# Sprint 05

Marathon C

April 15, 2020

ucode

# Contents



ucode

# Engage

## DESCRIPTION

Hey, hey, dear! Let's go!

Congrats on completing the first week of the Marathon C!
However, your journey still continues. So, prepare your mind for more knowledge.

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, because 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. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- How did you like the first week of the marathon? How much did you sleep?
- What knowledge have you gained during this time in ucode?
- How did you spend your weekend? How are you doing with the Races? What did you accomplish?
- 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 a factorial? What is a GCD and a LCM?
- What is a recursion?

## 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 from scratch, from the basics of last week. "Play" with the Terminal, `cd -`, `ls -latr`, `mkdir`, `touch`, `cat -be`.
- Create a simple program with the use of arrays of pointers. Try to do the most difficult tasks from Sprint 03 and Sprint 04 again.
- Read about program arguments. Code a simple program that uses arguments.
- Create a program that displays each new argument followed by a newline.
- Clone your git repository that is issued on the challenge page in the LMS.
- Communicate with students and share information.
- Let's do the `task00`.

## 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!

# Act: 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

# Act: 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 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

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 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 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 the integer arguments and prints the sum to standard output followed by a newline
- skips the argument if it is not a valid integer. Integers with a single `-` or `+` signs before the number are considered as valid arguments
- outputs `0` if all arguments are invalid
- 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" "10  " | cat -e
4$
>./mx_sum_args 1 +2 -3 +-4 5+ 6at "  7" | cat -e
0$
>./mx_sum_args a1 2- | cat -e
0$
>
```

# Act: 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 the leading characters to standard output, followed by a 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$
>
```

# Act: 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 a newline
- does nothing if there are no command-line arguments

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$
>
```

# Act: 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.
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

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

## EXAMPLE

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

# Act: 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

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

## SYNOPSIS

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

## EXAMPLE

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

## SEE ALSO

Recursion

# Act: Task 08

## NAME

Recursive exponentiation

## DIRECTORY

`t08/`

## SUBMIT

`mx_pow_rec.c`

## ALLOWED FUNCTION

None

## DESCRIPTION

Create a function that computes `n` raised to the power of a positive integer `pow` using `recursion`.

## RETURN

Returns `n` raised to the power of the 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

# Act: 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 a table of multiplication of positive integers to standard output in the range specified as command-line arguments which are `digits`
- uses a tab character `\t` as a delimiter when displaying the results
- prints each table row followed by a newline
- does nothing if the 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
>
```

# Act: Task 10

## NAME

Greatest common divisor

## DIRECTORY

`t10/`

## SUBMIT

`mx_gcd.c`

## ALLOWED FUNCTION

None

## DESCRIPTION

Create a `recursive` function that computes 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

# Act: Task 11

## NAME

Least common multiple

## DIRECTORY

`t11/`

## SUBMIT

`mx_lcm.c, mx_gcd.c`

## ALLOWED FUNCTION

None

## DESCRIPTION

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

## RETURN

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

## 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