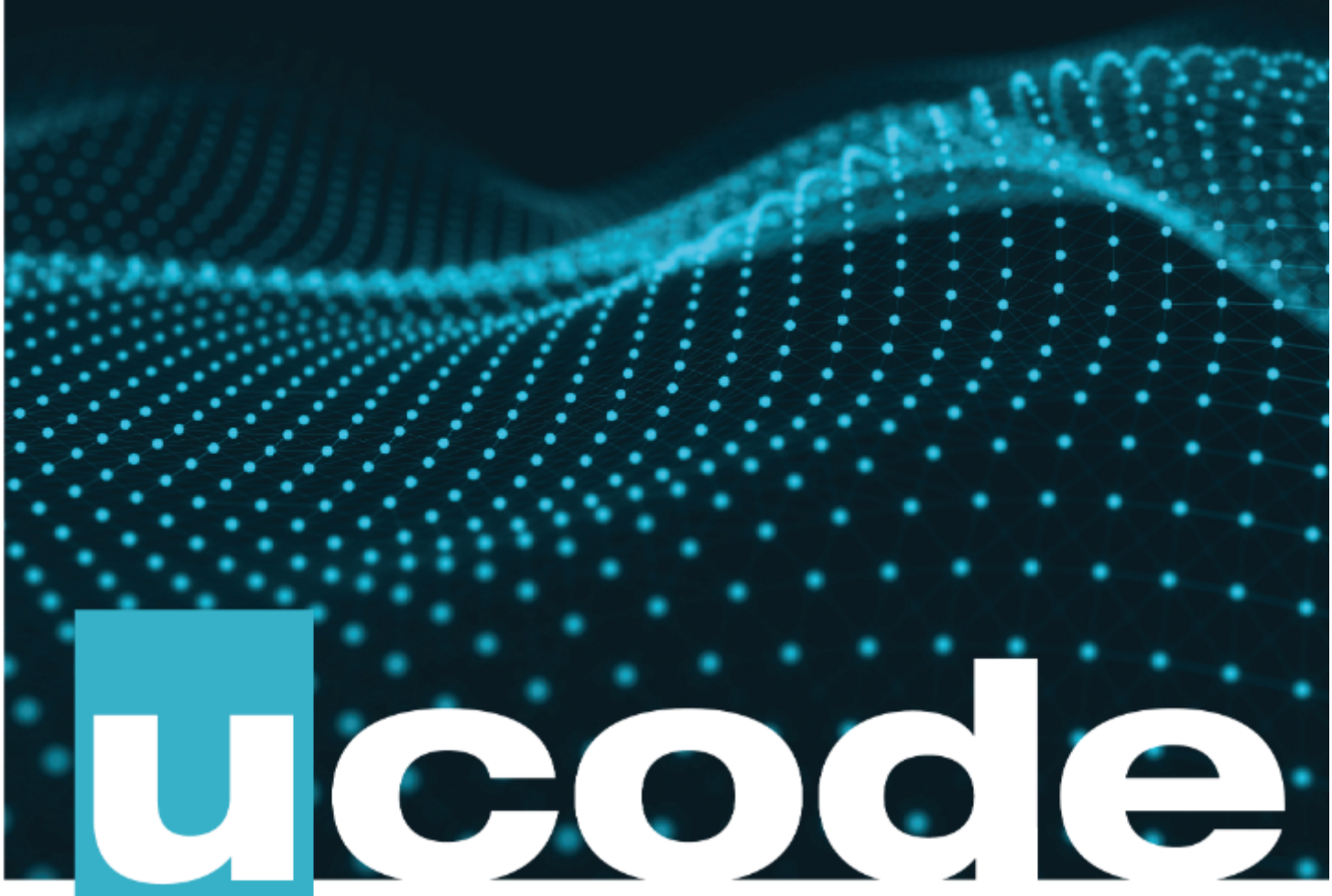
# Sprint 09

Marathon C

April 22, 2020



ucode

# Contents



ucode

# Engage

## DESCRIPTION

Hiya!
There are already nine Sprints behind.
During this period of time, you should have gained a lot of knowledge.

Have you ever thought about automating your daily routine? As a developer, you must always think about automation of your work and work of programs. As soon as you notice a repeating pattern, you must automate it.

Automating the build of a program saves time and effort. Compiling source code into binary code, packaging binary code, running automated tests, cleaning garbage, uninstalling a program, etc. These and many more processes can be done using a `Makefile`.

Another way to automate software development lies in function calls. Wouldn't it be convenient to call functions indirectly through a variable? In this Sprint you have a great opportunity to learn a new concept called a `function pointer`.

## BIG IDEA

Automation.

## ESSENTIAL QUESTION

What automation tools can be used in C programming?

## CHALLENGE

Build programs using a Makefile.

# 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 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 is the `standard stream` in C?
- What is a `macro`?
- How does the compiler process macros?
- How does a `Makefile` work?
- What is the scope of use of `function pointers`?
- What is the syntax of function pointers in C?
- What is a `binary file`?
- What is `enumeration` in C?
- Should the Makefile relink objects if they haven't changed?
- What are `wildcards` and how to use them?
- What are `automatic variables` and how to use them?
- Is there something useful about Makefiles in the Auditor?
- What is a `static library` and how to create it?
- What is a `conditional operator` and how to write it in C?
- Where could you use `conditional operators`?

## 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 yesterday. Repeat everything you know and do not know about pointers, because today they will also be needed.
- Read about the standard error stream `stderr` and how to work with it.
- Research and think about when and why you need to use `macros`.
- Find information about `function pointers` in `C`. Try to use them in practice.
- Try to understand why you need a `Makefile`.
- Read this article about `Makefiles`. Keep in mind that this article is just a brief introduction, research this tricky topic more by yourself. Pay close attention to it.
- Define the required directory structure for the source code and develop the solution.
- Clone your git repository that is issued on the challenge page in the LMS.
- Arrange brainstorming sessions with other students.
- Try to implement your thoughts in code.
- Push the solution to the repository.

## 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.
- 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

Print error

## DIRECTORY

`t00/`

## SUBMIT

`printerr.h, mx_printerr.c, mx_strlen.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

Macros

## DIRECTORY

`t01/`

## SUBMIT

`macros.h`

## ALLOWED FUNCTIONS

None

## DESCRIPTION

Create a `macros.h` file that contains single-line function-like macros:

- `MX_SUM(x, y)` - returns the sum of arguments `x, y`
- `MX_MULT(x, y)` - returns the multiplication of arguments `x, y`
- `MX_MIN(x, y)` - returns the smallest of arguments `x, y`
- `MX_ABS(x)` - returns the absolute value of the argument `x`
- `MX_IS_ODD(x)` - returns `1` if the argument `x` is odd and `0` in other cases

Oracle will test this header with its main.

## SEE ALSO

Macros

# Act: Task 02

## NAME

Makefile: first encounter

## DIRECTORY

`t02/`

## SUBMIT

`Makefile, inc/*.[h], src/*.[c]`

## BINARY

`minilibmx.a`

## FORBIDDEN STUFF

`*, ?, %, $@, $<, $^`

## DESCRIPTION

Create a `Makefile` without wildcards that:

- gets source files from the `src` directory
- gets header files from the `inc` directory
- compiles object files in the `obj` directory
- build the binary in the root directory of the project

The following nine mandatory functions of your minilib must be handled:

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

Read the `Auditor` to find out more about Makefile conventions. For example, this time, your `Makefile` must have at least the following rules: `all`, `MINILIBMX`, `uninstall`, `clean` and `reinstall`.

## SEE ALSO

Makefile
man ar

# Act: Task 03

## NAME

Makefile: second encounter

## DIRECTORY

`t03/`

## SUBMIT

`Makefile, inc/*.[h], src/*.[c]`

## BINARY

`minilibmx.a`

## DESCRIPTION

In this task, follow the instructions below:

- do everything that you were asked to in the previous task, but, this time, with wildcards
- every file name must be mentioned in the `Makefile` only once
- do not forget about `Auditor`

# Act: Task 04

## NAME

Comparator

## DIRECTORY

`t04/`

## SUBMIT

`mx_comparator.c`

## ALLOWED FUNCTIONS

None

## DESCRIPTION

In this task, follow the directions below:

- create a function that searches the integer `x` in the array `arr` given `size`
- searching criteria must be defined in `bool (*compare)(int, int)` passed as a function pointer parameter, in which the first parameter is the array element and the second is `x`

## RETURN

- returns the `index` of the first element in the array matching `true` criteria of the `compare` function
- returns `-1` in case of errors or if `x` has not been found

## SYNOPSIS

```
int mx_comparator(const int *arr, int size, int x, bool (*compare)(int, int));
```

## EXAMPLE

```
bool equal_nums(int a, int b) {
    return a == b;
}

arr = {1, 2, 3, 4, 5};
mx_comparator(arr, 5, 3, equal_nums); //returns 2
mx_comparator(arr, 5, -1, equal_nums); //returns -1
```

## SEE ALSO

Function pointer

# 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
```

## SEE ALSO

Function pointer

# Act: Task 06

## NAME

The running time

## DIRECTORY

`t06/`

## SUBMIT

`mx_timer.c`

## ALLOWED FUNCTIONS

clock

## DESCRIPTION

Create a function that calculates the execution time of the function `f` in seconds.

## RETURN

- returns the execution time of the function `f` in seconds
- returns `-1` in case of errors

## SYNOPSIS

```
double mx_timer(void (*f)());
```

## FOLLOW THE WHITE RABBIT

man clock

# 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));
```

## EXAMPLE

```
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}'
```

# Act: Task 08

## NAME

Calculator

## DIRECTORY

`t08/`

## SUBMIT

`Makefile, inc/minilibmx.h, src/*.[c]`

## ALLOWED FUNCTIONS

`write, malloc, exit`

## BINARY

`calc`

## DESCRIPTION

Create a simple command-line calculator program.

- The calculator must obtain two operands and the operation as command-line arguments and output the result of the math equation to the standard output followed by a newline.
- The program must support five math operations: addition, subtraction, multiplication, division, modulo.
- You must use the `t_operation` to match each math operation with the corresponding function.
- You must use the enum `e_operation`.
- You must use the enum `e_error` for error handling.
- Usage and error messages must be printed to the standard error stream `stderr`, and must exactly match the messages listed in the CONSOLE OUTPUT

Tips for this task

1. All other stuff required to be in header files must be included in the `minilibmx.h`.
2. Oracle will compile your program with our `calculator.h` which is listed in the SYNOPSIS.
3. You must not SUBMIT `inc/calculator.h` but your `Makefile` must compile the program with it.

## SYNOPSIS

```c
#ifndef CALCULATOR_H
#define CALCULATOR_H

#include "minilibmx.h"

int mx_add(int a, int b);
int mx_sub(int a, int b);
int mx_mul(int a, int b);
int mx_div(int a, int b);
int mx_mod(int a, int b);

typedef struct s_operation {
    char op;
    int (*f)(int a, int b);
}              t_operation;

enum e_operation {
    SUB,
    ADD,
    MUL,
    DIV,
    MOD
};

enum e_error {
    INCORRECT_OPERAND,
    INCORRECT_OPERATION,
    DIV_BY_ZERO
};

#endif
```

## CONSOLE OUTPUT

```
>./calc | cat -e
usage: ./calc [operand1] [operation] [operand2]
>./calc 5 / 0 | cat -e
error: division by zero
>./calc 34az + 2147483641 | cat -e
error: invalid number
>./calc 5 @ 5 | cat -e
error: invalid operation
>./calc 5 + 5 | cat -e
10$
>
```