Version: 1.0

Selection

C-Structures

Summary

Implement fundamental C structures and memory management techniques to manipulate and store data efficiently.

#C #Structures #MemMgmt

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's what Wikipedia have to say about Platypus:

The platypus (Ornithorhynchus anatinus), also known as the duck-billed platypus, is a semiaquatic egg-laying mammal endemic to eastern Australia, including Tasmania. Together with the four species of echidna, it is one of the five extant species of monotremes, the only mammals that lay eggs instead of giving birth. The animal is the sole living representative of its family (Ornithorhynchidae) and genus (Ornithorhynchus), though a number of related species have been found in the fossil record.

The unusual appearance of this egg-laying, duck-billed, beaver-tailed, otter-footed mammal baffled European naturalists when they first encountered it, with some considering it an elaborate hoax. It is one of the few venomous mammals, the male platypus having a spur on the hind foot that delivers a venom capable of causing severe pain to humans. The unique features of the platypus make it an important subject in the study of evolutionary biology and a recognisable and iconic symbol of Australia; it has appeared as a mascot at national events and is featured on the reverse of its 20-cent coin. The platypus is the animal emblem of the state of New South Wales.

Until the early 20th century, it was hunted for its fur, but it is now protected throughout its range. Although captive breeding programs have had only limited success and the platypus is vulnerable to the effects of pollution, it is not under any immediate threat.

This subject is absolutly not talking about platypus.

Chapter 3

Tutorial: Arrays

- So far, you have used simple types in C: `char`, `char`, etc. Let's start using more complex types! We'll begin with **arrays**, then **structures**.
- An **array** is a series of objects in the same type, all located contiguously in memory of the computer : an `array` of `int`, an `array` of `char`, etc...
- An array can be directly created using the following syntax :

main.c

```
int main(void)
{
    int my_int_tab[42];
}
```

This creates an array of 42 `int` in memory. The content of each `int` is unknown, like any other creation of any variable.

- You can access to each array element individually :

main.c

```
int main(void)
{
    int my_int_tab[42];

    my_int_tab[0] = 5; //Access to the first element
    my_int_tab[41] = 7; //Access to the last element
}
```

- You can also manipulate them like a variable :

main.c

```
int main(void)
{
    int my_int_tab[42];
    int a;

    a = my_int_tab[21] + 1;
}
```

- Let's try ! Create a `C` file with an **array** of 5 `integers`. Fill it with values 0, 1, 2, 3, 4 using a loop. Print all elements to verify.
- Print the same array from last element to first. Use another loop going backwards.
- You have already met arrays: when you define a string using the double quotes. "Hello" creates an **array** of 6 `char` (including the final 0), and automatically fills each char with the ascii code of each letter from the string.
- Arrays and pointers are closely related. Create an **array** of 3 `long integers` and a `pointer` to `long`. Fill the array using only the pointer (not the array name directly). Print the results.

Not sure where to start here ? Check your understanding with your neighbor, and dive in !

- Once you have your `array`, write two functions: one to fill an **array** of `integers`, another to display it. Both should receive the `array` as parameter. Test from `main`.

Chapter 4

Tutorial: Structures

- An array is cool, but comes with one caveat: instead of grouping elements with the same type, you sometimes want to group elements with different types. Like an `int`, then a `char`, then a `long`.
- If for standard `C` types, you already have a specific type name and you can directly create a variable of this type, there is not -yet- a type name for our own purpose (1 `int`, 1 `char`, 1 `long`). So creating a structure in `C` will come in 2 steps: first you need to define the new type. Then you can create elements using this new type (as many as you want, just like you can have multiple `int` or `char` in your code).
- So let's use **structures** to group different types together. Define a **structure** outside any function:

Description

```
struct s_learner {
    int age;
    char grade;
    long id;
};
```

Create a variable of this type: `struct s_learner learner1;`. Fill each variable like: `learner1.age = 20;`. Print all member values.

Defining a struct doesn't reserve memory - it just creates a new type name.

- Create 2 variables of the same struct type. Fill them with different values. Print both to verify each has independent member values.

When creating several variables, do you think it use the same RAM space for each member, or than RAM space will be reserved to store each of your structures?

- Create an array of 3 `struct` elements. Fill all members of all elements. Print each one

showing: `array index`, `age`, `grade`, and `id`.

- Experiment with `struct pointers`. Create a pointer `ptr` to your `struct`, assign it a `struct` address, then access members using `ptr->age` syntax.
- Feel free to experiment ! You can put `arrays` inside `structs`, `structs` inside `structs`, and even pointers to the same `struct type` inside a `struct`.

Chapter 5

Exercise 0: ft_point.h

	Exercise: 0	
ft_point.h		
Directory: *ex0/*		
Files to Submit: `ft_point.h`		
Authorized: `None`		

- Create a file `ft_point.h` that'll compile the following main:

main.c

```c
#include "ft_point.h"

void set_point(t_point *point)
{
    point->x = 42;
    point->y = 21;
}

int main(void)
{
    t_point point;

    set_point(&point);
    return (0);
}
```

Chapter 6

Exercise 1: ft_strs_to_tab

	Exercise: 1	
ft_strs_to_tab		
Directory: *ex1/*		
Files to Submit: `ft_strs_to_tab.c`		
Authorized: `malloc, free`		

- Create a function that takes an array of string as argument and the size of this array.

Prototype:

```
struct s_stock_str *ft_strs_to_tab(int ac, char **av);
```

- It will transform each element of av into a structure.
- The structure will be defined in the `ft_stock_str.h` file that we will provide, like this:

ft_stock_str.h

```
typedef struct   s_stock_str
{
    int  size;
    char *str;
    char *copy;
}                t_stock_str;
```

  - `size` being the length of the string;
  - `str` being the string;
  - `copy` being a copy of the string;

- It should keep the order of av.

- The returned array should allocated in memory and its last element's `str` set to 0, this will mark the end of the array.
- It should return a NULL pointer if an error occurs.

Chapter 7

Exercise 2: ft_show_tab

	Exercise: 2	
ft_show_tab		
Directory: *ex2/*		
Files to Submit: `ft_show_tab.c`		
Authorized: `write`		

- Create a function that displays the content of the array created by the previous function.
- Here's how it should be prototyped:

Prototype:

```
void ft_show_tab(struct s_stock_str *par);
```

- The structure will be the same as the previous exercise and will be defined in the `ft_stock_str.h` file.
- For each element, we'll display:
- the string followed by a '\n'
- the size followed by a '\n'
- the copy of the string (that could have been modified) followed by a '\n'
- We'll test your function with our `ft_strs_to_tab` (previous exercise). Make it work according to this!

Chapter 8

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.