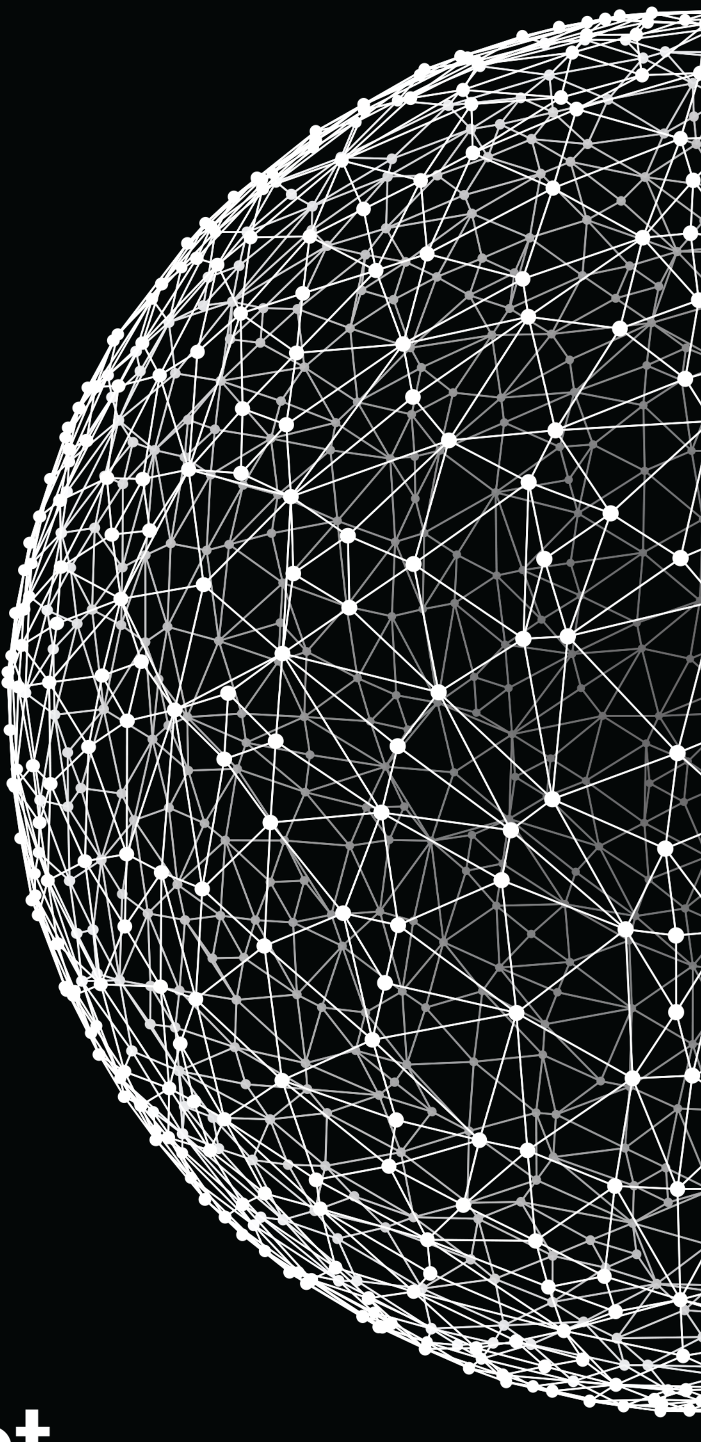
Checkpoint 01

Marathon C

September 3, 2020



ucode connect

Contents

ucode connect

Challenge Based Learning

- The Checkpoint is a reflection of what you've learned during your time at ucode connect so far. It is a tool for you to evaluate your understanding of the material, and for us to make sure all students are on the right track.
- You will help yourself by completing this task honestly. This challenge will help you understand whether you are moving in the right direction. You have an opportunity to identify areas that you know well, and those that you need to spend more time on. Also, try to reflect on whether you have used the possibilities of P2P learning to the max, and think about how to improve in that area.
- During Checkpoint you can not speak with other students, you do not have a right to chat, listen to music, make noise, or generally do anything that may disturb the other students in any way.
- Your phones and other technological devices must be turned off and put away. If a phone rings, you will be disqualified from the Checkpoint and kicked out immediately.
- Do not use the internet. Complete the Checkpoint on your own to get the best result.
- To achieve the best result for yourself, you must have only a piece of paper and a pen.
- Only in the case of a technical issue, problem understanding the story, or other questions related to the Checkpoint, you can write to the channel #___ucode_connect_team (but don't ask how to solve a task, that's on you). The ucode connect team will quickly respond to your request.
- 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!
- Read `man` to carry out some tasks if you need it.
- Compile C-files with clang compiler and use these flags: `-std=c11 -Wall -Wextra -Werror -Wpedantic`.
- Pay attention to what is allowed in a certain task. You may recode any other function you think is necessary. Use of forbidden stuff is considered a cheat and your tasks will be failed.
- The evaluation is fully automated. Tasks will pass automatic evaluation which is called `Oracle`.
- An important part of your study is an analysis of mistakes to avoid them in the future. To help you to revise your mistakes, the read-only access to your Checkpoint repository will stay open even after the Checkpoint is finished. You can find the link to the repository at the challenge page in the LMS.
  Use this opportunity wisely. Try completing the tasks that you've failed during the Checkpoint.

Act: Task 00

NAME

Print error

DIRECTORY

`t00/`

SUBMIT

`mx_printerr.c`

ALLOWED FUNCTIONS

write

DESCRIPTION

Create a function that outputs a string of characters to the standard error stream `stderr` .

SYNOPSIS

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

FOLLOW THE WHITE RABBIT

man 2 write
man stderr

Act: Task 01

NAME

Print arguments

DIRECTORY

`t01/`

SUBMIT

`mx_print_args.c`

ALLOWED FUNCTIONS

`write`

DESCRIPTION

Create a program that:

- prints its arguments to the 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 02

NAME

New string

DIRECTORY

`t02/`

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

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

EXAMPLE

```
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 03

NAME

Duplicate string

DIRECTORY

`t03/`

SUBMIT

`mx_strdup.c`

ALLOWED FUNCTIONS

malloc

DESCRIPTION

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

SYNOPSIS

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

FOLLOW THE WHITE RABBIT

man 3 strdup

Act: Task 04

NAME

Print program name

DIRECTORY

`t04/`

SUBMIT

`mx_print_name.c`

ALLOWED FUNCTIONS

write

DESCRIPTION

Create a program that prints to the standard output:

- its name and argument count
- both followed by a newline

CONSOLE OUTPUT

```
>./mx_print_name Follow the white rabbit | cat -e
./mx_print_name$
5$
>
```

Act: Task 05

NAME

For each

DIRECTORY

`t05/`

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(const 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
```

Act: Task 06

NAME

Decimal to hex

DIRECTORY

`t06/`

SUBMIT

`mx_nbr_to_hex.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.

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 07

NAME

Sort with comparator

DIRECTORY

`t07/`

SUBMIT

`mx_sort.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that sorts an array of integers in place in the order defined by the function `f` .

SYNOPSIS

```
void mx_sort(int *arr, int size, bool (*f)(int, int));
```

SYNOPSIS

```
bool compare(int a, int b) {
    return a > b;
}

arr = {5, 4, 3, 2, 1};
mx_sort(arr, 5, compare); //array has become '{1, 2, 3, 4, 5}'
```