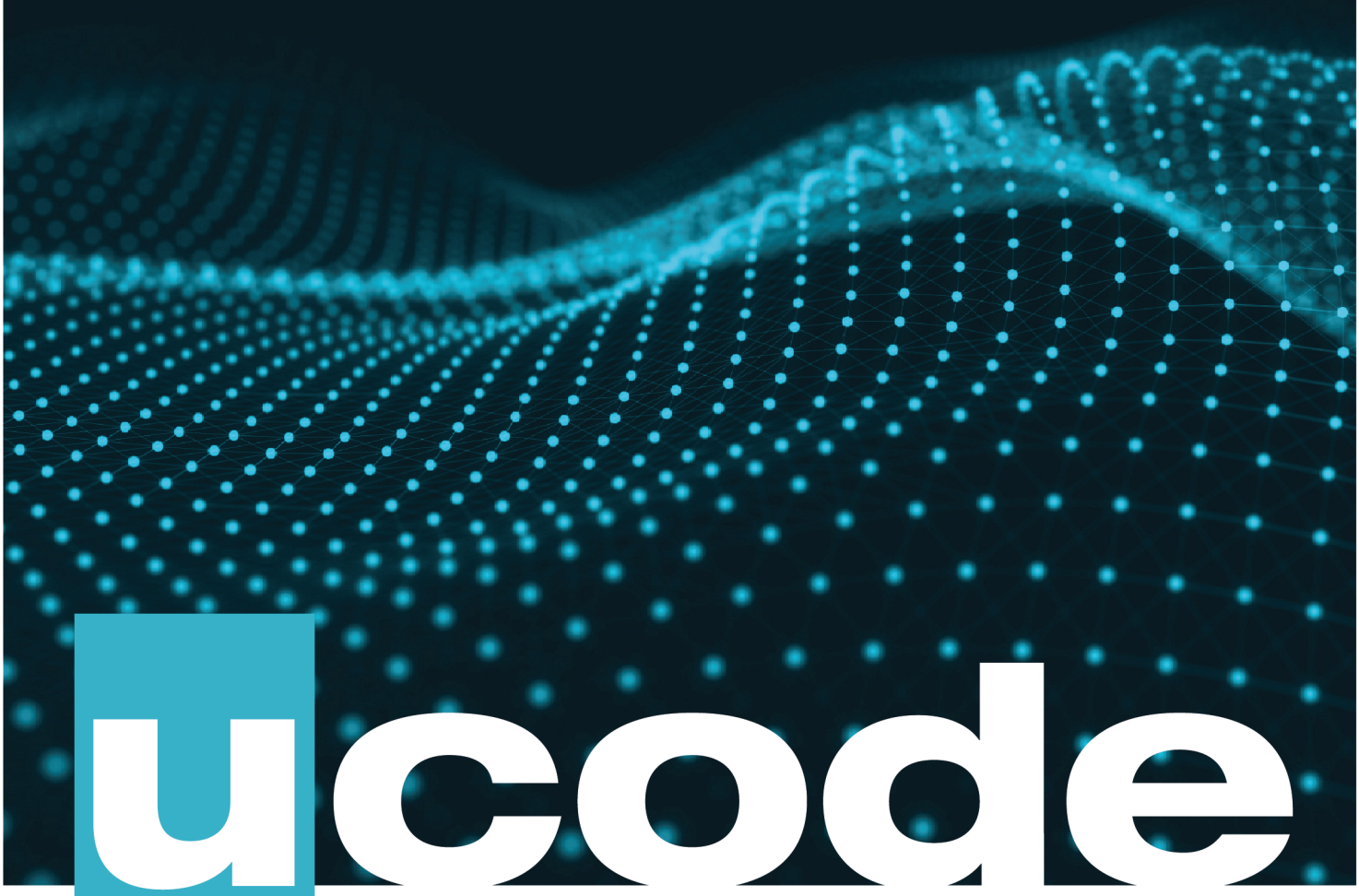
Sprint 05

Marathon C

October 15, 2019



ucode

Contents

ucode

Engage

DESCRIPTION

Hey, hey, dear! Let's go!

Gratz!
Second week started!
But we know, your way just started. So, prepare your mind. We go even further.

This sprint is designed to study simple algorithms and the development of algorithmic thinking. You will learn what program arguments are and code some simple mathematical formulas.

Hope you're ready, coz we are! :)

BIG IDEA

Develop algorithmic thinking.

ESSENTIAL QUESTION

How to implement simple math formulas in `C`?

CHALLENGE

Code simple algorithms.

Investigate

GUIDING QUESTION

We invite you to find answers to the following questions. This will help you realize what knowledge you will get from this challenge and how to move forward.

Ask your neighbor on the right, left, or behind you, and discuss the following questions together. You can find the answers in the Internet and share it with student around you.

We encourage you to ask as many questions about `C` programming as possible. Note down your discussion.

- How was your last week? How long did you sleep?)
- What new knowledge have you gained in this time in ucode?
- How was your weekend? How are you doing with races? What did and what not?
- What do you know about UNIX? What commands do you know?
- What did you learn about pointers?
- What is an array of pointers?
- What kinds of errors in `C` do you know?
- What is factorial? What is GCD and LCM?
- What is recursion?

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. Start from scratch, from the basics of last week. "Play" with the terminal, `cd -`, `ls -latr`, `mkdir`, `touch`, `cat -be` :)
2. Create a simple program with the use of arrays of pointers. Try to do the most difficult tasks from Sprints 03 and 04.
3. Read about the program arguments. Code a simple program that use arguments.
4. Create a program that displays each new argument with the next line.
5. Do the `task00`.

ANALYSIS

You need to analyze all the collected information before you start.

- 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 the story.
- You should submit only the specified files in the required directory and nothing else. In case you are allowed to submit any files you should submit only files that you used to complete a task. Garbage shall not pass.
- You should compile C files with clang compiler and use these flags: `-std=c11 -Wall -Wextra -Werror -Wpedantic`.
- You should use only functions which allowed in a certain task.
- Usage of forbidden functions is considered as cheat and your challenge will be failed.
- You must complete tasks according to the rules specified in `the Auditor`.
- Your exercises will be checked and evaluated by students. The same as you. `Peer-to-Peer (P2P) learning`.
- Also, your exercises will pass automatic evaluation which is called `Oracle`.
- Got a question or you do not understand something? Ask the students or just Google that.
- Google every new word you have not heard before.
- Use your brain and follow the white rabbit to prove that you are the Chosen one!!!

Act

SOLUTION DEVELOPMENT

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

1. Clone your git repository.
2. Only what you pushed into your repository will be evaluated.
3. Communicate and brainstorm with other students.
4. You have 40 hours to overcome this challenge.

Task 00

NAME

Print program name

DIRECTORY

`t00/`

SUBMIT

`mx_print_name.c, mx_printchar.c, mx_printstr.c, mx_printint.c, mx_strlen.c`

ALLOWED FUNCTION

`write`

DESCRIPTION

Create a program that prints to 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$
>
```

SEE ALSO

Command line arguments C

Task 01

NAME

Print arguments

DIRECTORY

`t01/`

SUBMIT

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

ALLOWED FUNCTION

`write`

DESCRIPTION

Create a program that:

- prints its arguments to standard output excluding program name;
- prints each argument followed by the 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$
>
```

Task 02

NAME

Sort arguments

DIRECTORY

`t02/`

SUBMIT

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

ALLOWED FUNCTION

`write`

DESCRIPTION

Create a program that:

- sorts the arguments, excluding the name of the program, in the order of ASCII;
- prints its arguments to standard output followed by the 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$
>
```

Task 03

NAME

Sum arguments

DIRECTORY

`t03/`

SUBMIT

`mx_sum_args.c, mx_printchar.c, mx_printint.c, mx_atoi.c, mx_isspace.c, mx_isdigit.c`

ALLOWED FUNCTION

`write`

DESCRIPTION

Create a program that:

- sums integer arguments and prints the sum on the standard output followed by the newline
- skips argument if it is not valid integer. Integers with single `-` and `+` signs before number are considered as valid arguments.
- does nothing if there are no command-line arguments.

CONSOLE OUTPUT

```
>./mx_sum_args 1- -7 | cat -e
-7$
>./mx_sum_args a1 b 2 c-3 | cat -e
2$
>./mx_sum_args 1 "   2" "3" "4  " | cat -e
4$
>./mx_sum_args 1 +2 -3 +-4 5+ 6at "  7" | cat -e
0$
>
```

Task 04

NAME

Print exact program name

DIRECTORY

`t04/`

SUBMIT

`mx_print_pname.c, mx_printchar.c, mx_printstr.c, mx_strchr.c, mx_strlen.c`

ALLOWED FUNCTION

`write`

DESCRIPTION

Create a program that prints its name excluding leading characters to standard output followed by the newline.

CONSOLE OUTPUT

```
>./mx_print_pname Follow the white rabbit | cat -e
mx_print_pname$
>
>/Users/root/marathonc/sprint05/t04/mx_print_pname | cat -e
mx_print_pname$
>
```

Task 05

NAME

Integer to binary

DIRECTORY

`t05/`

SUBMIT

`mx_print_argbints.c, mx_printchar.c, mx_printint.c, mx_atoi.c, mx_isspace.c, mx_isdigit.c`

ALLOWED FUNCTION

`write`

DESCRIPTION

Create a program that:

- prints a binary representation of each integer received as a command-line argument;
- prints each binary to standard output followed by the newline;
- does nothing if there are no command-line arguments.

P.S.
You will get well formatted integers as arguments.

CONSOLE OUTPUT

```
>./mx_print_argbints 2 "  -2" 2147483647 -2147483648 0 1 | cat -e
00000000000000000000000000000010$
11111111111111111111111111111110$
01111111111111111111111111111111$
10000000000000000000000000000000$
00000000000000000000000000000000$
00000000000000000000000000000001$
>
```

Task 06

NAME

Iterative factorial

DIRECTORY

`t06/`

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.

P.S.
Case when factorial of given `n` bigger than MAX_INT - `error` case.

RETURN

- return factorial of a `non-negative` integer;
- in case of `errors` function returns `0`.

SYNOPSIS

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

EXAMPLE

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

Task 07

NAME

Recursive factorial

DIRECTORY

`t07/`

SUBMIT

`mx_factorial_rec.c`

ALLOWED FUNCTION

None

DESCRIPTION

Create a function that calculates the factorial of a `non-negative` integer using `recursion`.

RETURN

- return factorial of a `non-negative` integer;
- in case of `errors` function returns `0`.

SYNOPSIS

```
int mx_factorial_rec(int n);
```

EXAMPLE

```
mx_factorial_rec(2); //returns 2
mx_factorial_rec(5); //returns 120
```

SEE ALSO

Recursion

Task 08

NAME

Recursive exponentiation

DIRECTORY

`t08/`

SUBMIT

`mx_pow_rec.c`

ALLOWED FUNCTION

None

DESCRIPTION

Create a function which will compute `n` raised to the power of positive integer `pow` using `recursion`.

RETURN

Returns `n` raised to the power of positive integer `pow`.

SYNOPSIS

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

EXAMPLE

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

FOLLOW THE WHITE RABBIT

`man pow`

SEE ALSO

Recursion
Exponentiation

Task 09

NAME

Multiplication table

DIRECTORY

`t09/`

SUBMIT

`mx_mult_table.c, mx_printchar.c, mx_printint.c, mx_atoi.c, mx_isdigit.c, mx_isspace.c, mx_strlen.c`

ALLOWED FUNCTION

`write`

DESCRIPTION

Create a program that:

- prints to standard output a table of multiplication of positive integers in the range specified as the command-line arguments which are `digits`;
- use the tab character `\t` as the delimiter when displaying the results;
- prints each table row followed by the newline;
- does nothing if number of command-line arguments is not equal to 2 or arguments are invalid.

CONSOLE OUTPUT

```
>./mx_mult_table 1 4 | cat -e
1       2       3       4$
2       4       6       8$
3       6       9       12$
4       8       12      16$
>./mx_mult_table 4 4 | cat -e
16$
>./mx_mult_table 4 2 | cat -e
4       6       8$
6       9       12$
8       12      16$
>./mx_mult_table 3 12 | cat -e
>
```

Task 10

NAME

Greatest common divisor

DIRECTORY

`t10/`

SUBMIT

`mx_gcd.c`

ALLOWED FUNCTION

None

DESCRIPTION

Create a `recursive` function that compute the greatest common divisor of two integers.

RETURN

Returns the greatest common divisor of two integers.

SYNOPSIS

```
int mx_gcd(int a, int b);
```

EXAMPLE

```
mx_gcd(20, 15); //returns 5
mx_gcd(-20, -15); //returns 5
```

SEE ALSO

Greatest common divisor

Task 11

NAME

Least common multiple

DIRECTORY

`t11/`

SUBMIT

`mx_lcm.c, mx_gcd.c`

ALLOWED FUNCTION

None

DESCRIPTION

Create a function that compute the least common multiple (LCM) of two integers.

RETURN

- returns the least common multiple of two integers;
- in case of errors function returns `0`.

SYNOPSIS

```
int mx_lcm(int a, int b);
```

EXAMPLE

```
mx_lcm(20, 15); //returns 60
mx_lcm(-20, 15); //returns 60
```

SEE ALSO

Least common multiple

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.