Sprint 03

Marathon C

April 15, 2020

ucode

Contents

ucode

Engage

DESCRIPTION

Hi!

Let's keep going and move on to new topics.

During this Sprint, you'll learn pointers in C and write more complex algorithms. Pointers are very important basic construction. Therefore, treat this challenge with special attention.

BIG IDEA

Learn to constantly learn.

ESSENTIAL QUESTION

What knowledge is important to you now?

CHALLENGE

Learn to use pointers in C.

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.

- How many people did you communicate and work with yesterday? 4, 8, 15, 16, 23..?
- What are your impressions of the assessments? Reflection?
- What did you learn during the assessment of an another student?
- What is the biggest discovery in `C` for you at the moment?
- What is still unclear in `C` for you at this time?
- How to transform uppercase to lowercase?
- What is the `write` function? What do you know about it?
- What are `pointers`? Are there `strings` in C?

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. Write a program that outputs integer values to standard output using C (`mx_printint.c`) if you didn't do it yesterday.
- Spend time to fill in the gaps in knowledge from previous Sprints.
- If you have any questions, ask other students. Peer-to-Peer is your key to success.
- Take the most difficult task from the previous Sprints that you could not do before and try doing it now.
- Clone your git repository that is issued on the challenge page in the LMS.
  Use `git clone` for this.
- Open the story and read it!
- 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. 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!

- Compile C-files with clang compiler and use these flags:
  `clang -std=c11 -Wall -Wextra -Werror -Wpedantic` .
- 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

Dereferencing a pointer

DIRECTORY

`t00/`

SUBMIT

`mx_deref_pointer.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that takes as a parameter `******str` pointer to pointer to pointer to pointer to pointer to pointer of `char` and sets the string `Follow the white rabbit!` to the pointer of `char` .

SYNOPSIS

```
void mx_deref_pointer(char ******str);
```

SEE ALSO

Pointers in C

Act: Task 01

NAME

Referencing a pointer

DIRECTORY

`t01/`

SUBMIT

`mx_ref_pointer.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that takes `int i` as a parameter and sets its value to another parameter `int ******ptr` , which is a pointer to pointer to pointer to pointer to pointer to the pointer of `int` .

SYNOPSIS

```
void mx_ref_pointer(int i, int ******ptr);
```

SEE ALSO

Pointers in C

Act: Task 02

NAME

Reverse case

DIRECTORY

`t02/`

SUBMIT

`mx_reverse_case.c, mx_tolower.c, mx_toupper.c, mx_islower.c, mx_isupper.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that reverses the case of string characters in place.

SYNOPSIS

```
void mx_reverse_case(char *s);
```

EXAMPLE

```
HeLLo Neo // string before function call

hEllO nEO // string after function call
```

Act: Task 03

NAME

Swap characters

DIRECTORY

`t03/`

SUBMIT

`mx_swap_char.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that swaps the characters of a string using pointers.

SYNOPSIS

```
void mx_swap_char(char *s1, char *s2);
```

EXAMPLE

```
str = "ONE";
mx_swap_char(&str[0], &str[1]); //'str' now is "NOE"
mx_swap_char(&str[1], &str[2]); //'str' now is "NEO"
```

Act: Task 04

NAME

Reverse string

DIRECTORY

`t04/`

SUBMIT

`mx_str_reverse.c, mx_strlen.c, mx_swap_char.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that reverses a string using pointers.

SYNOPSIS

```
void mx_str_reverse(char *s);
```

EXAMPLE

```
str = "game over";
mx_str_reverse(str); //'str' now is "revo emag"
```

Act: Task 05

NAME

Compare strings

DIRECTORY

`t05/`

SUBMIT

`mx_strcmp.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

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

SYNOPSIS

```
int mx_strcmp(const char *s1, const char *s2);
```

FOLLOW THE WHITE RABBIT

man 3 strcmp

Act: Task 06

NAME

Copy string

DIRECTORY

`t06/`

SUBMIT

`mx_strcpy.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

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

SYNOPSIS

```
char *mx_strcpy(char *dst, const char *src);
```

FOLLOW THE WHITE RABBIT

man 3 strcpy

Act: Task 07

NAME

Separate string

DIRECTORY

`t07/`

SUBMIT

`mx_str_separate.c, mx_printchar.c`

ALLOWED FUNCTIONS

write

DESCRIPTION

Create a function that:

- separates a given string by a delimiter
- prints each fragment to standard output
- separates each fragment with a newline

SYNOPSIS

```
void mx_str_separate(const char *str, char delim);
```

CONSOLE OUTPUT

```
>./mx_str_separate | cat -e    # str = "game over", delim = ' '
game$
over$
>./mx_str_separate | cat -e    # str = "game over", delim = 'm'
ga$
e over$
>
./mx_str_separate | cat -e    # str = "MMMMatrix", delim = "M"
$
atrix$
>
```

Act: Task 08

NAME

Exponentiation

DIRECTORY

`t08/`

SUBMIT

`mx_pow.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

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

RETURN

Returns the result of `n` to the power of `pow` .

SYNOPSIS

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

EXAMPLE

```
mx_pow(3, 3); //returns 27
mx_pow(2.5, 3); //returns 15.625
mx_pow(2, 0); //returns 1
```

FOLLOW THE WHITE RABBIT

man pow

SEE ALSO

Exponentiation

Act: Task 09

NAME

Narcissistic number

DIRECTORY

`t09/`

SUBMIT

`mx_is_narcissistic.c, mx_pow.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that checks whether a number is narcissistic.

RETURN

Returns `true` if the number is narcissistic, else `false`.

SYNOPSIS

```
bool mx_is_narcissistic(int num);
```

EXAMPLE

```
mx_is_narcissistic(3); //returns true
mx_is_narcissistic(-3); //returns false
mx_is_narcissistic(10); //returns false
```

SEE ALSO

Narcissistic number

Act: Task 10

NAME

Prime number

DIRECTORY

`t10/`

SUBMIT

`mx_is_prime.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that checks whether a number is prime.

RETURN

Returns `true` if the number is prime, else `false`.

SYNOPSIS

```
bool mx_is_prime(int num);
```

EXAMPLE

```
mx_is_prime(3); //returns true
mx_is_prime(4); //returns false
```

SEE ALSO

Prime number

Act: Task 11

NAME

Mersenne prime

DIRECTORY

`t11/`

SUBMIT

`mx_is_mersenne.c, mx_pow.c, mx_is_prime.c`

ALLOWED FUNCTIONS

None

DESCRIPTION

Create a function that checks whether a number is a Mersenne prime.

Hardcoding is forbidden!

RETURN

Returns `true` if the number is a Mersenne prime, else `false` .

SYNOPSIS

```
bool mx_is_mersenne(int n);
```

EXAMPLE

```
mx_is_mersenne(3); //returns true
mx_is_mersenne(11); //returns false
```

SEE ALSO

Mersenne prime number