Version: 1.0

Selection

C-Pointers

Summary

Practice fundamental pointer operations in C, including value assignment, swapping, arithmetic operations, and array manipulation.

#C

#Pointers

#Memory-Management

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

The first known mention of the game of RPS was in the book Wuzazu written by the Chinese Ming-dynasty writer Xie Zhaozhi who wrote that the game dated back to the time of the Chinese Han dynasty (206 BC – 220 AD). In the book, the game was called shoushiling. Li Rihua's book Note of Liuyanzhai also mentions this game, calling it shoushiling, huozhitou, or huoquan.

Throughout Japanese history there are frequent references to "sansukumi-ken", meaning "ken" fist games with a "san" three-way "sukumi" deadlock. This is in the sense that A beats B, B beats C, and C beats A. The games originated in China before being imported to Japan and subsequently becoming popular.

By the early 20th century, rock–paper–scissors had spread beyond Asia, especially through increased Japanese contact with the west. Its English-language name is therefore taken from a translation of the names of the three Japanese hand-gestures for rock, paper and scissors: elsewhere in Asia the open-palm gesture represents "cloth" rather than "paper". The shape of the scissors is also adopted from the Japanese style.

In 1927 La Vie au patronage, a children's magazine in France, described it in detail, referring to it as a "jeu japonais" ("Japanese game"). Its French name, "Chi-fou-mi", is based on the Old Japanese words for "one, two, three" ("hi, fu, mi")

A New York Times article of 1932 on the Tokyo rush hour describes the rules of the game for the benefit of American readers, suggesting it was not at that time widely known in the U.S. The 1933 edition of the Compton's Pictured Encyclopedia described it as a common method of settling disputes between children in its article on Japan; the name was given as "John Kem Po" and the article pointedly asserted, "This is such a good way of deciding an argument that American boys and girls might like to practice it too."

Chapter 3

Tutorial

- It's now time to talk about the magic part of C programming: **POINTERS**! A **pointer** is a special type in `C` that contains an address. Each byte in RAM has its own individual address, and a pointer stores the address of another byte in RAM.

- A `pointer` is not standalone - you always have a pointer to another `C` type. Ex: you can have a char pointer, or an int pointer. An int pointer stores the address of an int existing somewhere else in memory.

- You should declare a pointer as follow `<type> *<var_name>`. For example: `int *pointer_on_int;`

> A pointer doesn't contain a value like other types. It contains an address to where a value is stored. If you do not understand fully, you must check your understanding with one of your neighbors.

- Create a working directory and write a simple `C` file. Declare an int pointer and compile it to verify the syntax:

main.c

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

- You can assign an address to the pointer using the `&` operator to get the address of an existing variable:

Example

```
int a;
int *p;
p = &a;
```

- Add an `int` variable, an `int` pointer, and assign the address of the `int` to the pointer. Compile and test.

- You can use the pointer to read and update the other variable. This is called “dereferencing” using the `*` operator. For example, `*p` can be used as a regular `int` variable.

- Assign a value to your `int` variable and print it using your `ft_putnbr` function.

- Now print the same value but using the pointer instead of the variable name: `ft_putnbr(*p);`. Verify both prints show the same value.

- Update the `int` using the variable name, then print again using both the variable and the pointer. What do you observe?

- Update the `int` using the pointer: `*p = 100;` Then print again using both methods. What happens?

> Let's make it simple for `int *p;` :
> `*p` is a var of type `int` whose address is stored in `p`.

- Now let's explore `strings`. In `C`, a string is stored as a series of bytes in RAM, each containing the `ASCII` value of each character. The string ends with a byte containing 0.

- You can create a string using double quotes: `‘‘Hello’’`. This finds memory space and stores ASCII codes. The expression evaluates to the address of the first character.

- Declare a `char` pointer and assign it a `string`:

Example

```
char *s;

s = ‘‘Hello’’;
```

- Use `*s` to access the first `char` and print it with `ft_putchar(*s);`.

- Here comes “pointer arithmetic”! If `s` points to the first `char`, then `s+1` points to the second `char`.
  Try the following:

Example

```
char *s;

s = ‘‘Hello’’;
s = s + 1;
ft_putchar(*s);
```

- Now, try :

Example

```
char *s;

s = ‘‘Hello’’;
ft_putchar(*(s+1));
```

Check with one of your peers if you see any difference.

- Pointer arithmetic depends on the pointed type. `char_ptr+1` points to next `char`, `int_ptr+1` points to next `int` (4 bytes away).
- Experiment with moving through a string character by character using pointer arithmetic. Print each character individually.
- Try this mystery: write a function that receives an `int` pointer as parameter and modifies the value it points to. Call this function from main and verify the original variable was modified.

Chapter 4

Exercise 0: ft_ft

	Exercise0	
ft_ft		
Directory: *ex0/*		
Files to Submit: `ft_ft.c`		
Authorized: `None`		

- Create a function that takes a pointer to int as a parameter, and sets the value "42" to that int.

Prototype:

```
void ft_ft(int *nbr);
```

Chapter 5

Exercise 1: ft_ultimate_ft

	Exercise1	
ft_ultimate_ft		
Directory: *ex1/*		
Files to Submit: `ft_ultimate_ft.c`		
Authorized: `None`		

- Create a function that takes a pointer to pointer to pointer to pointer to pointer to pointer to pointer to pointer to pointer to int as a parameter and sets the value "42" to that int.

Prototype:

```
void ft_ultimate_ft(int *********nbr);
```

Chapter 6

Exercise 2: ft_swap

	Exercise2	
ft_swap		
Directory: *ex2/*		
Files to Submit: `ft_swap.c`		
Authorized: `None`		

- Create a function that swaps the value of two integers whose addresses are entered as parameters.

Prototype:

```
void ft_swap(int *a, int *b);
```

Chapter 7

Exercise 3: ft_div_mod

	Exercise3	
ft_div_mod		
Directory: *ex3/*		
Files to Submit: `ft_div_mod.c`		
Authorized: `None`		

- Create a function `ft_div_mod`.

Prototype:

```
void ft_div_mod(int a, int b, int *div, int *mod);
```

- This function divides parameters `a` by `b` and stores the result in the int pointed by `div`. It also stores the remainder of the division of `a` by `b` in the int pointed by `mod`.

Chapter 8

Exercise 4: ft_ultimate_div_mod

	Exercise4	
ft_ultimate_div_mod		
Directory: *ex4/*		
Files to Submit: `ft_ultimate_div_mod.c`		
Authorized: `None`		

- Create a function `ft_ultimate_div_mod`.

Prototype:

```
void ft_ultimate_div_mod(int *a, int *b);
```

- This function divides parameters `a` by `b`.
  - The result of this division is stored in the int pointed by `a`.
  - The remainder of the division is stored in the int pointed by `b`.

Chapter 9

Exercise 5: ft_rev_int_tab

	Exercise5	
ft_rev_int_tab		
Directory: *ex5/*		
Files to Submit: `ft_rev_int_tab.c`		
Authorized: `None`		

- Create a function which reverses a given array of integer (first goes last, etc).

Prototype:

```
void ft_rev_int_tab(int *tab, int size);
```

- The arguments are a pointer to int and the number of ints in the array.

Chapter 10

Exercise 6: ft_sort_int_tab

	Exercise6	
ft_sort_int_tab		
Directory: *ex6/*		
Files to Submit: `ft_sort_int_tab.c`		
Authorized: `None`		

- Create a function which sorts an array of integers by ascending order.

Prototype:

```
void ft_sort_int_tab(int *tab, int size);
```

- The arguments are a pointer to int and the number of ints in the array.

Chapter 11

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.