You just released the optional tasks of this project. Have fun!

0x04. C - More functions, more nested loops

C

By: Julien Barbier

Weight: 1

Project will start Sep 15, 2022 6:00 AM, must end by Sep 16, 2022 6:00 AM

was released at Sep 15, 2022 12:00 PM

An auto review will be launched at the deadline

Resources

Read or watch:

- Nested while loops (/rltoken/aDRkFzUkVysnD94Dpm3w5g)
- C - Functions (/rltoken/zf4IZeoe0yFZL2X7_nznQQ)
- Learning to Program in C (Part 06) (/rltoken/iQ87CI4Lf41U_uRh9QsoQA) (*stop at 14:00*)
- What is the purpose of a function prototype? (/rltoken/pUXhvD6-xI5BbWyj1AhCEA)
- C - Header Files (/rltoken/IFY075ffrszSJvHqPAa-zQ) (*stop before the "Once-Only Headers" paragraph*)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/Ya6FG69nkA7hRf_WG4E8gQ), **without the help of Google**:

General

- What are nested loops and how to use them
- What is a function and how do you use functions
- What is the difference between a declaration and a definition of a function

- What is a prototype
- Scope of variables
- What are the `gcc` flags `-Wall -Werror -pedantic -Wextra -std=gnu89`
- What are header files and how to to use them with `#include`

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using `gcc`, using the options `-Wall -Werror -Wextra -pedantic -std=gnu89`
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl (https://github.com/holbertonschool/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/holbertonschool/Betty/blob/master/betty-doc.pl)
- You are not allowed to use global variables
- No more than 5 functions per file
- You are not allowed to use the standard library. Any use of functions like `printf`, `puts`, etc... is forbidden
- You are allowed to use _putchar (https://github.com/holbertonschool/_putchar.c/blob/master/_putchar.c)
- You don't have to push `_putchar.c`, we will use our file. If you do it won't be taken into account
- In the following examples, the `main.c` files are shown as examples. You can use them to test your functions, but you don't have to push them to your repo (if you do we won't take them into account). We will use our own `main.c` files at compilation. Our `main.c` files might be different from the one shown in the examples
- The prototypes of all your functions and the prototype of the function `_putchar` should be included in your header file called `main.h`
- Don't forget to push your header file

More Info

You do not have to understand the call by reference (address), stack, static variables, recursions or arrays, yet.

Quiz questions

Great! You've completed the quiz successfully! Keep going! (Show quiz)

Tasks

0. isupper

mandatory

Write a function that checks for uppercase character.

- Prototype: `int _isupper(int c);`
- Returns `1` if `c` is uppercase
- Returns `0` otherwise

FYI: The standard library provides a similar function: `isupper` . Run `man isupper` to learn more.

```
julien@ubuntu:~/0x04$ cat 0-main.c
#include "main.h"
#include <stdio.h>

/**
 * main - check the code.
 *
 * Return: Always 0.
 */
int main(void)
{
    char c;

    c = 'A';
    printf("%c: %d\n", c, _isupper(c));
    c = 'a';
    printf("%c: %d\n", c, _isupper(c));
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 0-main.c 0-isup
per.c -o 0-isuper
julien@ubuntu:~/0x04$ ./0-isuper
A: 1
a: 0
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`

- File: `0-isupper.c`

(/)

Done! | Help | Check your code | >_ Get a sandbox

1. isdigit

mandatory

Write a function that checks for a digit (`0` through `9`).

- Prototype: `int _isdigit(int c);`
- Returns `1` if `c` is a digit
- Returns `0` otherwise

FYI: The standard library provides a similar function: isdigit. Run man isdigit to learn more.

```
julien@ubuntu:~/0x04$ cat 1-main.c
#include "main.h"
#include <stdio.h>

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    char c;

    c = '0';
    printf("%c: %d\n", c, _isdigit(c));
    c = 'a';
    printf("%c: %d\n", c, _isdigit(c));
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 1-main.c 1-isdi
git.c -o 1-isdigit
julien@ubuntu:~/0x04$ ./1-isdigit
0: 1
a: 0
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `1-isdigit.c`

Done! Help Check your code >_ Get a sandbox

2. Collaboration is multiplication

mandatory

Write a function that multiplies two integers.

- Prototype: `int mul(int a, int b);`

```
julien@ubuntu:~/0x04$ cat 2-main.c
#include "main.h"
#include <stdio.h>

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    printf("%d\n", mul(98, 1024));
    printf("%d\n", mul(-402, 4096));
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 2-main.c 2-mul.
c -o 2-mul
julien@ubuntu:~/0x04$ ./2-mul
100352
-1646592
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `2-mul.c`

Done! Help Check your code >_ Get a sandbox

3. The numbers speak for themselves

mandatory

Write a function that prints the numbers, from `0` to `9`, followed by a new line.

- Prototype: `void print_numbers(void);`
- You can only use `_putchar` twice in your code

```
julien@ubuntu:~/0x04$ cat 3-main.c
#include "main.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    print_numbers();
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 _putchar.c 3-ma
in.c 3-print_numbers.c -o 3-print_numbers
julien@ubuntu:~/0x04$ ./3-print_numbers | cat -e
0123456789$
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `3-print_numbers.c`

Done! | Help | Check your code | >_ Get a sandbox

4. I believe in numbers and signs

mandatory

Write a function that prints the numbers, from `0` to `9`, followed by a new line.

- Prototype: `void print_most_numbers(void);`
- Do not print `2` and `4`
- You can only use `_putchar` twice in your code

```
julien@ubuntu:~/0x04$ cat 4-main.c
#include "main.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    print_most_numbers();
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 _putchar.c 4-ma
in.c 4-print_most_numbers.c -o 4-print_most_numbers
julien@ubuntu:~/0x04$ ./4-print_most_numbers
01356789
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `4-print_most_numbers.c`

Done! | Help | Check your code | >_ Get a sandbox

5. Numbers constitute the only universal language

mandatory

Write a function that prints 10 times the numbers, from `0` to `14`, followed by a new line.

- Prototype: `void more_numbers(void);`
- You can only use `_putchar` three times in your code

```
julien@ubuntu:~/0x04$ cat 5-main.c
#include "main.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    more_numbers();
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 _putchar.c 5-ma
in.c 5-more_numbers.c -o 5-more_numbers
julien@ubuntu:~/0x04$ ./5-more_numbers
01234567891011121314
01234567891011121314
01234567891011121314
01234567891011121314
01234567891011121314
01234567891011121314
01234567891011121314
01234567891011121314
01234567891011121314
01234567891011121314
julien@ubuntu:~/0x04
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `5-more_numbers.c`

Done! | Help | Check your code | >_ Get a sandbox

6. The shortest distance between two points is a straight line

mandatory

Write a function that draws a straight line in the terminal.

- Prototype: `void print_line(int n);`
- You can only use `_putchar` function to print
- Where `n` is the number of times the character `_` should be printed
- The line should end with a `\n`
- If `n` is `0` or less, the function should only print `\n`

```
julien@ubuntu:~/0x04$ cat 6-main.c
#include "main.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    print_line(0);
    print_line(2);
    print_line(10);
    print_line(-4);
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 _putchar.c 6-ma
in.c 6-print_line.c -o 6-lines
julien@ubuntu:~/0x04$ ./6-lines | cat -e
$
__$
__________$
$
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `6-print_line.c`

Done! | Help | Check your code | >_ Get a sandbox

7. I feel like I am diagonally parked in a parallel universe

mandatory

Write a function that draws a diagonal line on the terminal.

- Prototype: `void print_diagonal(int n);`
- You can only use `_putchar` function to print
- Where `n` is the number of times the character `\` should be printed
- The diagonal should end with a `\n`
- If `n` is `0` or less, the function should only print a `\n`

```
julien@ubuntu:~/0x04$ cat 7-main.c
#include "main.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    print_diagonal(0);
    print_diagonal(2);
    print_diagonal(10);
    print_diagonal(-4);
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 _putchar.c 7-ma
in.c 7-print_diagonal.c -o 7-diagonals
julien@ubuntu:~/0x04$ ./7-diagonals | cat -e
$
\$
 \$
\$
 \$
  \$
   \$
    \$
     \$
      \$
       \$
        \$
         \$
$
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `7-print_diagonal.c`

Done! | Help | Check your code | >_ Get a sandbox

8. You are so much sunshine in every square inch

mandatory

Write a function that prints a square, followed by a new line.

- Prototype: `void print_square(int size);`
- You can only use `_putchar` function to print
- Where `size` is the size of the square
- If `size` is `0` or less, the function should print only a new line
- Use the character `#` to print the square

```
julien@ubuntu:~/0x04$ cat 8-main.c
#include "main.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    print_square(2);
    print_square(10);
    print_square(0);
    return (0);
}

julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 _putchar.c 8-ma
in.c 8-print_square.c -o 8-squares
julien@ubuntu:~/0x04$ ./8-squares
##
##
##########
##########
##########
##########
##########
##########
##########
##########
##########
##########

julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `8-print_square.c`

9. Fizz-Buzz

mandatory

The “Fizz-Buzz test” is an interview question designed to help filter out the 99.5% of programming job candidates who can’t seem to program their way out of a wet paper bag.

Write a program that prints the numbers from `1` to `100`, followed by a new line. But for multiples of three print `Fizz` instead of the number and for the multiples of five print `Buzz`. For numbers which are multiples of both three and five print `FizzBuzz`.

- Each number or word should be separated by a space
- You are allowed to use the standard library

```
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 9-fizz_buzz.c -o 9-fizz_buzz
julien@ubuntu:~/0x04$ ./9-fizz_buzz
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11 Fizz 13 14 FizzBuzz 16 17 Fizz 19 Buzz Fizz 22 23 Fizz Buzz 26 Fizz 28 29 FizzBuzz 31 32 Fizz 34 Buzz Fizz 37 38 Fizz Buzz 41 Fizz 43 44 FizzBuzz 46 47 Fizz 49 Buzz Fizz 52 53 Fizz Buzz 56 Fizz 58 59 FizzBuzz 61 62 Fizz 64 Buzz Fizz 67 68 Fizz Buzz 71 Fizz 73 74 FizzBuzz 76 77 Fizz 79 Buzz Fizz 82 83 Fizz Buzz 86 Fizz 88 89 FizzBuzz 91 92 Fizz 94 Buzz Fizz 97 98 Fizz Buzz
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `9-fizz_buzz.c`

Done! | Help | Check your code | >_ Get a sandbox

10. Triangles

mandatory

Write a function that prints a triangle, followed by a new line.

- Prototype: `void print_triangle(int size);`
- You can only use `_putchar` function to print
- Where `size` is the size of the triangle
- If `size` is `0` or less, the function should print only a new line
- Use the character `#` to print the triangle

```
julien@ubuntu:~/0x04$ cat 10-main.c
#include "main.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    print_triangle(2);
    print_triangle(10);
    print_triangle(1);
    print_triangle(0);
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 _putchar.c 10-m
ain.c 10-print_triangle.c -o 10-triangles
julien@ubuntu:~/0x04$ ./10-triangles
 #
##
         #
        ##
       ###
      ####
     #####
    ######
   #######
  ########
 #########
##########
#

julien@ubuntu:~/0x04$ ./10-triangles | tr ' ' . | cat -e
.#$
##$
.........#$
........##$
.......###$
......####$
.....#####$
....######$
...#######$
..########$
.#########$
##########$
#$
$
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `10-print_triangle.c`

Done! | Help | Check your code | >_ Get a sandbox

11. The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic

#advanced

The prime factors of `1231952` are `2`, `2`, `2`, `2`, `37` and `2081`.

Write a program that finds and prints the largest prime factor of the number `612852475143`, followed by a new line.

- You are allowed to use the standard library
- Your program will be compiled with this command: `gcc -Wall -pedantic -Werror -Wextra -std=gnu89 100-prime_factor.c -o 100-prime_factor -lm`

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `100-prime_factor.c`

Done! | Help | Check your code | >_ Get a sandbox

12. Numbers have life; they're not just symbols on paper

#advanced

Write a function that prints an integer.

- Prototype: `void print_number(int n);`
- You can only use `_putchar` function to print
- You are not allowed to use `long`
- You are not allowed to use arrays or pointers
- You are not allowed to hard-code special values

```
julien@ubuntu:~/0x04$ cat 101-main.c
#include "main.h"

/**
 * main - check the code
 *
 * Return: Always 0.
 */
int main(void)
{
    print_number(98);
    _putchar('\n');
    print_number(402);
    _putchar('\n');
    print_number(1024);
    _putchar('\n');
    print_number(0);
    _putchar('\n');
    print_number(-98);
    _putchar('\n');
    return (0);
}
julien@ubuntu:~/0x04$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 _putchar.c 101-main.c 101-print_number.c -o 101-print_numbers
julien@ubuntu:~/0x04$ ./101-print_numbers
98
402
1024
0
-98
julien@ubuntu:~/0x04$
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x04-more_functions_nested_loops`
- File: `101-print_number.c`

Done! | Help | Check your code | >_ Get a sandbox