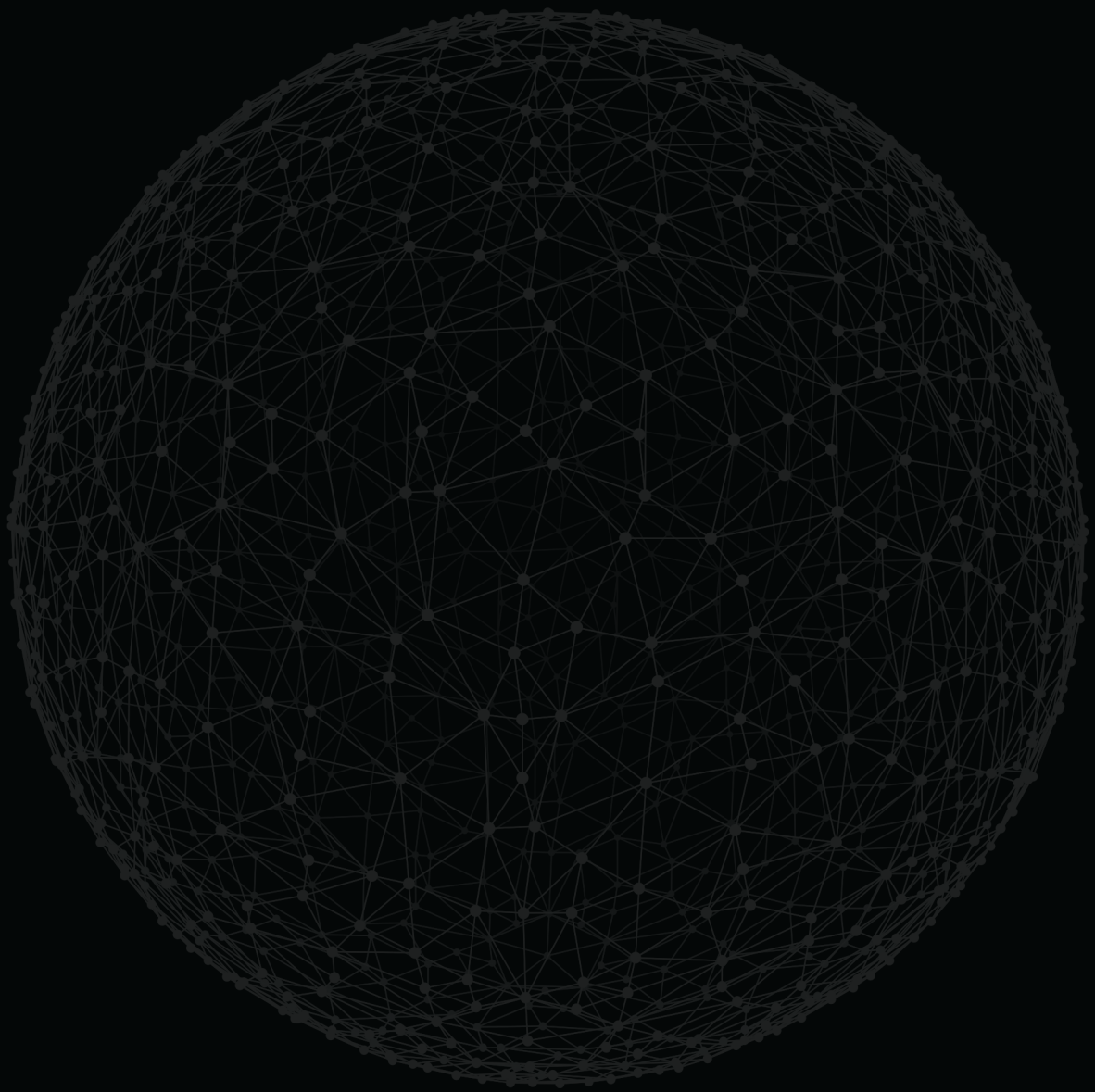
1. The program is similar with the utility `cp`.
2. You do not need to process `cp` options.
3. Use the `standard error` for printing errors.
4. You can use `errno`.

CONSOLE OUTPUT

```
>./mx_cp src_file | cat -e
usage: ./mx_cp [source_file] [destination_file]
>cat -e src_file
cat: src_file: No such file or directory
>./mx_cp src_file dst_file | cat -e
mx_cp: src_file: No such file or directory
>echo "Matrix" > src_file
>cat -e src_file
Matrix$
>cat -e dst_file
cat: dst_file: No such file or directory
>./mx_cp src_file dst_file | cat -e
>cat -e dst_file
Matrix$
>
```

FOLLOW THE WHITE RABBIT

```
man cp
man stderr
man errno
```



Act: Task 02

NAME

Cat

DIRECTORY

`t02/`

SUBMIT

`Makefile, inc/*.[h], src/*.[c]`

ALLOWED FUNCTIONS

`open, read, close, write, strerror, exit`

BINARY

`mx_cat`

DESCRIPTION

Create a program that:

- has the same behaviour as the system utility `cat`
- prints the respective message to the `stderr` followed by a newline if a source file does not exist

Tips for this task

1. You do not need to process `cat` options.
2. Use the `standard error` for printing some errors.
3. You can use `errno`.

CONSOLE OUTPUT

```
>./mx_cat | cat -e
hello
hello$
>./mx_cat asdfg | cat -e
mx_cat: asdfg: No such file or directory
>./mx_cat Makefile
#contents of the file
>
```

FOLLOW THE WHITE RABBIT

```
man cat
man stderr
man errno
```

Act: Task 03

NAME

File to string

DIRECTORY

`t03/`

SUBMIT

`file_to_str.h, mx_file_to_str.c, mx_strjoin.c, mx_strcat.c, mx_strcpy.c, mx_strdup.c, mx_strlen.c, mx_strnew.c`

ALLOWED FUNCTIONS

`malloc, free, open, read, close`

DESCRIPTION

Create a function that:

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

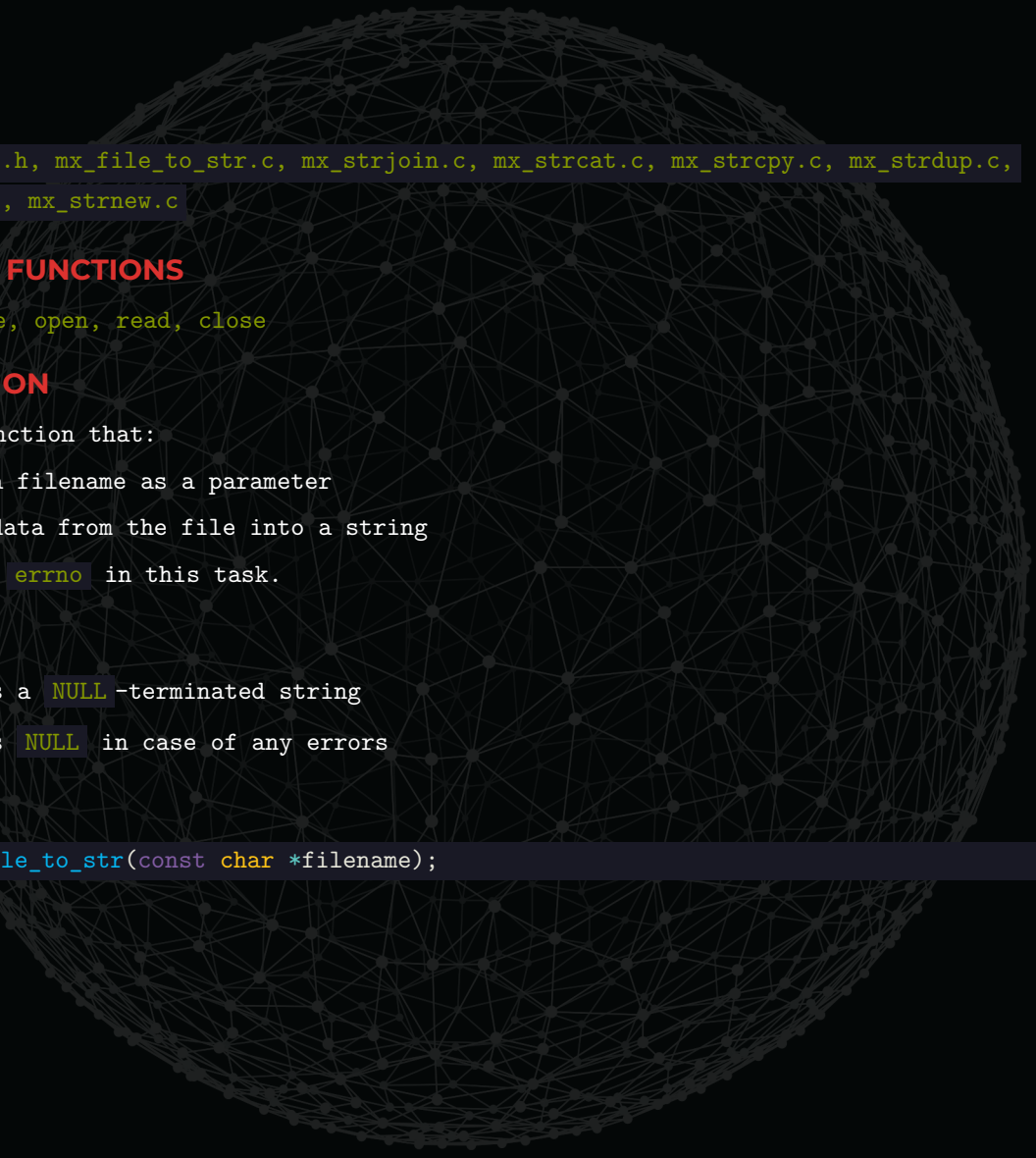
You can use `errno` in this task.

RETURN

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

SYNOPSIS

```c
char *mx_file_to_str(const char *filename);
```



Act: Task 04

NAME

Word count

DIRECTORY

`t04/`

SUBMIT

`Makefile, inc/*.[h], src/*.[c]`

ALLOWED FUNCTIONS

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

BINARY

`mx_wc`

DESCRIPTION

Create a program that:

- has the same behaviour as the system utility `wc` without flags
- separates the output of counted words, lines and bytes by a single tab character `\t`

Tips for this task

1. Use the `standard error` for printing errors.
2. You can use `errno`.
3. You must not process binary files. It is not a goal of this task.

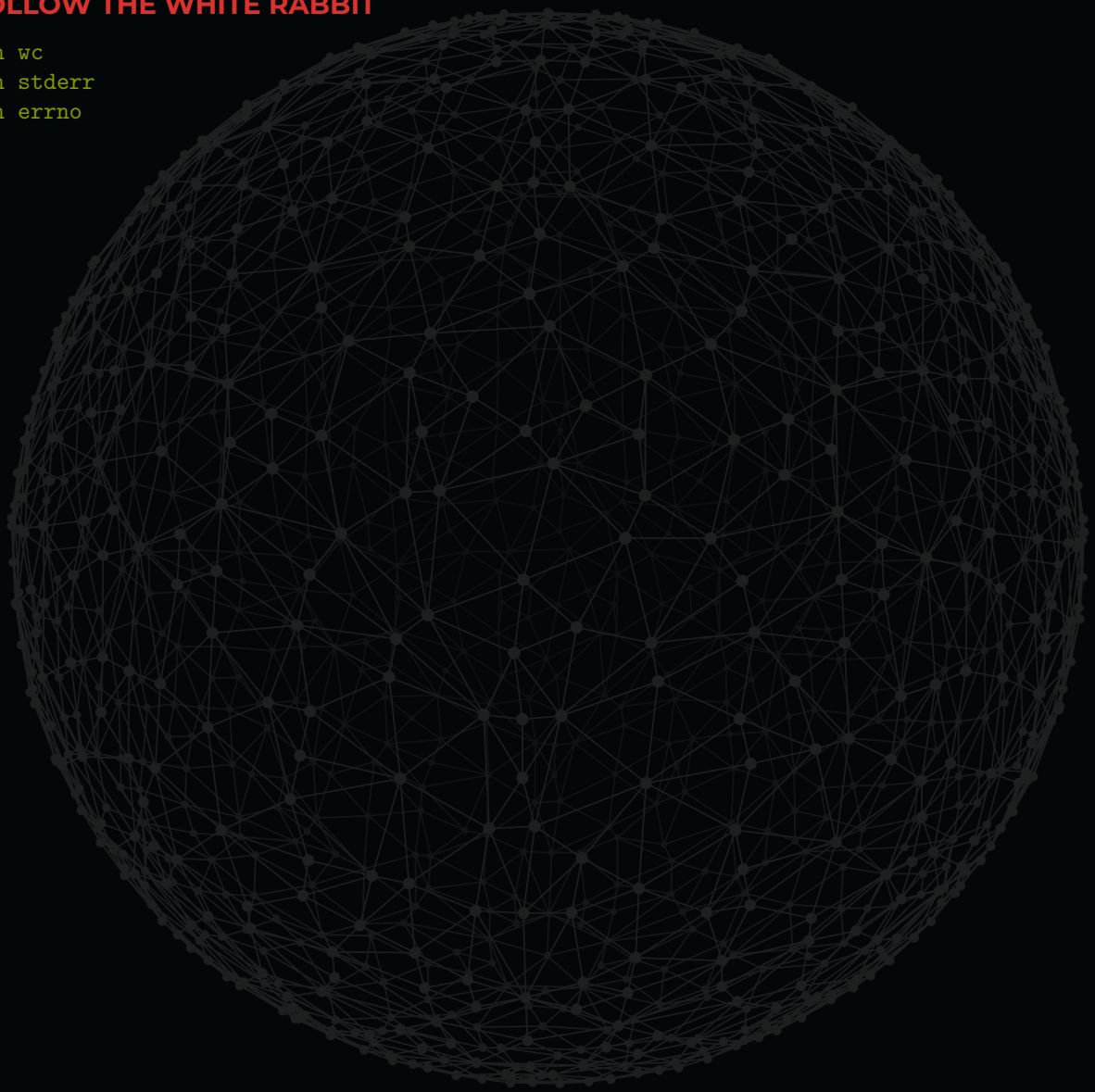
CONSOLE OUTPUT

```
>./mx_wc | cat -e
a
b
c
d
^D      4   4   8$
>cat -e file1
Neo$
Thomas$
Keanu$
>cat -e file2
Morpheus$
Trinity$
>./mx_wc file1 file2 | cat -e
    3   3   17 file1$
    2   2   17 file2$
    5   5   34 total$
>./mx_wc asdfg | cat -e
mx_wc: asdfg: open: No such file or directory
```

```
>./mx_wc . | cat -e
mx_wc: .: read: Is a directory
>
```

FOLLOW THE WHITE RABBIT

```
man wc
man stderr
man errno
```



Act: Task 05

NAME

Parse agents

DIRECTORY

`t05/`

SUBMIT

`Makefile, inc/*.[h], src/*.[c]`

ALLOWED FUNCTIONS

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

BINARY

`parse_agents`

DESCRIPTION

Create a program that:

- takes as input a file of a pseudo-json format given in `resources`
- parses it into an array of structures `s_agent`
- prints the agents to the standard output sorted in ascending order
- prints `error` to the `stderr` followed by a newline in case of invalid file format or any other errors
- sorts according to flags `-p`, `-s`, `-n` to sort by power, strength and name by ASCII order respectively
- informs the user if no arguments have been given or an invalid flag has been provided with a message to the `stderr`: `usage: ./parse_agents [-p | -s | -n] [file_name]`

You must find out the criteria of a valid file format by yourself analyzing the given example.

SYNOPSIS

```
typedef struct  s_agent
{
    char *name;
    int power;
    int strength;
}               t_agent;
```

CONSOLE OUTPUT

```
>./parse_agents | cat -e
usage: ./parse_agents [-p | -s | -n] [file_name]
>./parse_agents -s invalid_file | cat -e
error
>./parse_agents resources/agents | cat -e
usage: ./parse_agents [-p | -s | -n] [file_name]
>./parse_agents -s resources/agents | cat -e
agent: Smith, strength: 1, power: 3$
agent: Mulder, strength: 2, power: 5$
agent: Snowden, strength: 3, power: 8$
agent: Bond, strength: 9, power: 1$
>./parse_agents -z resources/agents | cat -e
usage: ./parse_agents [-p | -s | -n] [file_name]
>
```

FOLLOW THE WHITE RABBIT

```
man stderr
```

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.