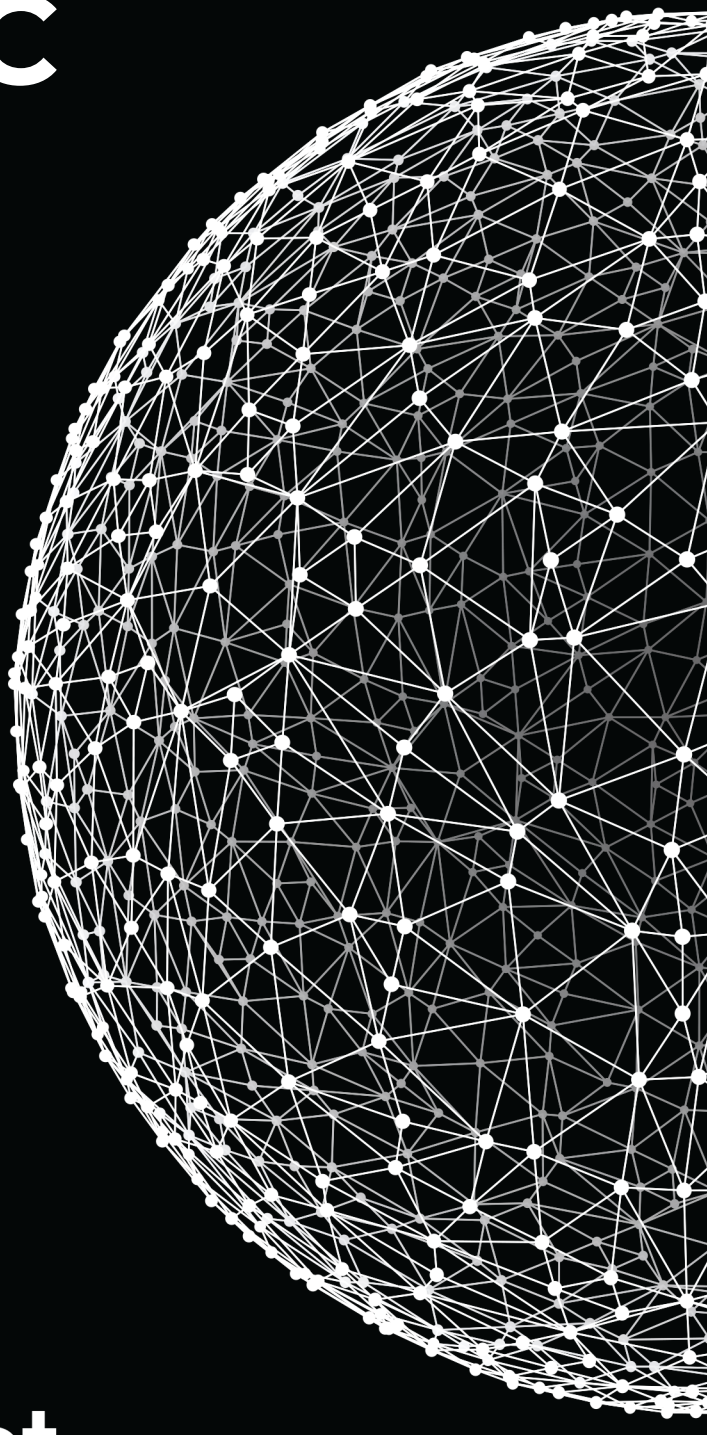
Refresh Marathon C

Track C



September 4, 2020

ucode connect

Contents

Engage

DESCRIPTION

Hello!

We invite you to continue your fascinating journey.
This challenge is an excellent way to refresh your knowledge of the basics of the C programming language.
It will be useful in preparing for the next challenges. Don't look back to how you've completed these tasks during the Marathon. Challenge yourself and recreate your work.
You'll do a better job this time. A repetition of knowledge is what will be beneficial to you.

Good luck, The One!

BIG IDEA

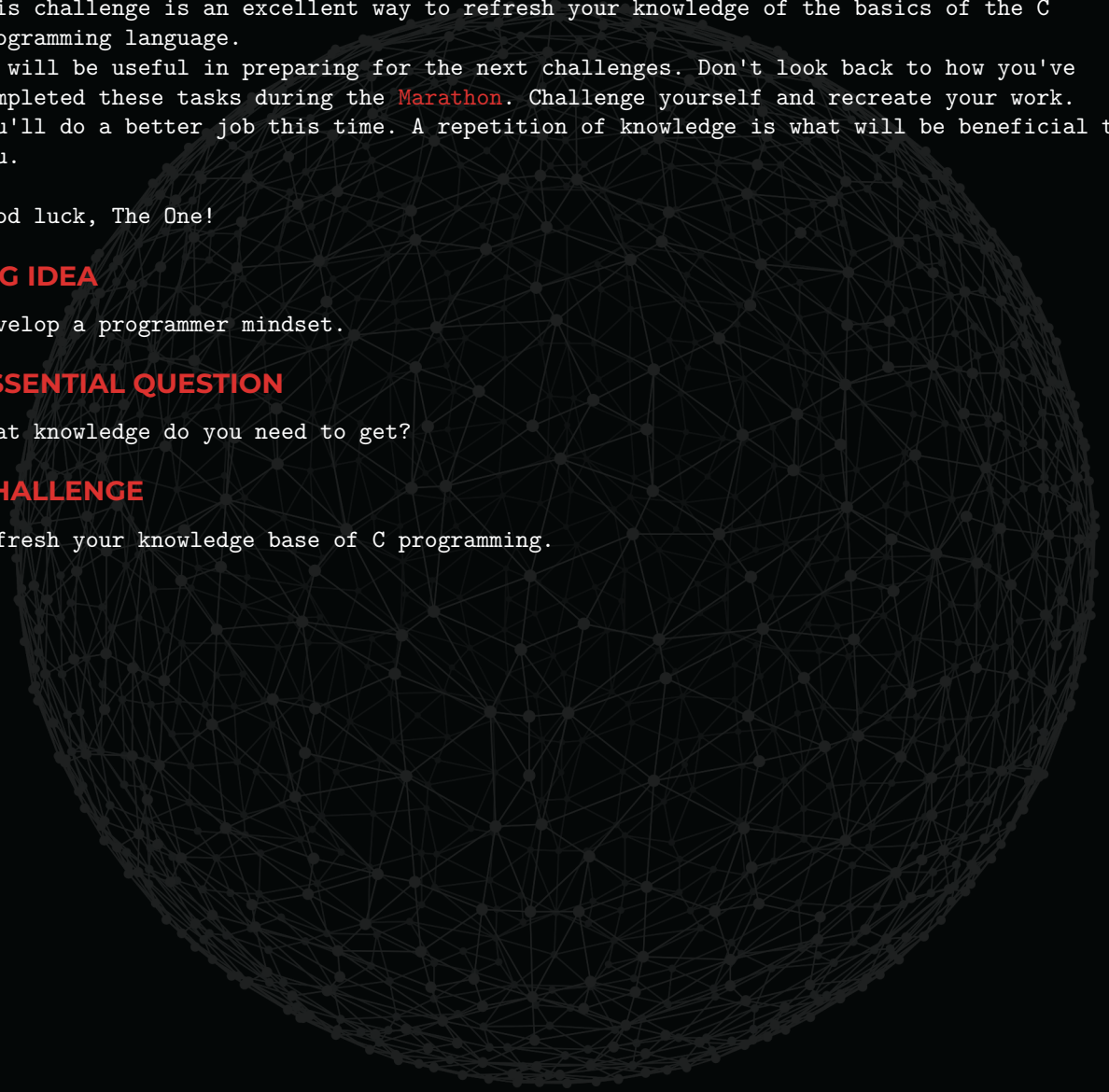
Develop a programmer mindset.

ESSENTIAL QUESTION

What knowledge do you need to get?

CHALLENGE

Refresh your knowledge base of C programming.



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 do you remember from the Marathon C?
- What topics have you covered?
- Do you remember what a variable is?
- What is a function?
- What is a loop in programming?
- What is a program?
- What do you think should be paid attention to and repeated well?

GUIDING ACTIVITIES

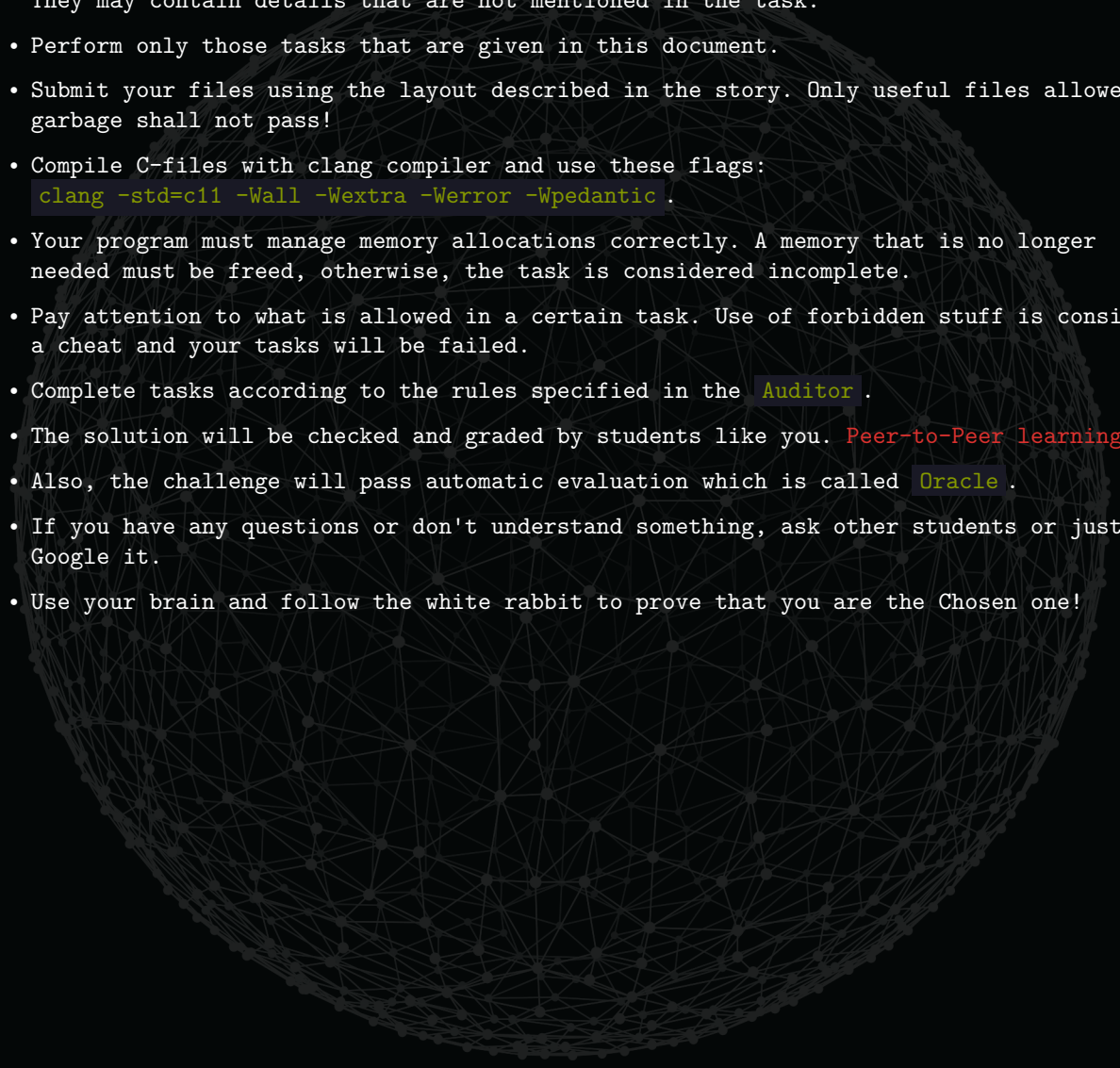
Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Read the story. Find familiar tasks.
- Learn more about the tasks you need to complete. Catch up on the topics that you didn't have enough time for during the Marathon.
- As a developer, you need to have a well-integrated development environment that perfectly fits your preferences. Read about:
  - Sublime Text
  - Atom
  - CLion
  - Visual Studio Code
- Keep in mind that Terminal support makes it easy to submit tasks to the repository and navigate directories.
- But, of course, everything is at your discretion and convenience.
- So, find out selected IDE/IDEs, configure, try to do some tasks from this story with this tool.
- Clone your git repository that is issued on the challenge page in the LMS.
- Write code. Compile the written code and run your program.
- Don't look back to how you've completed these tasks during the Marathon. Recreate your work.
- Arrange to brainstorm tasks with other 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. Examine the given examples carefully. They may contain details that are not mentioned in the task.
- 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

Print character

DIRECTORY

`t00/`

SUBMIT

`mx_printchar.c`

ALLOWED FUNCTIONS

`write`

DESCRIPTION

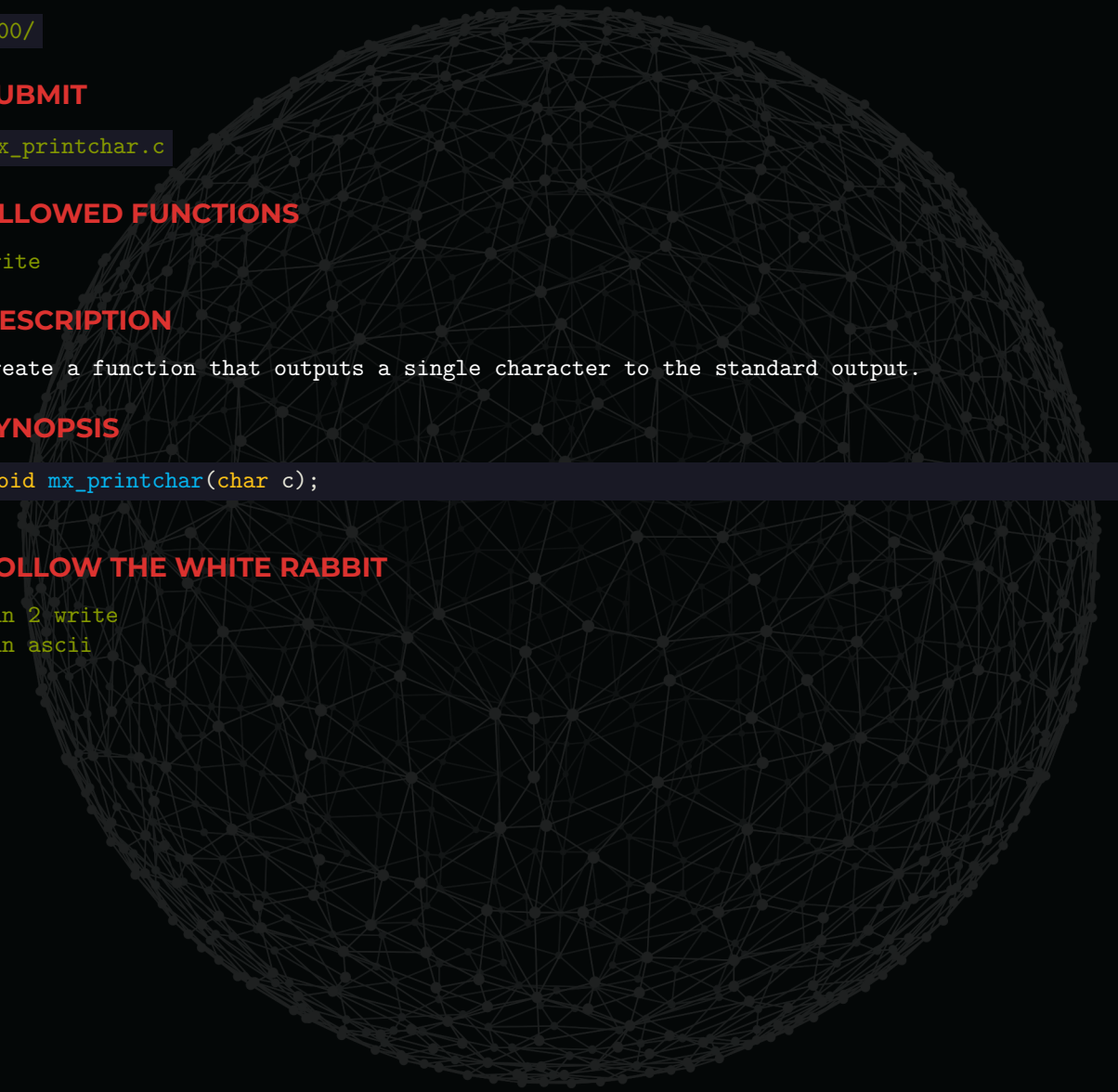
Create a function that outputs a single character to the standard output.

SYNOPSIS

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

FOLLOW THE WHITE RABBIT

```
man 2 write
man ascii
```



Act: Task 01

NAME

Print alphabet

DIRECTORY

`t01/`

SUBMIT

`mx_print_alphabet.c, mx_printchar.c`

ALLOWED FUNCTIONS

`write`

DESCRIPTION

Create a function that outputs the alphabet, alternating upper and lower case characters in ascending order to the standard output followed by a newline. See the output in the EXAMPLE below.

SYNOPSIS

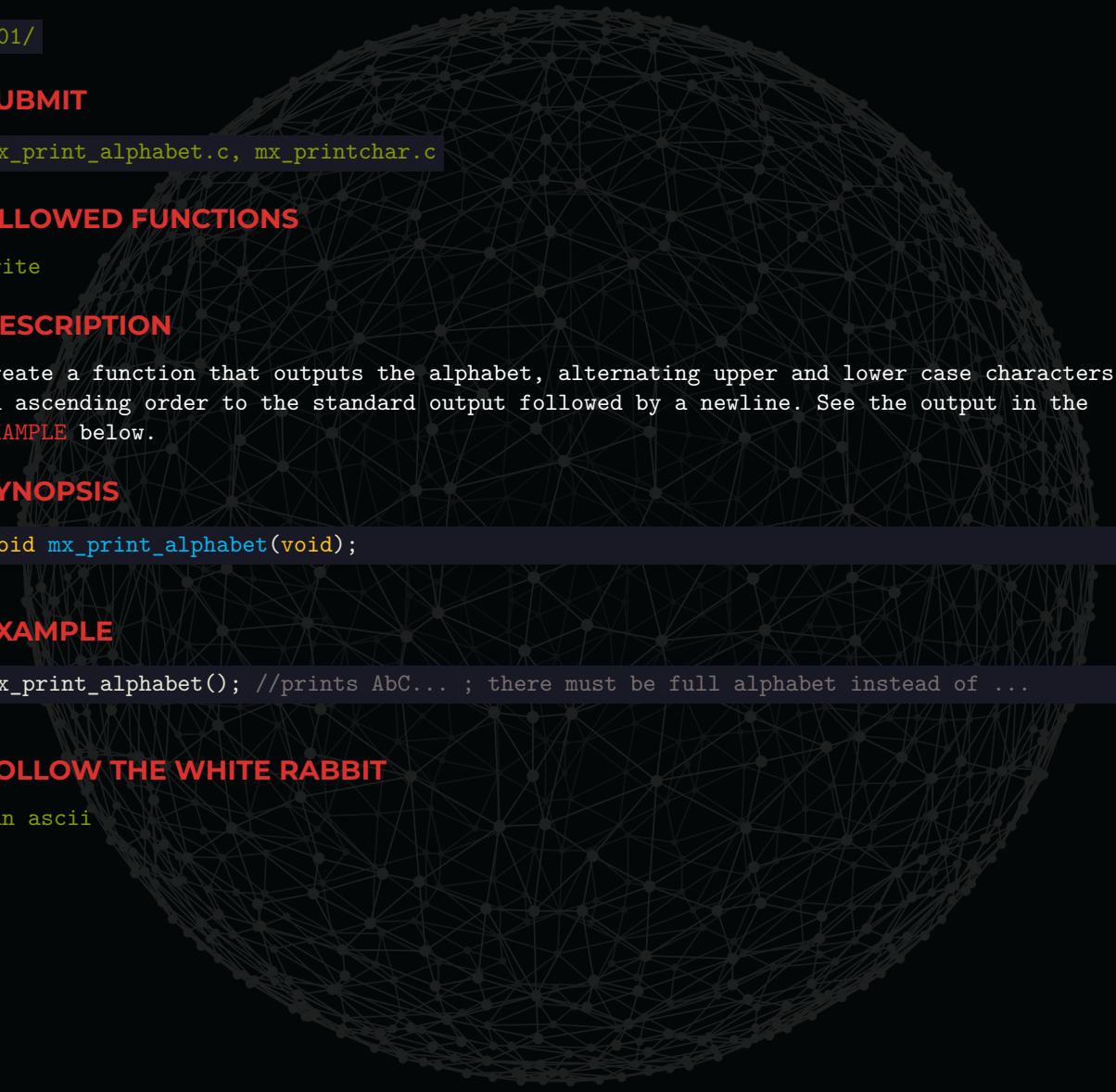
```
void mx_print_alphabet(void);
```

EXAMPLE

```
mx_print_alphabet(); //prints AbC... ; there must be full alphabet instead of ...
```

FOLLOW THE WHITE RABBIT

`man ascii`



Act: Task 02

NAME

String length

DIRECTORY

`t02/`

SUBMIT

`mx_strlen.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

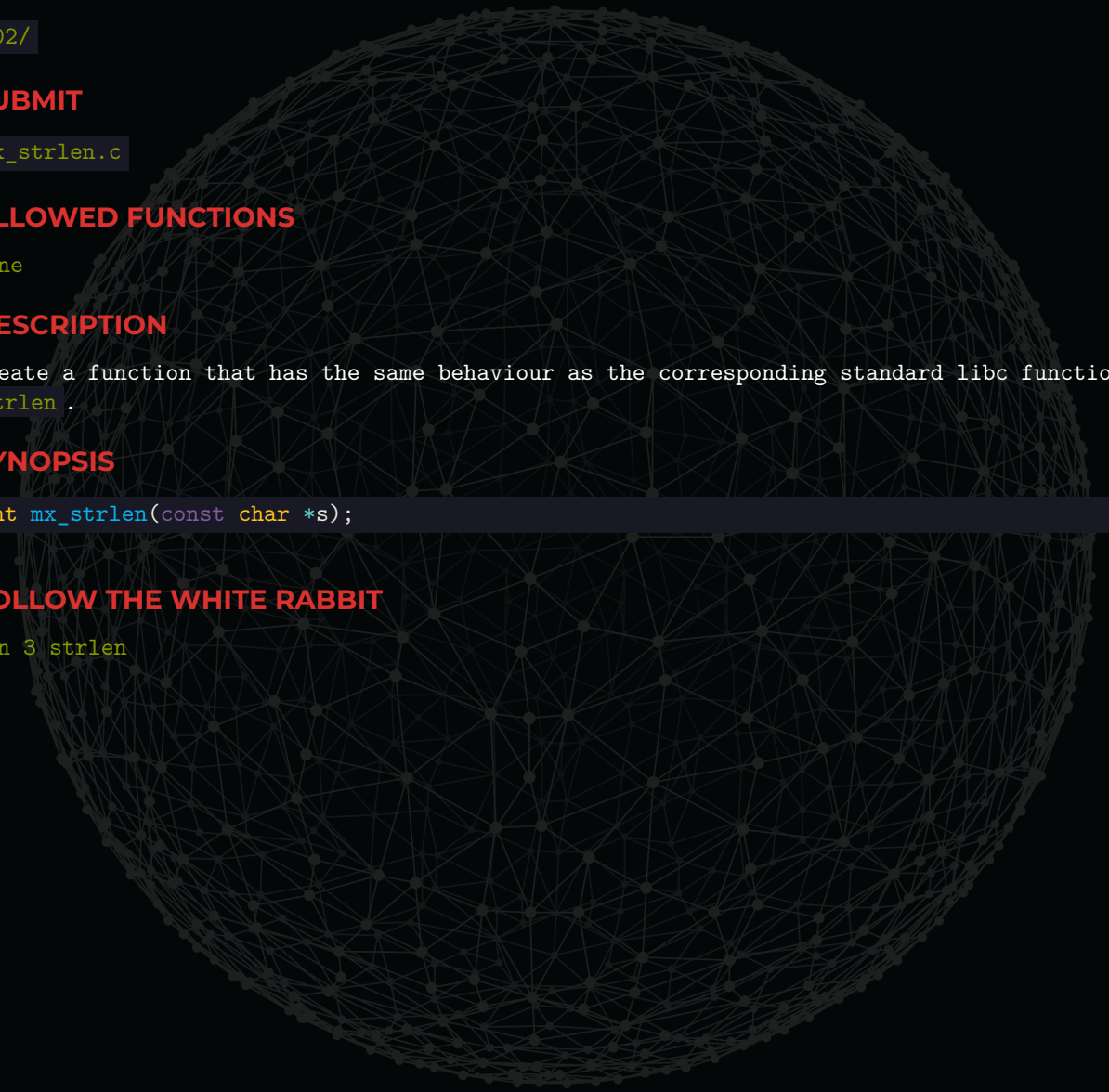
Create a function that has the same behaviour as the corresponding standard libc function `strlen` .

SYNOPSIS

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

FOLLOW THE WHITE RABBIT

`man 3 strlen`



Act: Task 03

NAME

Print string

DIRECTORY

`t03/`

SUBMIT

`mx_printstr.c, mx_strlen.c`

ALLOWED FUNCTIONS

`write`

DESCRIPTION

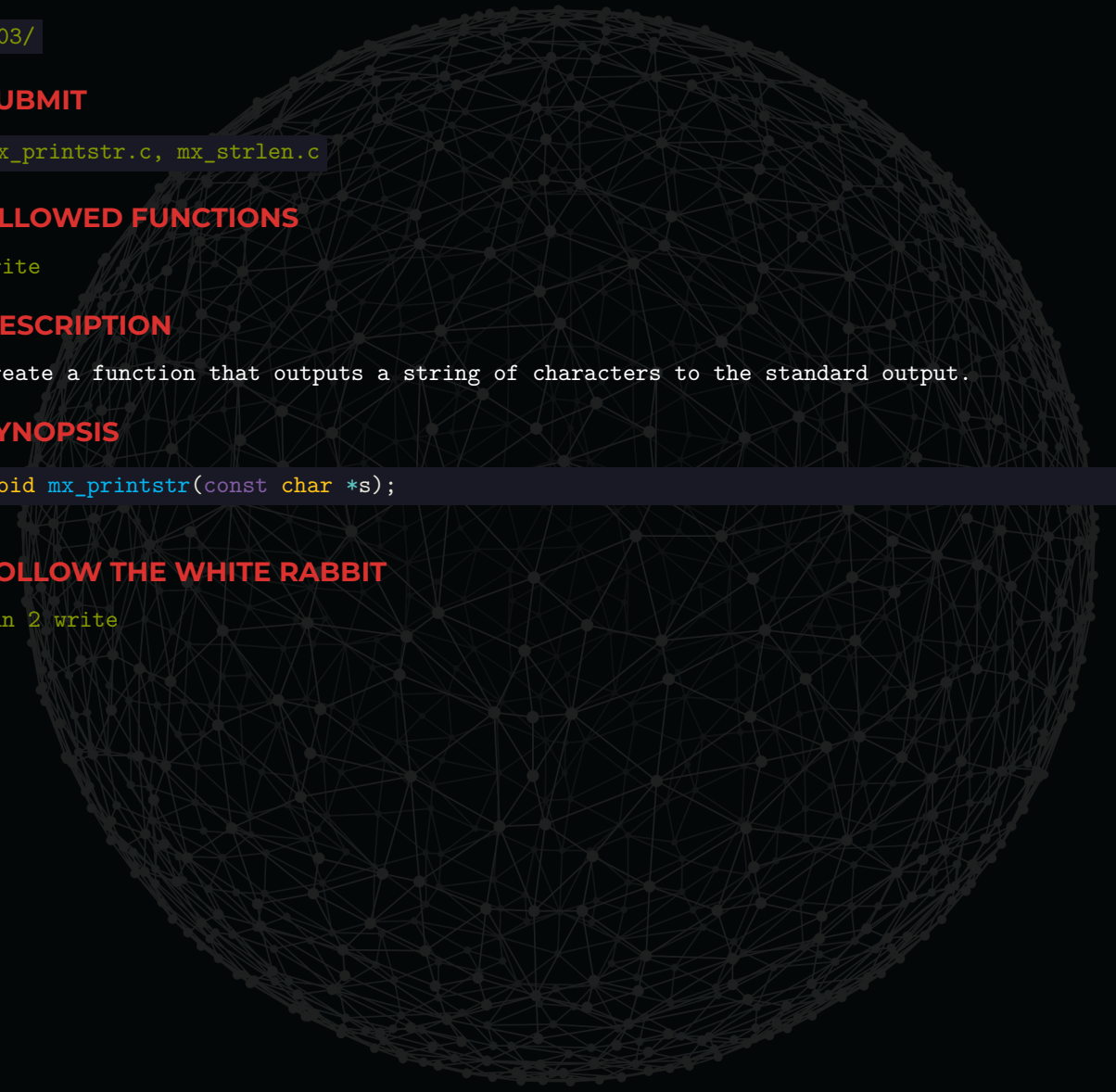
Create a function that outputs a string of characters to the standard output.

SYNOPSIS

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

FOLLOW THE WHITE RABBIT

`man 2 write`



Act: Task 04

NAME

Print integer

DIRECTORY

`t04/`

SUBMIT

`mx_printint.c, mx_printchar.c`

ALLOWED FUNCTIONS

`write`

DESCRIPTION

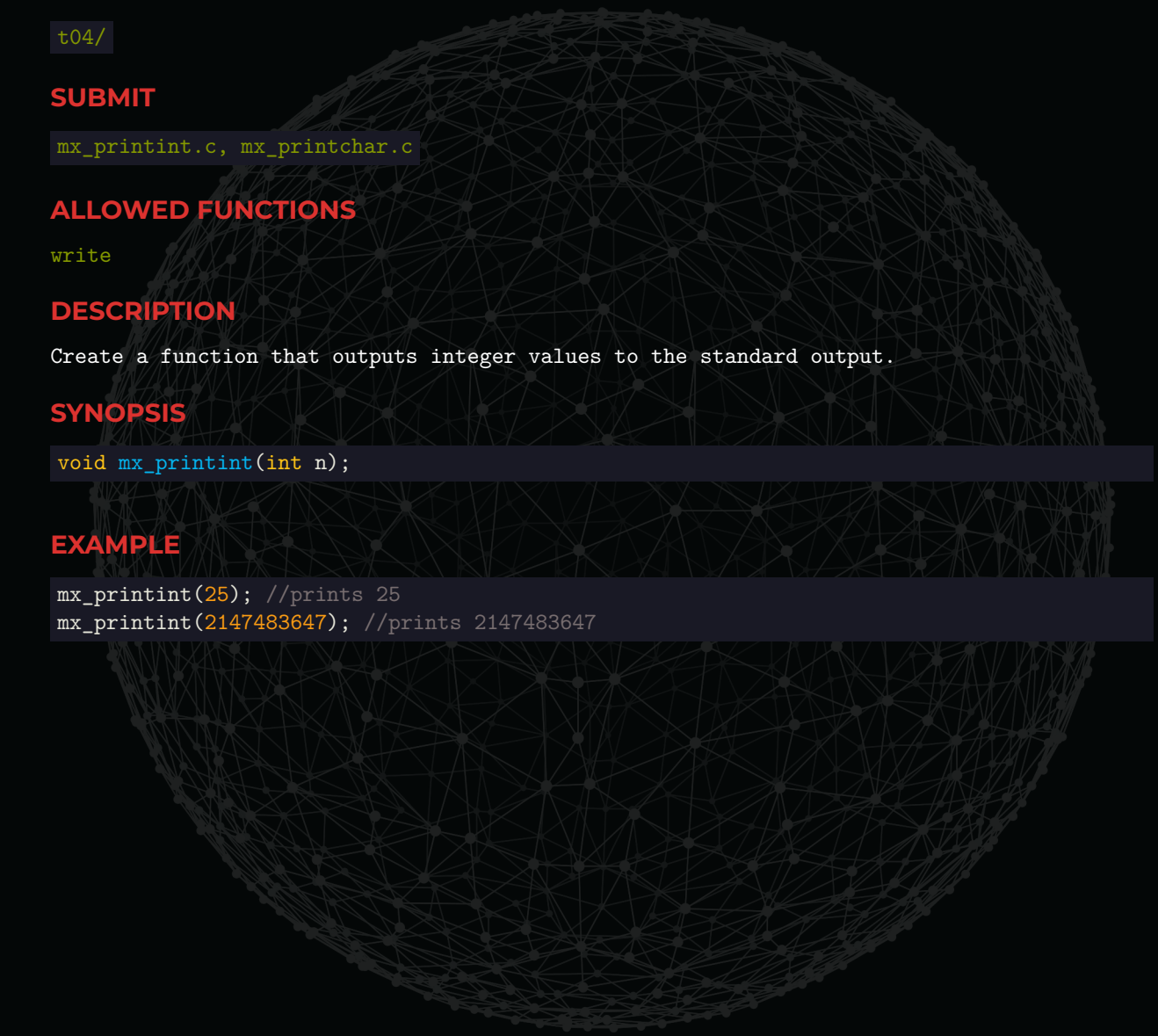
Create a function that outputs integer values to the standard output.

SYNOPSIS

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

EXAMPLE

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



Act: Task 05

NAME

Exponentiation

DIRECTORY

`t05/`

SUBMIT

`mx_pow.c`

ALLOWED FUNCTIONS

None

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

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

EXAMPLE

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

FOLLOW THE WHITE RABBIT

`man pow`

SEE ALSO

Exponentiation

Act: Task 06

NAME

Square root

DIRECTORY

`t06/`

SUBMIT

`mx_sqrt.c`

ALLOWED FUNCTIONS

None

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

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

EXAMPLE

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

Act: Task 07

NAME

Sort array

DIRECTORY

`t07/`

SUBMIT

`mx_sort_arr_int.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

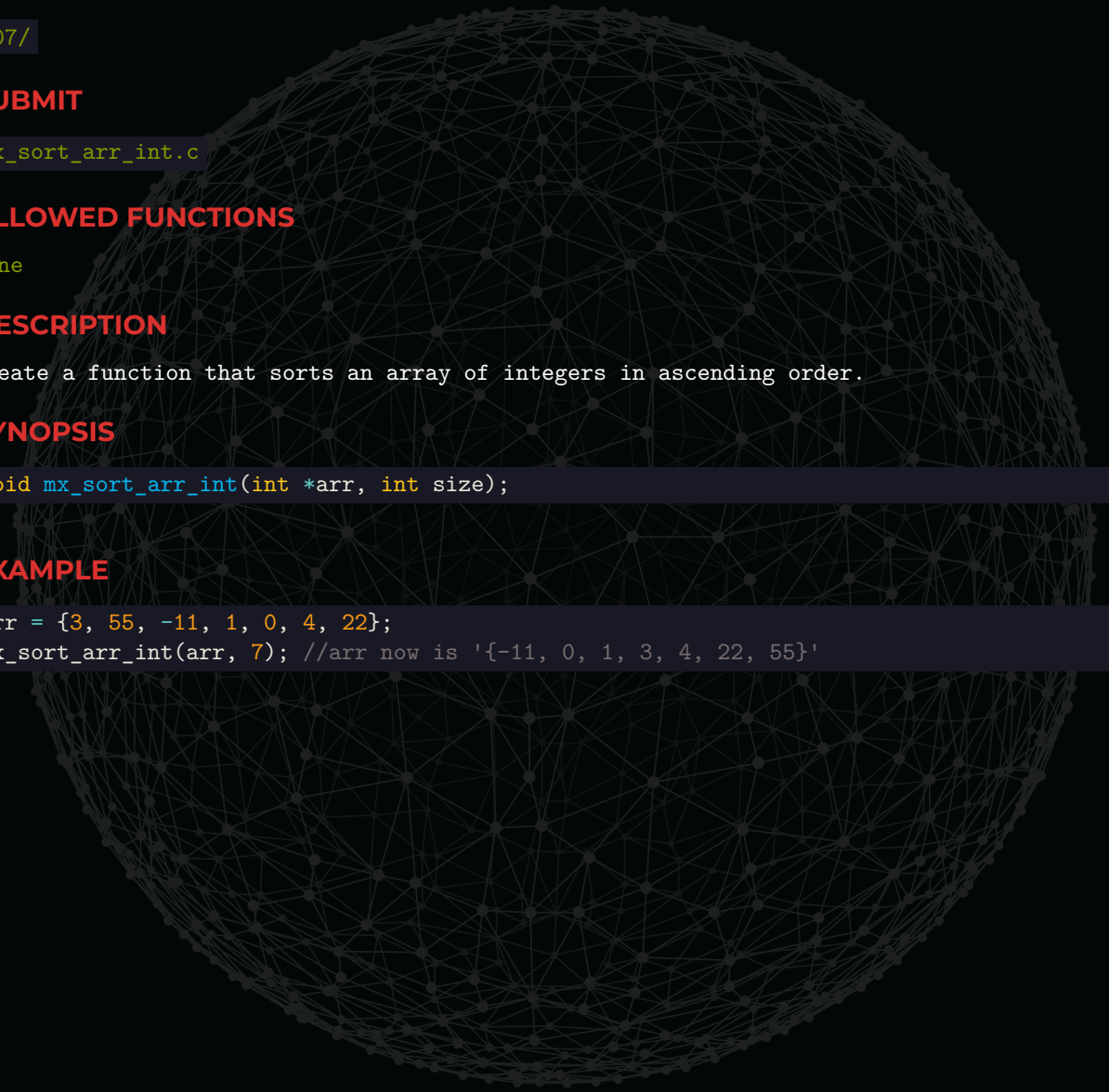
Create a function that sorts an array of integers in ascending order.

SYNOPSIS

```
void mx_sort_arr_int(int *arr, int size);
```

EXAMPLE

```
arr = {3, 55, -11, 1, 0, 4, 22};
mx_sort_arr_int(arr, 7); //arr now is '{-11, 0, 1, 3, 4, 22, 55}'
```



Act: Task 08

NAME

Compare strings

DIRECTORY

`t08/`

SUBMIT

`mx_strcmp.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

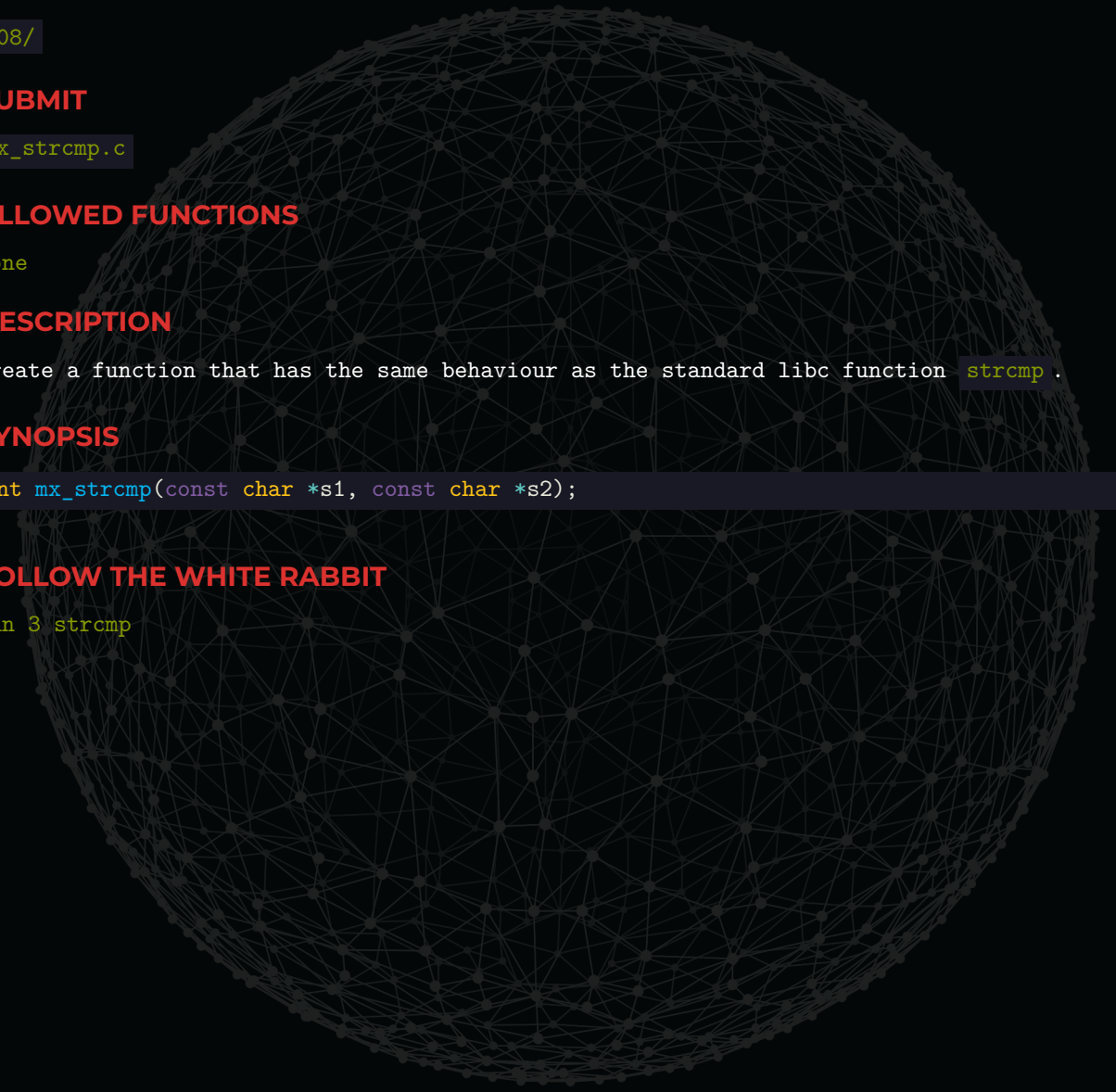
Create a function that has the same behaviour as the standard libc function `strcmp`.

SYNOPSIS

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

FOLLOW THE WHITE RABBIT

`man 3 strcmp`



Act: Task 09

NAME

Copy string

DIRECTORY

`t09/`

SUBMIT

`mx_strcpy.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

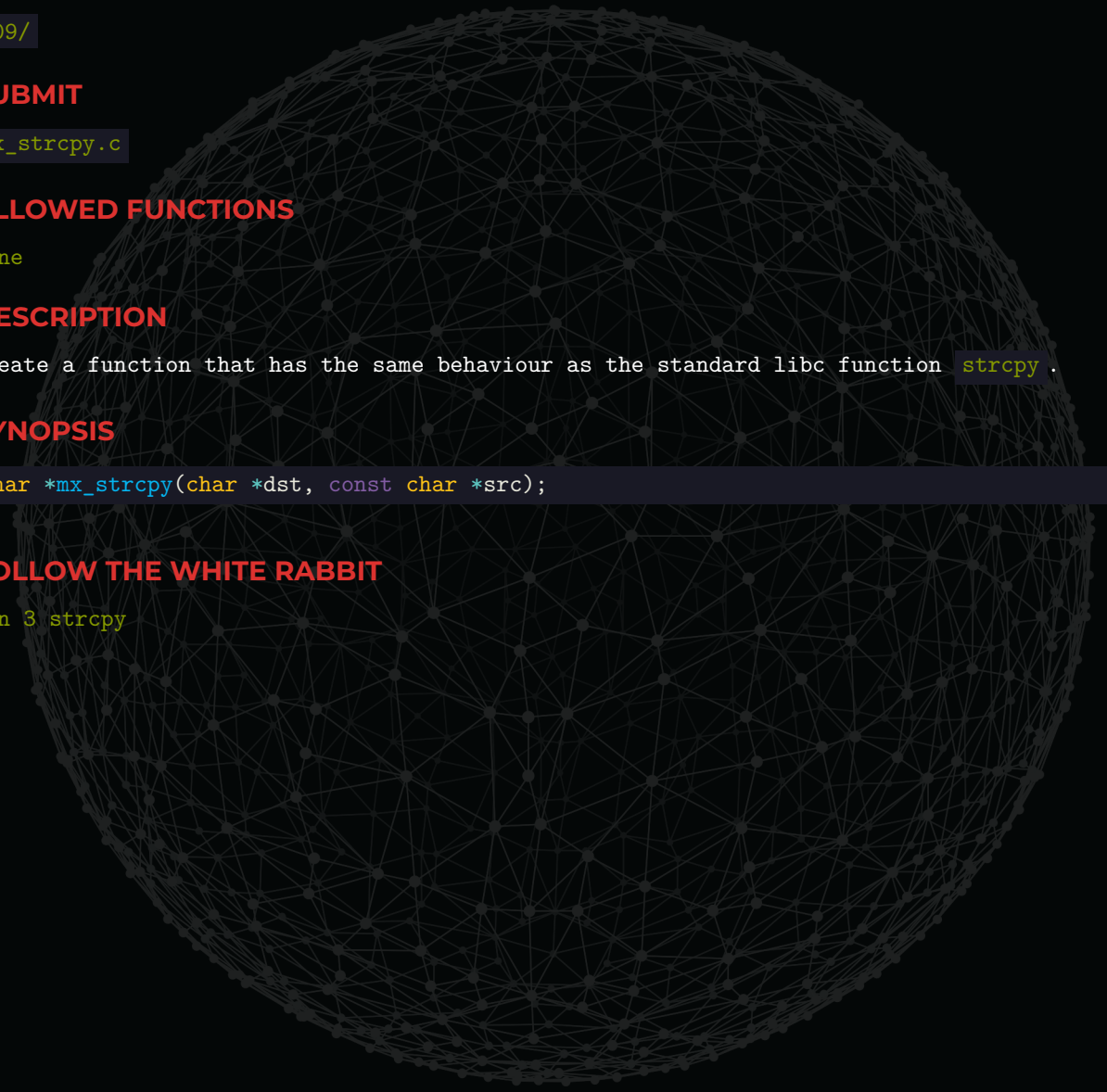
Create a function that has the same behaviour as the standard libc function `strcpy`.

SYNOPSIS

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

FOLLOW THE WHITE RABBIT

`man 3 strcpy`



Act: Task 10

NAME

Print arguments

DIRECTORY

`t10/`

SUBMIT

`mx_print_args.c, mx_printchar.c, mx_printstr.c, mx_strlen.c`

ALLOWED FUNCTIONS

`write`

DESCRIPTION

Create a program that:

- prints its arguments to standard output, excluding program name
- prints each argument followed by a newline
- does nothing if there are no command-line arguments

CONSOLE OUTPUT

```
>./mx_print_args Follow the white rabbit | cat -e
Follow$
the$
white$
rabbit$
>
```

Act: Task 11

NAME

Sort arguments

DIRECTORY

`t11/`

SUBMIT

`mx_print_sargs.c, mx_printchar.c, mx_printstr.c, mx_strlen.c, mx_strcmp.c`

ALLOWED FUNCTIONS

`write`

DESCRIPTION

Create a program that:

- sorts the arguments, excluding the name of the program, in ASCII order
- prints its arguments to standard output followed by a newline
- does nothing if there are no command-line arguments

CONSOLE OUTPUT

```
>./mx_print_sargs Follow the white rabbit | cat -e
Follow$
rabbit$
the$
white$
>
```

Act: Task 12

NAME

Iterative factorial

DIRECTORY

`t12/`

SUBMIT

`mx_factorial_iter.c`

ALLOWED FUNCTION

None

DESCRIPTION

Create a function that calculates the factorial of a `non-negative` integer using an `iterative` algorithm.
Hint: Case when the factorial of a given `n` bigger than MAX_INT - `error` case.

RETURN

- returns the factorial of the `non-negative` integer
- returns `0` in case of `errors`

SYNOPSIS

```c
int mx_factorial_iter(int n);
```

EXAMPLE

```c
mx_factorial_iter(2); //returns 2
mx_factorial_iter(5); //returns 120
```

Act: Task 13

NAME

Bubble sort

DIRECTORY

`t13/`

SUBMIT

`mx_bubble_sort.c, mx_strcmp.c`

ALLOWED FUNCTIONS

None

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
```

SEE ALSO

Bubble sort

Act: Task 14

NAME

Binary search

DIRECTORY

`t14/`

SUBMIT

`mx_binary_search.c, mx_strcmp.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that:

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

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

EXAMPLE

```c
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
```

SEE ALSO

Binary search

Act: Task 15

NAME

New string

DIRECTORY

`t15/`

SUBMIT

`mx_strnew.c`

ALLOWED FUNCTIONS

`malloc`

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

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

EXAMPLE

```c
mx_strnew(10); //returns string with size 10 and terminated by '\0'
mx_strnew(-1); //returns NULL
```

FOLLOW THE WHITE RABBIT

`man 3 malloc`

Act: Task 16

NAME

Duplicate string

DIRECTORY

`t16/`

SUBMIT

`mx_strdup.c, mx_strnew.c, mx_strlen.c, mx_strcpy.c`

ALLOWED FUNCTIONS

`malloc`

DESCRIPTION

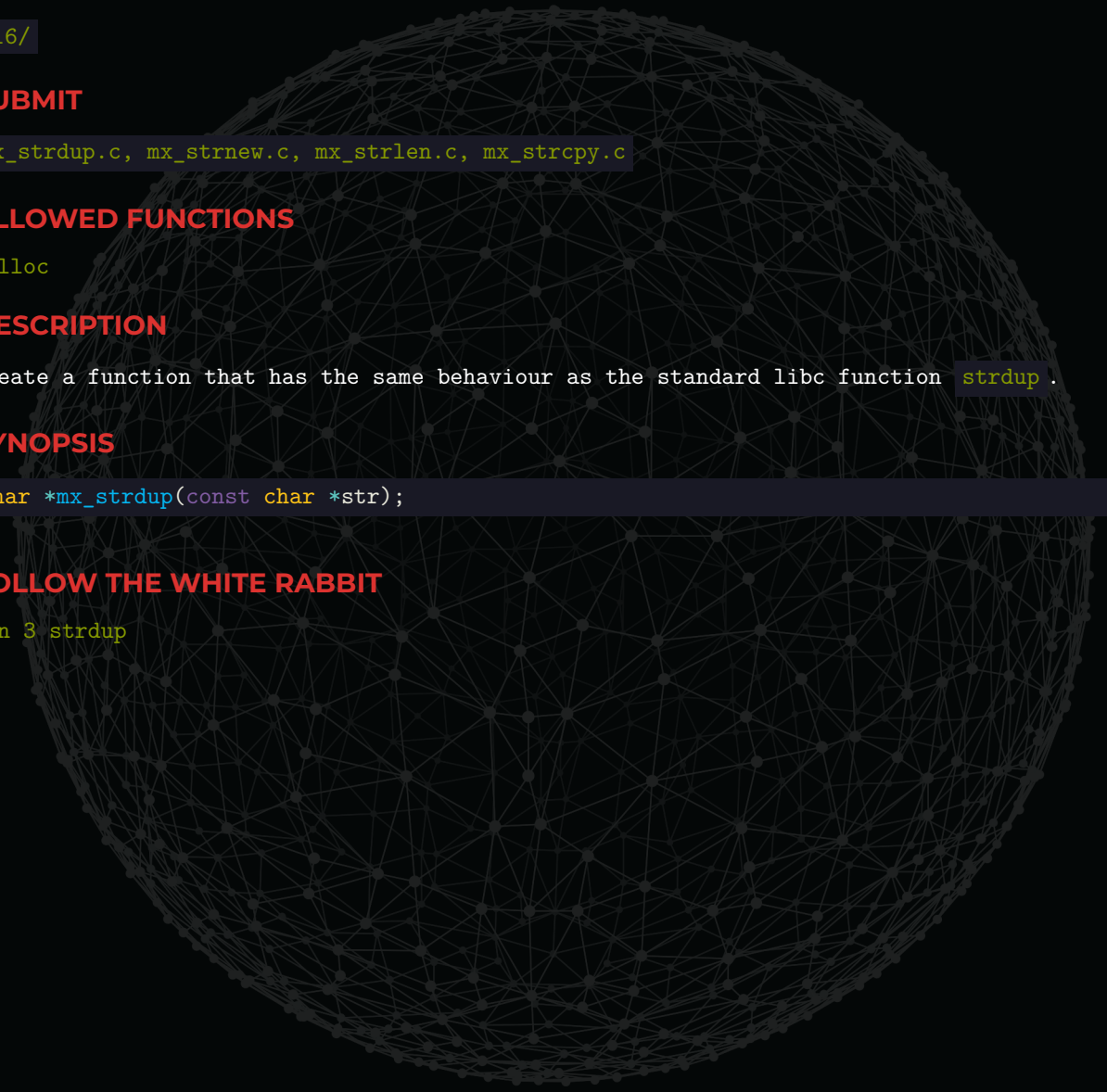
Create a function that has the same behaviour as the standard libc function `strdup`.

SYNOPSIS

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

FOLLOW THE WHITE RABBIT

`man 3 strdup`



Act: Task 17

NAME

Hex to decimal

DIRECTORY

`t17/`

SUBMIT

`mx_hex_to_nbr.c`

ALLOWED FUNCTIONS

None

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
```

Act: Task 18

NAME

For each

DIRECTORY

`t18/`

SUBMIT

`mx_foreach.c`

ALLOWED FUNCTIONS

None

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));
```

EXAMPLE

```
void mx_printint(int n);

arr = {1, 2, 3, 4, 5};
mx_foreach(arr, 5, mx_printint); //prints "12345" to the standart output
```

SEE ALSO

Function pointer

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.