Version: 1.0

Selection

C-Algorithmics-Fundamentals

Summary

• • •

This project introduces fundamental algorithmic concepts in C, focusing on recursion, mathematical computations, and problem-solving techniques.

#C

#Algorithms

#Recursion

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 activities. 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

Here is a discuss extract from the `Silicon Valley` serie:

```
- I mean, why not just use Vim over Emacs? (CHUCKLES)
- I do use Vim over Emac.
- Oh, God, help us! Okay, uh you know what? I just don't think this is
  going to work. I'm so sorry. Uh, I mean like, what, we're going to bring
  kids into this world with that over their heads? That's not really fair
  to them, don't you think?
- Kids? We haven't even slept together.
- And guess what, it's never going to happen now, because there is no
  way I'm going to be with someone who uses spaces over tabs.
- Richard! (PRESS SPACE BAR MANY TIMES)
- Wow. Okay. Goodbye.
- One tab saves you eight spaces! - (DOOR SLAMS) - (BANGING)


. . .


(RICHARD MOANS)

- Oh, my God! Richard, what happened?
- I just tried to go down the stairs eight steps at a time. I'm okay, though.
- See you around, Richard.
- Just making a point.
```

Hopefully, you are not forced to use emacs and your space bar to complete the following exercices.

Chapter 3

Tutorial

- Let's talk about one of the big classics of programming: **Recursivity**. You should now be familiar with the execution timeline of your code. When your program is executed, it starts with the `main` function.

- If your `main` function calls another function `F`, the `main` function is suspended. The parameters of `F` are calculated and made available inside `F` code. When `F` ends, it returns a value, and the suspended `main` function continues.

- What if `F` also calls another `F` function? At any moment, only one function is executed at a time, and you have a chain of suspended functions, starting from `main`.

> 💡 Take a pause here and make sure you understand this execution behavior before continuing! Validate your understanding with one of your peers.

- Now let's twist: What if your function `F` calls... function `F` itself? It's totally possible in C, and it's called **recursivity**. Following our execution timeline, it means that `F` is suspended, its variables still exist in RAM, and the local variables of `F` -the second- are created.

- The key is to understand that the computer acts like the first and second `F` functions were completely different. The variables of the first `F` function will be different in memory from the variables of the second `F` function.

- Create a working directory and write a simple recursive function that calls itself from `main`:

```
main.c

void f()
{
    f();
}
```

Compile and run this program. What happens? Your program should crash very quickly.

- Add a counter to see how many times the function is called before crashing:

main.c

```c
void f()
{
    static int count = 0;
    count++;
    printf(''%d\n'', count);
    f();
}
```

- Run again and note the last number printed. This is roughly how many recursive calls your system can handle.

- Now prevent the crash by stopping recursion before the limit. Use your count to stop calling the function again:

main.c

```c
void f()
{
    static int count = 0;
    count++;
    if (count < 1000) //Use a number below your crash limit
        f();
}
```

- Your program should now end normally. Test it.

- Remove the print inside your function. Using only one `print` in `main`, show the number of recursive calls using the return value:

```
main.c

int f()
{
    static int count = 0;
    count++;
    if (count < 1000)
        return (1 + f());
    return (1);
}
```

- The function must calculate the total through the return values of recursive calls, not just return a fixed number.

- Each recursive call has its own set of parameters and local variables. They don't interfere with each other even though they're the same function.

Chapter 4

Exercise 0: ft_iterative_factorial

	Exercise: 0	
	ft_iterative_factorial	
Directory: *ex0/*		
Files to Submit: `ft_iterative_factorial.c`		
Authorized: `None`		

- Create an iterative function that returns a number. This number is the result of a factorial operation on the number given as a parameter.

- If the argument is not valid the function should return 0.

- Overflows must not be handled, the function return will be undefined.

Prototype:
```
int ft_iterative_factorial(int nb);
```

Chapter 5

Exercise 1: ft_recursive_factorial

	Exercise: 1	
	ft_recursive_factorial	
Directory: *ex1/*		
Files to Submit: `ft_recursive_factorial.c`		
Authorized: `None`		

- Create a recursive function that returns the factorial of the number given as a parameter.

- If the argument is not valid the function should return 0.

- Overflows must not be handled, the function return will be undefined.

Prototype:

```
int ft_recursive_factorial(int nb);
```

Chapter 6

Exercise 2: ft_iterative_power

![]	Exercise: 2	
	ft_iterative_power	
	Directory: *ex2/*	
	Files to Submit: `ft_iterative_power.c`	
	Authorized: `None`	

- Create an iterative function that returns the value of a power applied to a number.

- A negative power returns 0.

- Overflows must not be handled.

- We've decided that 0 power 0 will returns 1

Prototype:

```
int ft_iterative_power(int nb, int power);
```

Chapter 7

Exercise 3: ft_recursive_power

	Exercise: 3	
	ft_recursive_power	
Directory: *ex3/*		
Files to Submit: `ft_recursive_power.c`		
Authorized: `None`		

- Create a recursive function that returns the value of a number raised to a power.

- A negative power returns 0.

- Overflows must not be handled, the function return will be undefined.

- We've decided that 0 power 0 will returns 1

Prototype:

```
int ft_recursive_power(int nb, int power);
```

Chapter 8

Exercise 4: ft_fibonacci

	Exercise: 4	
ft_fibonacci		
Directory: *ex4/*		
Files to Submit: `ft_fibonacci.c`		
Authorized: `None`		

- Create a function `ft_fibonacci` that returns the n-th element of the Fibonacci sequence, the first element being at the 0 index. We'll consider that the Fibonacci sequence starts like this: 0, 1, 1, 2.

- Overflows must not be handled, the function return will be undefined.

Prototype:

```
int ft_fibonacci(int index);
```

- The function `ft_fibonacci` must be recursive.

- If the `index` is negative, the function should return -1.

Chapter 9

Submission and peer-evaluation

Submit your assignment to 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 submit only the files requested by the subject of this project.