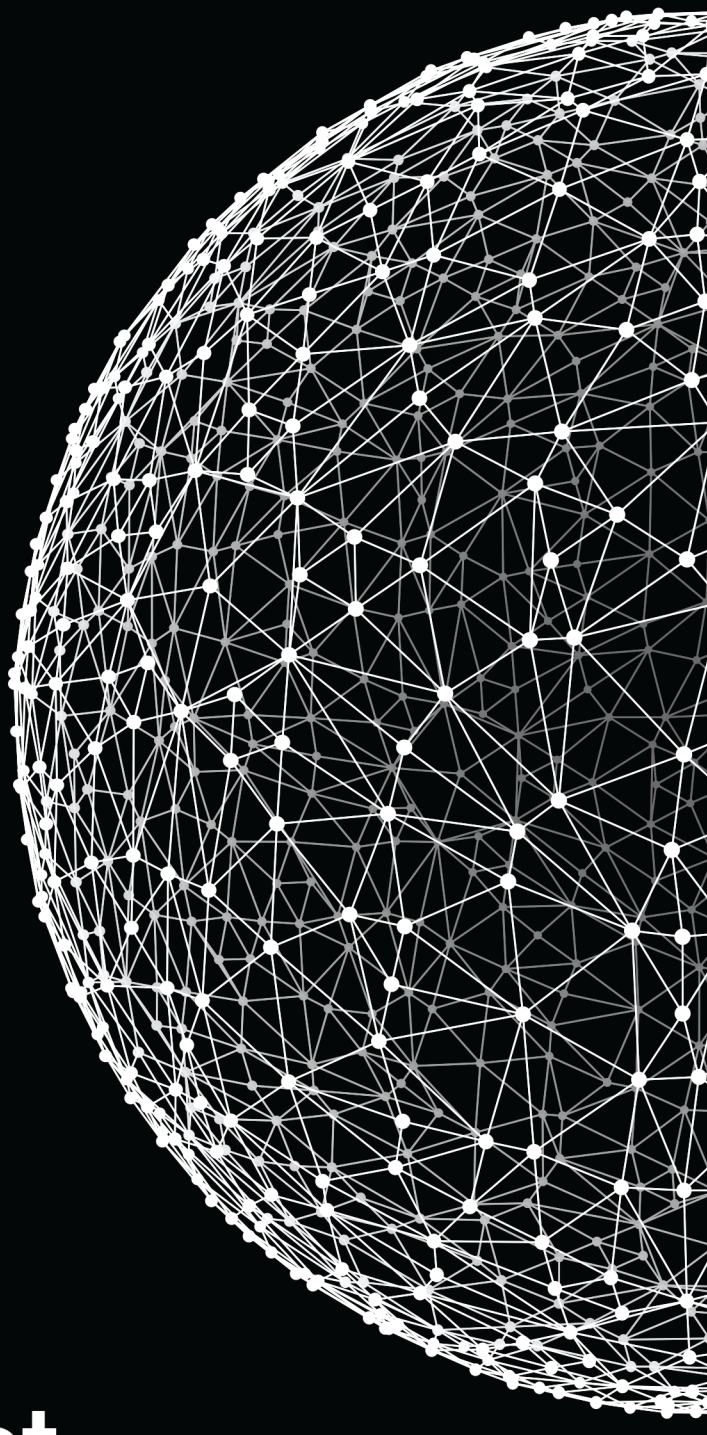
Sprint 08

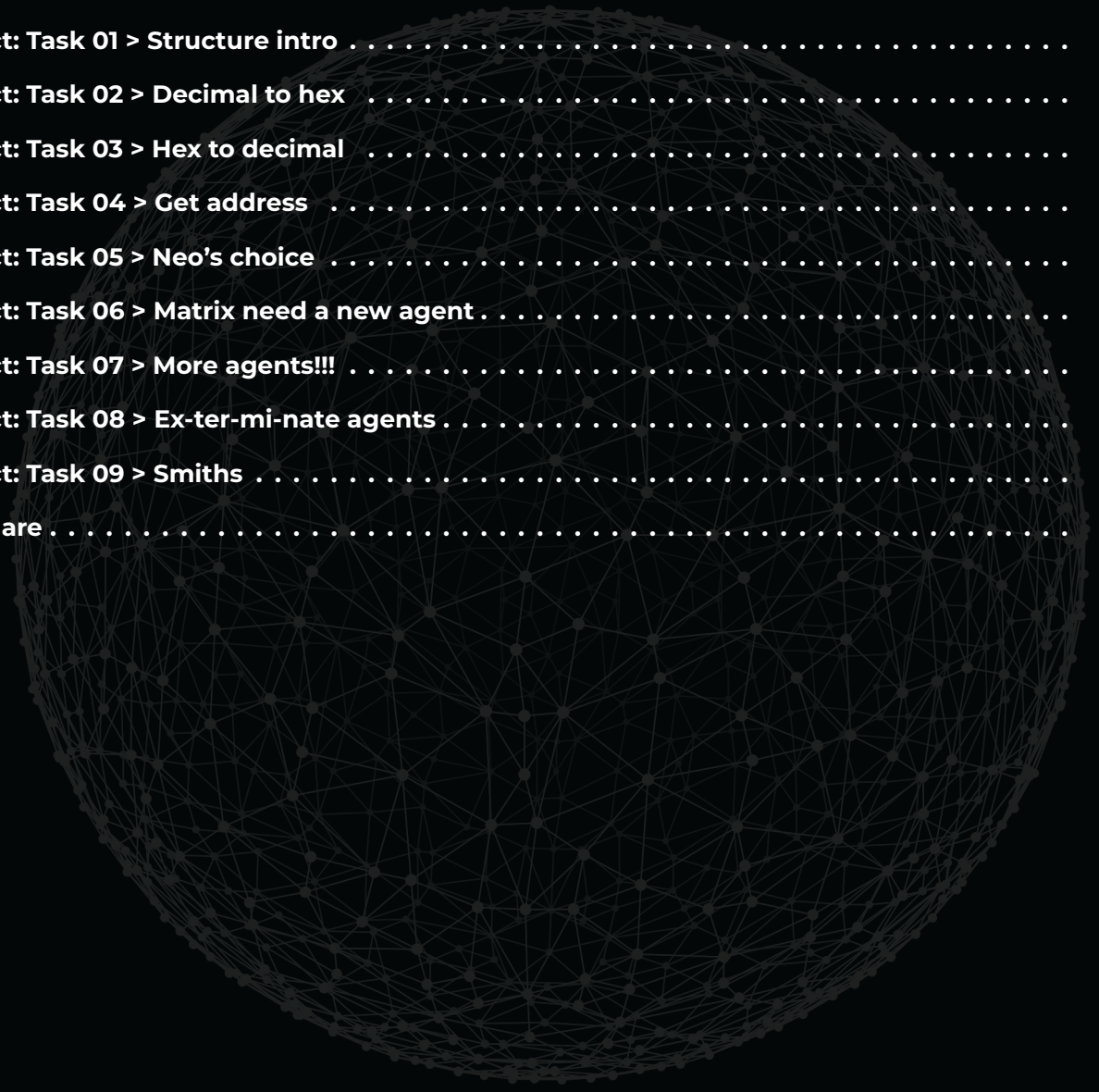
Marathon C

October 15, 2020



ucode connect

Contents



ucode connect

Engage

DESCRIPTION

Hello!

We hope that the previous Sprint helped you understand how memory works. And now this information has a place in your mind palace.

As you have already noticed, the more complex the task, the more knowledge it requires. The same applies to data manipulation. Standard data types are not enough to solve the problem effectively. The number of files, functions, and variables is becoming greater and greater. The number of different elements increases, and it becomes more difficult to manage them.
In this challenge you must deal with this issue.

This Sprint will help you find solutions to many issues you've already faced before. You will learn what are headers and structures.

BIG IDEA

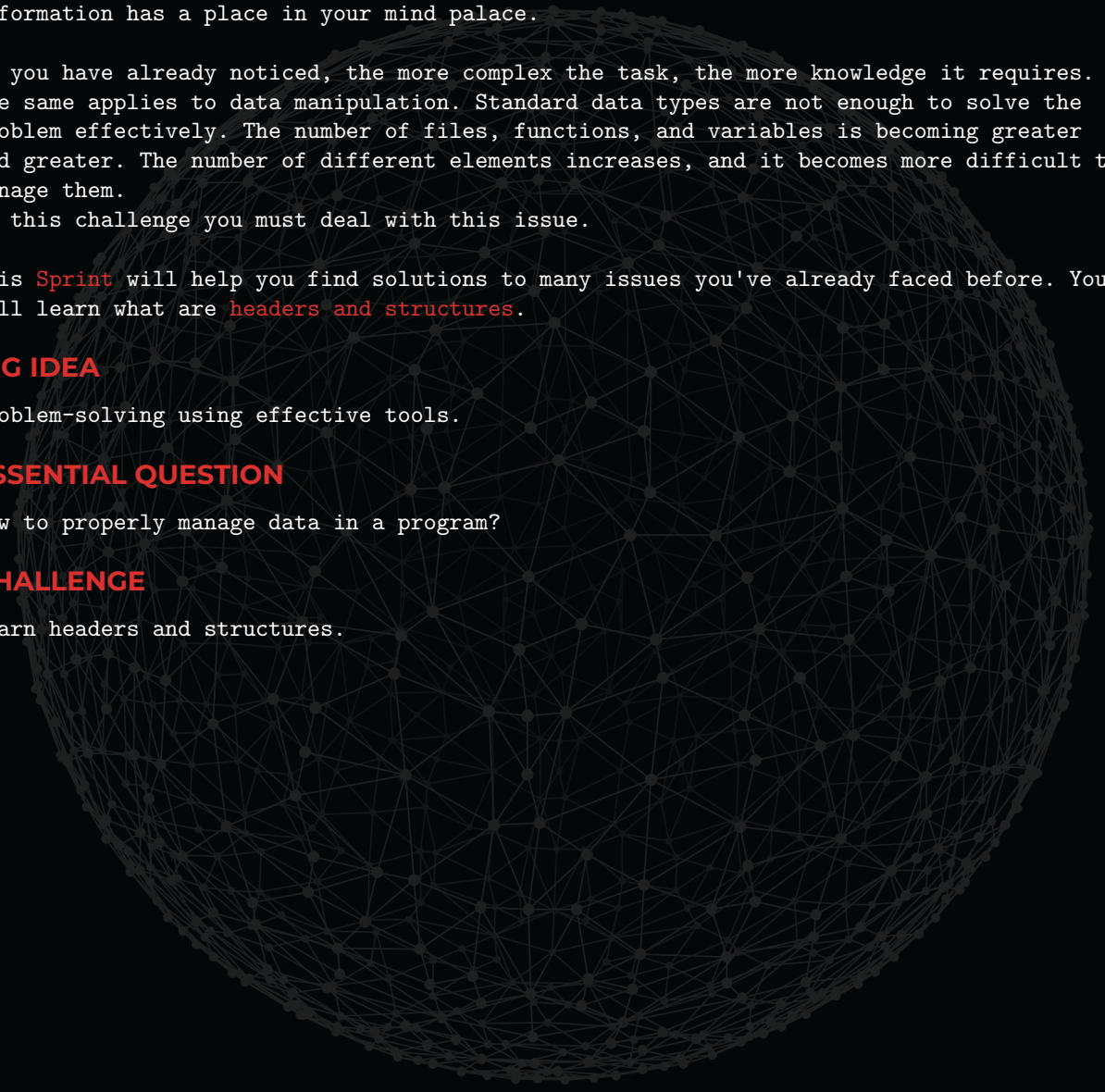
Problem-solving using effective tools.

ESSENTIAL QUESTION

How to properly manage data in a program?

CHALLENGE

Learn headers and structures.



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 data types exist in C?
  - How to create user defined data types in C?
  - What are the primary data types?
  - What are the derived data types?
  - What are arrays?
- What is a structure?
  - How to declare structure variables?
  - How to initialize structure members?
  - How to access structure elements?
  - What is designated initialization?
- What are headers?
  - How to include headers in a file?
  - How to use the created header file?
  - How to compile a program using a header file?
  - What are once-only headers? How to use `#pragma once`?
- What is a positional number system?
  - What types exist?
  - What is the base of a numeral system?
  - What is the `0x` prefix for hexadecimal numbers?

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.

- Repeat the basics from the previous challenges. Repeat everything you know and do not know about pointers. In this challenge they are also needed.
- Find information about the terms: `positional number system`, `header file`, `typedef`, `struct` in C.
- Read the tasks below.
- Clone your git repository that is issued on the challenge page in the LMS.
- Open the Auditor and find examples of correct use of header files and structures.
- Arrange to brainstorm tasks with other students.
- Try to implement your thoughts in code.

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

Header intro

DIRECTORY

`t00/`

SUBMIT

`header.h`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a header that contains prototypes of your library functions. Prototypes of the next functions are mandatory:

- `mx_printchar`
- `mx_printint`
- `mx_printstr`
- `mx_strcpy`
- `mx_strlen`
- `mx_strcmp`
- `mx_isdigit`
- `mx_isspace`
- `mx_atoi`

Oracle will test these functions by using your header.

The headers must also comply with the Auditor rules. Starting from this challenge:

- typedef/struct/union/enum must only be declared in header files unless specified otherwise
- function prototypes and macro definitions must be only in header files

SEE ALSO

Once-Only Headers
C - Header Files

Act: Task 01

NAME

Structure intro

DIRECTORY

`t01/`

SUBMIT

`duplicate.h, mx_del_dup_sarr.c, mx_copy_int_arr.c`

ALLOWED FUNCTIONS

`malloc, free`

DESCRIPTION

In this task you must:

- rewrite the function `mx_del_dup_arr` using a structure
- develop a function that creates a new structure with a new array without duplicates and its size
- create a header that includes all the necessary headers, prototypes and structure `s_intarr`

RETURN

- returns a new structure with a new array without duplicates and its size
- returns `NULL` if the structure does not exist or creation fails

SYNOPSIS

```
typedef struct s_intarr
{
    int *arr;
    int size;
}              t_intarr;
```

```
t_intarr *mx_del_dup_sarr(t_intarr *src);
```

SEE ALSO

C struct
C programming structure

Act: Task 02

NAME

Decimal to hex

DIRECTORY

`t02/`

SUBMIT

`nbr_to_hex.h, mx_nbr_to_hex.c, mx_strnew.c`

ALLOWED FUNCTIONS

`malloc`

DESCRIPTION

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

RETURN

- returns the number converted to a hexadecimal string
- returns `0` if the number is not valid

SYNOPSIS

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

EXAMPLE

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

Act: Task 03

NAME

Hex to decimal

DIRECTORY

`t03/`

SUBMIT

`hex_to_nbr.h, mx_hex_to_nbr.c, mx_isdigit.c, mx_isalpha.c, mx_islower.c, mx_isupper.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

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

RETURN

- returns the `unsigned long` number
- returns `0` if the hexadecimal string is not valid

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("fffffffffff"); //returns 281474976710655
```

Act: Task 04

NAME

Get address

DIRECTORY

`t04/`

SUBMIT

`get_address.h, mx_get_address.c, mx_nbr_to_hex.c, mx_strcpy.c, mx_strlen.c, mx_strnew.c`

ALLOWED FUNCTIONS

`malloc, free`

DESCRIPTION

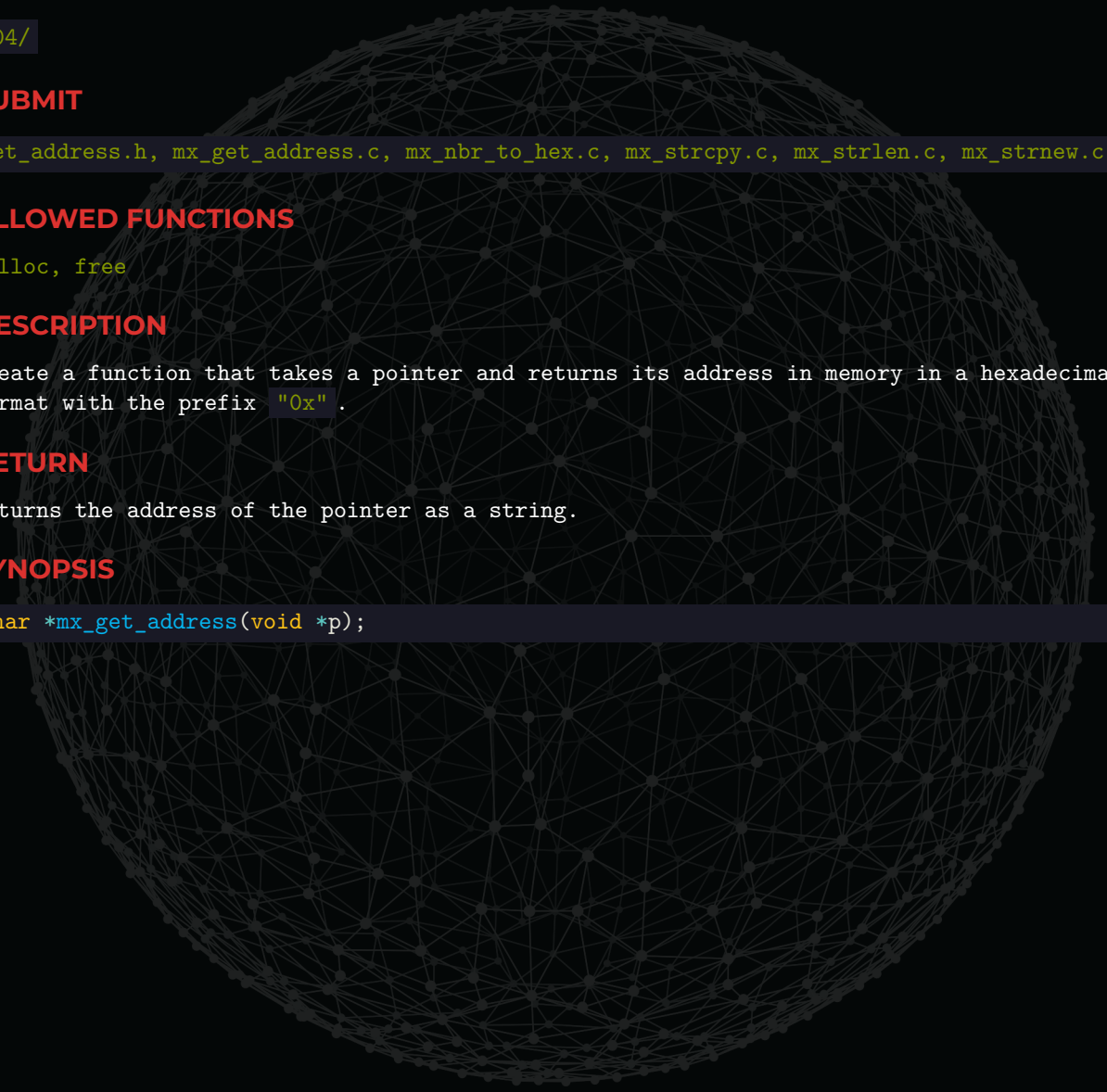
Create a function that takes a pointer and returns its address in memory in a hexadecimal format with the prefix `"0x"`.

RETURN

Returns the address of the pointer as a string.

SYNOPSIS

```c
char *mx_get_address(void *p);
```



Act: Task 05

NAME

Neo's choice

DIRECTORY

`t05/`

SUBMIT

`choice.h`

ALLOWED FUNCTIONS

None

DESCRIPTION

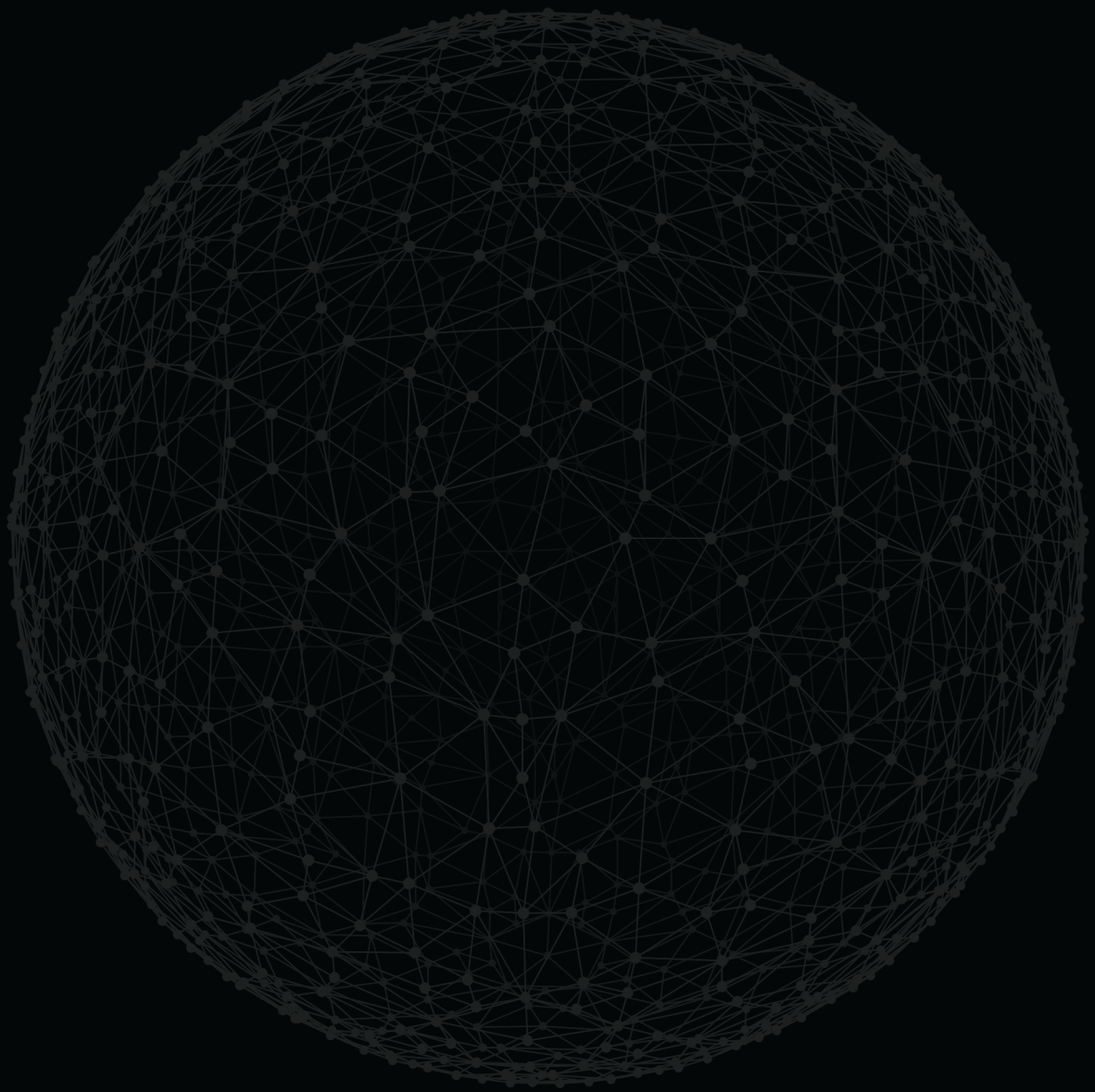
Create a header file `choice.h` with which the program compiles and works.
If Neo chooses:

- the red pill, the program prints `"Follow me!"`
- the blue pill, the program prints `"Perhaps I was wrong about you, Neo."`
- something else, the program prints `"Are you sure about that?"`

SYNOPSIS

```c
#include "choice.h"

t_phrase *choice(int pill) {
    char *res;

    if (pill == MX_RED_PILL) {
        res = mx_strdup(MX_SUCCESS_PHRASE);
    }
    else if (pill == MX_BLUE_PILL) {
        res = mx_strdup(MX_FAIL_PHRASE);
    }
    else {
        res = mx_strdup(MX_UNDEFINED_PHRASE);
    }
    return res;
}

int main(void) {
    t_phrase *phrase1 = choice(MX_RED_PILL);
    t_phrase *phrase2 = choice(MX_BLUE_PILL);
    t_phrase *phrase3 = choice((MX_RED_PILL + MX_BLUE_PILL) * 2);

    printf("%s\n", phrase1);
    printf("%s\n", phrase2);
```

```
    printf("%s\n", phrase3);
    return 0;
}
```



Act: Task 06

NAME

Matrix need a new agent

DIRECTORY

`t06/`

SUBMIT

`create_agent.h, mx_create_agent.c, mx_strdup.c, mx_strnew.c, mx_strlen.c, mx_strcpy.c`

ALLOWED FUNCTIONS

`malloc`

DESCRIPTION

The Matrix chose you. In this task you must:

- develop a new agent creator
- include `agent.h` in your header file to be able to use the structure `s_agent`. The Matrix will use your function with its own header `agent.h`
- create a function that allocates new (duplicate) memory for the `name` parameter

RETURN

- returns the `pointer` to the allocated memory of the new structure
- returns `NULL` if the `name` is `NULL` or agent creation fails

SYNOPSIS

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

```
t_agent *mx_create_agent(char *name, int power, int strength);
```

EXAMPLE

```
agent = mx_create_agent("Smith", 150, 66);
//agent->name is "Smith"
//agent->power is 150
//agent->strength is 66
```

Act: Task 07

NAME

More agents!!!

DIRECTORY

`t07/`

SUBMIT

`create_new_agents.h, mx_create_new_agents.c, mx_create_agent.c, mx_strdup.c, mx_strnew.c, mx_strlen.c, mx_strcpy.c`

ALLOWED FUNCTIONS

`malloc, free`

DESCRIPTION

In this task, you must follow the next items.

- Create a function that makes a `NULL-terminated` array of pointers to agents
- Data for each agent are stored in 3 different arrays: `name`, `power` and `strength`
- Each agent characteristic is placed with the same index in the respective array
- You must use the structure from `agent.h`

RETURN

- returns the `NULL`-terminated array of agents
- returns `NULL` if one of the parameters of the function is `NULL` or agent creation fails

SYNOPSIS

```
t_agent **mx_create_new_agents(char **name, int *power, int *strength, int count);
```

EXAMPLE

```
names = {"Thompson", "Smith", "Colson"};
powers = {33, 66, 99};
strengths = {133, 166, 196};
mx_create_new_agents(names, powers, strengths, 3); //returns 't_agent' type array
```

Act: Task 08

NAME

Ex-ter-mi-nate agents

DIRECTORY

`t08/`

SUBMIT

`exterminate_agents.h, mx_exterminate_agents.c`

ALLOWED FUNCTIONS

`free`

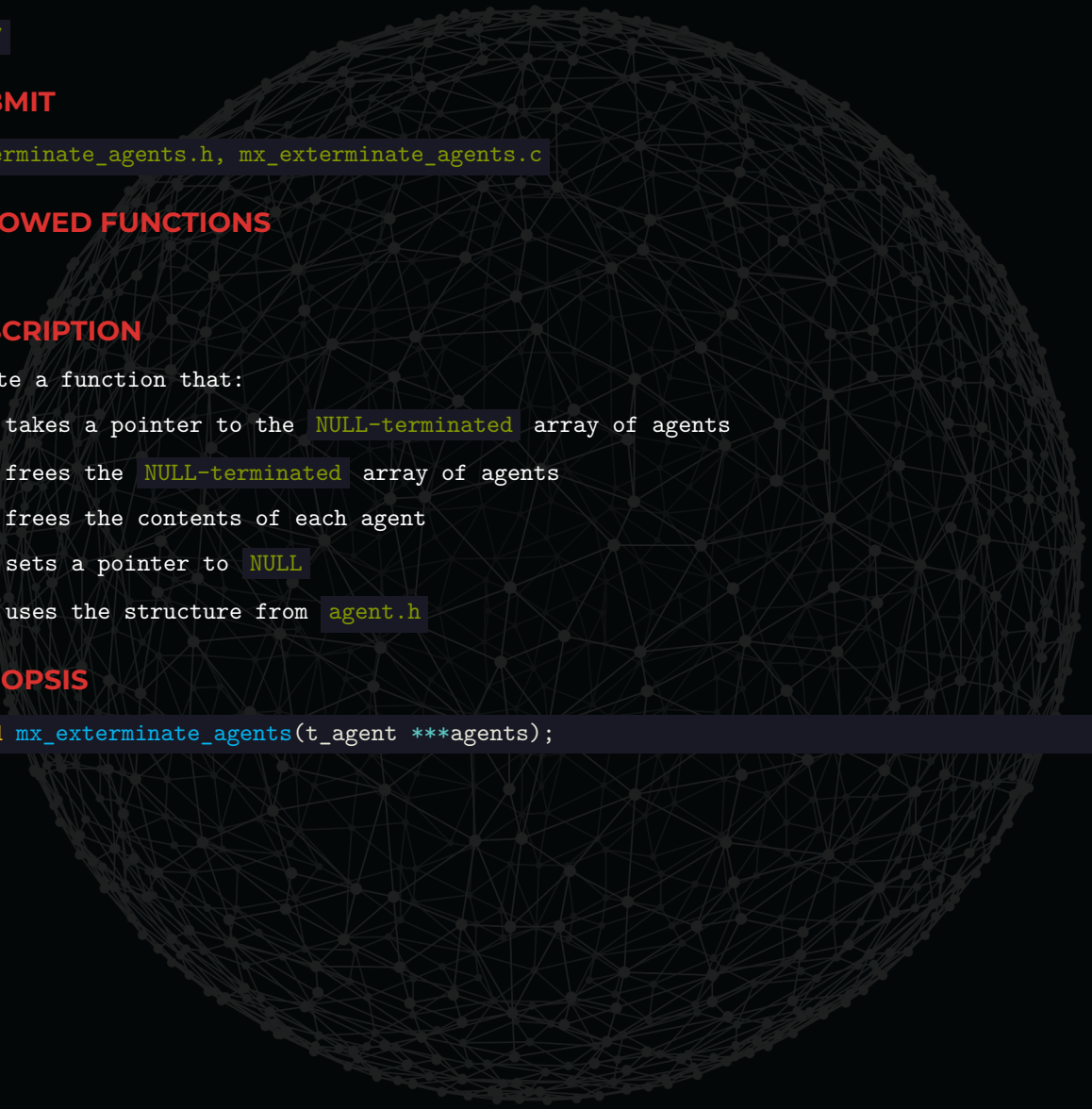
DESCRIPTION

Create a function that:

- takes a pointer to the `NULL-terminated` array of agents
- frees the `NULL-terminated` array of agents
- frees the contents of each agent
- sets a pointer to `NULL`
- uses the structure from `agent.h`

SYNOPSIS

```
void mx_exterminate_agents(t_agent ***agents);
```



Act: Task 09

NAME

Smiths

DIRECTORY

`t09/`

SUBMIT

`only_smiths.h, mx_only_smiths.c, mx_strcmp.c, mx_exterminate_agents.c, mx_create_agent.c, mx_strdup.c, mx_strnew.c, mx_strlen.c, mx_strcpy.c`

ALLOWED FUNCTIONS

`malloc, free`

DESCRIPTION

In this task, you must follow the next directions.

- Create a function that creates a new `NULL-terminated` array of pointers to agents
- The new array has only agents with the name `Smith` and a strength lower than the `strength` parameter of the function
- Input `agents` must be exterminated
- You must use the structure from `agent.h`

RETURN

- returns the new filtered array
- returns `NULL` if the original array is `NULL` or the new array creation fails

SYNOPSIS

```
t_agent **mx_only_smiths(t_agent **agents, int strength);
```

EXAMPLE

```
agents[0] = mx_create_agent("Smith", 150, 166);
agents[1] = mx_create_agent("Brown", 147, 57);
agents[2] = mx_create_agent("Smith", 151, 65);
agents[3] = mx_create_agent("Smith", 123, 321);
agents[4] = NULL;
mx_only_smiths(agents, 100); //returns array with 1 element
```

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.