Version: 1.0

Selection

C-Programming-Fundamentals

Summary

Implement fundamental C functions to display characters, the alphabet, numbers, and check integer signs using standard output.

#C #Fundamentals #Output

Intellectual Property Disclaimer

All content presented in this training module, including but not limited to texts, images, graphics, and other materials, is protected by intellectual property rights held by Association 42.

Terms of Use:

- **Personal use:** You are permitted to use the contents of this module solely for personal purpose. Any commercial use, reproduction, distribution, modification, or public display is strictly prohibited without prior written permission from Association 42.
- **Respect for Integrity:** You must not alter, transform, or adapt the content in any way that could harm its integrity.

Protection of Rights:

Any violation of these terms constitutes an infringement of intellectual property rights and may result in legal action. We reserve the right to take all necessary measures to protect our rights, including but not limited to claims for damages.

For any questions regarding the use of the content or to obtain authorization, please contact: `legal@42.fr`

Contents

Chapter 1

Instructions

- Only this page will serve as a reference: do not trust rumors.
- Watch out! This document could potentially change up until submission.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for all your exercises.
- Your exercises will be checked and graded by your fellow classmates.
- Additionally, your exercises will be checked and graded by a program called Moulinette.
- Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated, and there is no way to negotiate with it. So, to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try to understand your code if it doesn't adhere to the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be foolish to submit work that doesn't pass `norminette`'s check.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` consider a successfully completed harder exercise if an easier one is not perfectly functional.
- Using a forbidden function is considered cheating. Cheaters get -42, and this grade is non-negotiable.
- You'll only have to submit a main() function if we ask for a program.
- Moulinette compiles with these flags: -Wall -Wextra -Werror, and uses `cc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional files in your directory other than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ...`.
- Check out the Slack Piscine.

- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- By Odin, by Thor! Use your brain!!!

Do not forget to add the *standard 42 header* in each of your .c/.h files. Norminette checks its existence anyway!

Norminette must be launched with the *-R CheckForbiddenSourceHeader* flag. Moulinette will use it too.

Context

The C Piscine is intense. It's your first big challenge at 42 — a deep dive into problem-solving, autonomy, and community.

During this phase, your main objective is to build your foundation — through struggle, repetition, and especially **peer-learning** exchange.

In the AI era, shortcuts are easy to find. However, it's important to consider whether your AI usage is truly helping you grow — or simply getting in the way of developing real skills.

The Piscine is also a human experience — and for now, nothing can replace that. Not even AI.

For a more complete overview of our stance on AI — as a learning tool, as part of the ICT curriculum, and as a growing expectation in the job market — please refer to the dedicated FAQ available on the intranet.

Main message

- 👉 Build strong foundations without shortcuts.
- 👉 Really develop tech & power skills.
- 👉 Experience real peer-learning, start learning how to learn and solve new problems.
- 👉 The learning journey is more important than the result.
- 👉 Learn about the risks associated with AI, and develop effective control practices and countermeasures to avoid common pitfalls.

Learner rules:

- You should apply reasoning to your assigned tasks, especially before turning to AI.
- You should not ask for direct answers to the AI.
- You should learn about 42 global approach on AI.

Phase outcomes:

Within this foundational phase, you will get the following outcomes:

- Get proper tech and coding foundations.
- Know why and how AI can be dangerous during this phase.

Comments and example:

- Yes, we know AI exists — and yes, it can solve your projects. But you're here to learn, not to prove that AI has learned. Don't waste your time (or ours) just to demonstrate that AI can solve the given problem.
- Learning at 42 isn't about knowing the answer — it's about developing the ability to find one. AI gives you the answer directly, but that prevents you from building your own reasoning. And reasoning takes time, effort, and involves failure. The path to success is not supposed to be easy.
- Keep in mind that during exams, AI is not available — no internet, no smartphones, etc. You'll quickly realise if you've relied too heavily on AI in your learning process.
- Peer learning exposes you to different ideas and approaches, improving your interpersonal skills and your ability to think divergently. That's far more valuable than just chatting with a bot. So don't be shy — talk, ask questions, and learn together!
- Yes, AI will be part of the curriculum — both as a learning tool and as a topic in itself. You'll even have the chance to build your own AI software. In order to learn more about our crescendo approach you'll go through in the documentation available on the intranet.

✓ Good practice:

> I'm stuck on a new concept. I ask someone nearby how they approached it. We talk for 10 minutes — and suddenly it clicks. I get it.

✗ Bad practice:

> I secretly use AI, copy some code that looks right. During peer evaluation, I can't explain anything. I fail. During the exam — no AI — I'm stuck again. I fail.

Chapter 2

Foreword

Cod liver oil is a nutritional supplement derived from liver of cod fish (Gadidae).

As with most fish oils, it has high levels of the omega-3 fatty acids, eicosapentaenoic acid (EPA) and docosahexaenoic acid (DHA).
Cod liver oil also contains vitamin A and vitamin D.

It has historically been taken because of its vitamin A and vitamin D content.

It was once commonly given to children, because vitamin D has been shown to prevent rickets and other symptoms of vitamin D deficiency.

> Contrary to Cod liver oil, C is good, eat some!

Chapter 3

Tutorial

Welcome to your first **C** programming tutorial! Before diving into code, you need to understand the fundamentals. Watch the following video about programming languages and computer memory : The C coding Language Pay attention to how computers store and manipulate data.

- Understanding **memory** and how programs work is crucial before writing your first line of **C** code.
- After watching the video, create your first `C` program. Following the video example, create an empty `main` function in a file called `first_program.c`:

first_program.c

```
int main()
{
    return (0);
}
```

- Compile it using the folowing command : `cc first_program.c o first_program`
- Run it with the command `./first_program`. What happens ?
- Nothing visible happens? That's normal. You don't have any instructions yet! Modify your code and add a simple instruction like `1+1;` inside the `main` function. Compile and run it again.
- Still nothing visible? Actually your code did something, but you just don't "see" it. The computer calculated `1+1=2`, but didn't display anything. Time to make something visible!
- You need to display something in the terminal. Create the `ft_putchar` function that displays one character:

first_program.c

```
#include <unistd.h>

void ft_putchar(char c)
{
    write(1, &c, 1);
}
```

- Now create a complete program that displays a character. Call `ft_putchar` from your `main` function. The `char` parameter is an `ASCII` code (a number between 0 and 127 that represents a character).
- Try displaying different characters:

first_program.c

```
ft_putchar('A');
ft_putchar('B');
ft_putchar('*');
```

What happens ?

ASCII codes are how computers represent characters as numbers. 'A' is 65, 'a' is 97, '0' is 48, etc. With the help of your peers, try to find the shell command which display the ASCII table in your terminal.

- Now let's use a variable! Define a `char` variable, assign it an `ASCII` character value, and display it using `ft_putchar`:

first_program.c

```
char my_char;
my_char = 'X';
ft_putchar(my_char);
```

- Try adding one to the variable while calling `ft_putchar`:

first_program.c

```
char my_char;
my_char = 'X';
ft_putchar(my_char + 1);
```

What `character` appears and why?

- Now you need a loop! How could you repeat 10 times the display of one `character`? Research **counter variables** and the **while** structure online. Try to display the same `character` 10 times in a row.
- Create a loop that displays different `characters`. For example, display letters `A` to `J` using a counter variable that you add to a base `ASCII` value.
- Loops are fundamental in programming. They allow you to repeat actions without writing the same code multiple times.
- Experiment with different loop patterns: count up, count down, display patterns like stars or numbers...
- You're ready for the exercises! You'll experiment with various **loops**, **conditional branches** (`if, else`), and mix them together. Always remember to ask peers for help and to discuss your code.

  The key to learning programming is practice, experimentation, and peer discussion. Don't hesitate to try different approaches!

Chapter 4

Exercice 0: ft_putchar

	Exercise0	
ft_putchar		
Directory: *ex0/*		
Files to Submit: `ft_putchar.c`		
Authorized: `write`		

- Write a function that displays the character passed as a parameter.

Prototype:

```
void ft_putchar(char c);
```

- To display the character, you must use the `write` function as follows.

Prototype:

```
write(1, &c, 1);
```

The first retry delay is short, do not hesitate to trigger an intermediate evaluation to measure your progress.

Chapter 5

Exercise 1: ft_print_alphabet

	Exercise1	
ft_print_alphabet		
Directory: *ex1/*		
Files to Submit: `ft_print_alphabet.c`		
Authorized: `write`		

- Create a function that displays the alphabet in lowercase, on a single line, by ascending order, starting from the letter 'a'.

Prototype:

```
void ft_print_alphabet(void);
```

Do not hesitate to pickup randomly someone in your cluster to ask a question.

Chapter 6

Exercise 2: ft_print_reverse_alphabet

	Exercise2	
ft_print_reverse_alphabet		
Directory: *ex2/*		
Files to Submit: `ft_print_reverse_alphabet.c`		
Authorized: `write`		

- Create a function that displays the alphabet in lowercase, on a single line, by descending order, starting from the letter 'z'.

Prototype:

```
void ft_print_reverse_alphabet(void);
```

Git push regularly.

Chapter 7

Exercise 3: ft_print_numbers

	Exercise3	
ft_print_numbers		
Directory: *ex3/*		
Files to Submit: `ft_print_numbers.c`		
Authorized: `write`		

- Create a function that displays all digits, on a single line, by ascending order.

Prototype:

```
void ft_print_numbers(void);
```

Collaboration is a key to success.

Chapter 8

Exercise 4: ft_is_negative

	Exercise4	
ft_is_negative		
Directory: *ex4/*		
Files to Submit: `ft_is_negative.c`		
Authorized: `write`		

- Create a function that displays 'N' or 'P' depending on the integer's sign entered as a parameter. If `n` is negative, display 'N'. If `n` is positive or null, display 'P'.

Prototype:

```
void ft_is_negative(int n);
```

:

Failure is part of your learning journey.

Chapter 9

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

You need to return only the files requested by the subject of this project.