



C Piscine

C 12

Summary: This document is the subject for the module C 12 of the C Piscine @ 42.

Version: 9.0

Contents

I	Foreword	2
II	Instructions	4
III	AI Instructions	6
IV	Exercise 00 : ft_create_elem	8
V	Exercise 01 : ft_list_push_front	9
VI	Exercise 02 : ft_list_size	10
VII	Exercise 03 : ft_list_last	11
VIII	Exercise 04 : ft_list_push_back	12
IX	Exercise 05 : ft_list_push_strs	13
X	Exercise 06 : ft_list_clear	14
XI	Exercise 07 : ft_list_at	15
XII	Exercise 08 : ft_list_reverse	16
XIII	Exercise 09 : ft_list_FOREACH	17
XIV	Exercise 10 : ft_list_FOREACH_if	18
XV	Exercise 11 : ft_list_find	19
XVI	Exercise 12 : ft_list_remove_if	20
XVII	Exercise 13 : ft_list_merge	21
XVIII	Exercise 14 : ft_list_sort	22
XIX	Exercise 15 : ft_list_reverse_fun	23
XX	Exercise 16 : ft_sorted_list_insert	24
XXI	Exercise 17 : ft_sorted_list_merge	25
XXII	Submission and peer-evaluation	26

Chapter I

Foreword

SPOILER ALERT

DON'T READ THE NEXT PAGE

You've been warned.

- In **Star Wars**, Dark Vador is Luke's Father.
- In **The Usual Suspects**, Verbal is Keyser Soze.
- In **Fight Club**, Tyler Durden and the narrator are the same person.
- In **The Sixth Sense**, Bruce Willis has been dead since the beginning.
- In **The others**, the inhabitants of the house are ghosts and vice-versa.
- In **Bambi**, Bambi's mother dies.
- In **The Village**, monsters are the villagers and the movie actually takes place in our time.
- In **Harry Potter**, Dumbledore dies.
- In **Planet of apes**, the movie takes place on earth.
- In **Game of thrones**, Robb Stark and Joffrey Baratheon die on their wedding day.
- In **Twilight**, Vampires shine under the sun.
- In **Stargate SG-1, Season 1, Episode 18**, O'Neill and Carter are in Antarctica.
- In **The Dark Knight Rises**, Miranda Tate is Talia Al'Gul.
- In **Super Mario Bros**, The princess is in another castle.

Chapter II

Instructions

- Only this page serves as your reference, do not trust rumors.
- Watch out! This document may change before submission.
- Ensure you have the appropriate permissions on your files and directories.
- You must follow the **submission procedures** for all your exercises.
- Your exercises will be checked and graded by your fellow classmates.
- Additionally, your exercises will be evaluated by a program called **Moulinette**.
- **Moulinette** is meticulous and strict in its assessment. It is fully automated, and there is no way to negotiate with it. To avoid unpleasant surprises, be as thorough as possible.
- **Moulinette** is not open-minded. If your code does not adhere to the Norm, it won't attempt to understand it. **Moulinette** relies on a program called **norminette** to check if your files comply with the Norm. TL;DR: Submitting work that doesn't pass **norminette**'s check makes no sense.
- These exercises are arranged in order of difficulty, from easiest to hardest. We **will not** consider a successfully completed harder exercise if an easier one is not fully functional.
- Using a forbidden function is considered cheating. Cheaters receive a grade of **-42**, which is non-negotiable.
- You only need to submit a **main()** function if we specifically ask for a **program**.
- **Moulinette** compiles with the following flags: **-Wall -Wextra -Werror**, using **cc**.
- If your program does not compile, you will receive a grade of **0**.
- You **cannot** leave **any** additional file in your directory beyond those specified in the assignment.
- Have a question? Ask the peer on your right. If not, try the peer on your left.

- Your reference guide is called **Google / man / the Internet / ...**
- Check the "C Piscine" section of the forum on the intranet or the Piscine on Slack.
- Carefully examine the examples. They may contain crucial details that are not explicitly stated in the assignment...
- By Odin, by Thor! Use your brain!!!
- For the following exercises, you have to use the following structure:

```
typedef struct          s_list
{
    struct s_list      *next;
    void               *data;
}                      t_list;
```

- You'll have to include this structure in a file **ft_list.h** and submit it for each exercise.
- From exercise 01 onward, we'll use our **ft_create_elem**, so make arrangements (it could be useful to have its prototype in a file **ft_list.h**...).

Chapter III

AI Instructions

● 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 IV

Exercise 00 : ft_create_elem

	Exercise 00
	ft_create_elem
Turn-in directory:	<i>ex00/</i>
Files to turn in:	<code>ft_create_elem.c, ft_list.h</code>
Allowed functions:	<code>malloc</code>

- Create the function `ft_create_elem`, which creates a new element of `t_list` type.
- It should assign `data` to the given argument and `next` to `NULL`.
- Here is how it should be prototyped:

```
t_list *ft_create_elem(void *data);
```

Chapter V

Exercise 01 : ft_list_push_front

	Exercise 01
	ft_list_push_front
Turn-in directory:	<i>ex01/</i>
Files to turn in:	<code>ft_list_push_front.c, ft_list.h</code>
Allowed functions:	<code>ft_create_elem</code>

- Create the function `ft_list_push_front`, which adds a new element of type `t_list` to the beginning of the list.
- It should assign `data` to the given argument.
- If necessary, it will update the pointer at the beginning of the list.
- Here is how it should be prototyped:

```
void        ft_list_push_front(t_list **begin_list, void *data);
```

Chapter VI

Exercise 02 : ft_list_size

	Exercise 02
	ft_list_size
Turn-in directory:	<i>ex02/</i>
Files to turn in:	ft_list_size.c, ft_list.h
Allowed functions:	None

- Create the function `ft_list_size`, which returns the number of elements in the list.
- Here is how it should be prototyped:

```
int ft_list_size(t_list *begin_list);
```

Chapter VII

Exercise 03 : ft_list_last

	Exercise 03
	ft_list_last
Turn-in directory:	<i>ex03/</i>
Files to turn in:	<code>ft_list_last.c, ft_list.h</code>
Allowed functions:	None

- Create the function `ft_list_last`, which returns the last element of the list.
- Here is how it should be prototyped:

```
t_list *ft_list_last(t_list *begin_list);
```

Chapter VIII

Exercise 04 : ft_list_push_back

	Exercise 04
	ft_list_push_back
Turn-in directory:	<i>ex04/</i>
Files to turn in:	<code>ft_list_push_back.c, ft_list.h</code>
Allowed functions:	<code>ft_create_elem</code>

- Create the function `ft_list_push_back`, which adds a new element of `t_list` type at the end of the list.
- It should assign `data` to the given argument.
- If necessary, it will update the pointer at the beginning of the list.
- Here is how it should be prototyped:

```
void ft_list_push_back(t_list **begin_list, void *data);
```

Chapter IX

Exercise 05 : ft_list_push_strs

	Exercise 05
	ft_list_push_strs
Turn-in directory:	<i>ex05/</i>
Files to turn in:	<code>ft_list_push_strs.c, ft_list.h</code>
Allowed functions:	<code>ft_create_elem</code>

- Create the function `ft_list_push_strs`, which creates a new list that includes all the strings pointed to by the elements in `strs`.
- `size` is the size of `strs`.
- The first element should be at the end of the list.
- The first link's address in the list is returned.
- Here is how it should be prototyped:

```
t_list *ft_list_push_strs(int size, char **strs);
```

Chapter X

Exercise 06 : ft_list_clear

	Exercise 06
	ft_list_clear
Turn-in directory:	<i>ex06/</i>
Files to turn in:	ft_list_clear.c, ft_list.h
Allowed functions:	free

- Create the function `ft_list_clear`, which removes and frees all links from the list.
- `free_fct` is used to free each data.
- Here is how it should be prototyped:

```
void ft_list_clear(t_list *begin_list, void (*free_fct)(void *));
```

Chapter XI

Exercise 07 : ft_list_at

	Exercise 07
	ft_list_at
Turn-in directory:	<i>ex07/</i>
Files to turn in:	ft_list_at.c, ft_list.h
Allowed functions:	None

- Create the function `ft_list_at`, which returns the Nth element of the list, knowing that the first element of the list is when `nbr` equals 0.
- In case of error, it should return a null pointer.
- Here is how it should be prototyped:

```
t_list *ft_list_at(t_list *begin_list, unsigned int nbr);
```

Chapter XII

Exercise 08 : `ft_list_reverse`

	Exercise 08
	<code>ft_list_reverse</code>
	Turn-in directory: <code>ex08/</code>
	Files to turn in: <code>ft_list_reverse.c</code>
	Allowed functions: None

- Create the function `ft_list_reverse`, which reverses the order of a list's elements. The value of each element must remain the same.
- Beware that in this function, we will use our own `ft_list.h`.
- Here is how it should be prototyped:

```
void ft_list_reverse(t_list **begin_list);
```

Chapter XIII

Exercise 09 : ft_list_FOREACH

	Exercise 09
	ft_list_FOREACH
	Turn-in directory: <i>ex09/</i>
	Files to turn in: ft_list_FOREACH.c, ft_list.h
	Allowed functions: None

- Create the function `ft_list_FOREACH`, which applies the function given as an argument to each of the list's elements.
- `f` should be applied in the same order as the list.
- Here is how it should be prototyped:

```
void ft_list_FOREACH(t_list *begin_list, void (*f)(void *));
```

- The function pointed to by `f` will be used as follows:

```
(*f)(list_ptr->data);
```

Chapter XIV

Exercise 10 : ft_list_FOREACH_if

	Exercise 10
	ft_list_FOREACH_if
	Turn-in directory: <i>ex10/</i>
	Files to turn in: <i>ft_list_FOREACH_if.c, ft_list.h</i>
	Allowed functions: None

- Create the function `ft_list_FOREACH_if`, which applies the function given as an argument to some of the list's elements.
- Only apply the function to the elements when `cmp` with `data_ref` returns 0.
- `f` should be applied in the same order as the list.
- Here is how it should be prototyped:

```
void ft_list_FOREACH_if(t_list *begin_list, void (*f)(void *), void  
*data_ref, int (*cmp)());
```

- Functions pointed to by `f` and by `cmp` will be used as follows:

```
(*f)(list_ptr->data);  
(*cmp)(list_ptr->data, data_ref);
```



For example, the function `cmp` could be `ft_strcmp...`

Chapter XV

Exercise 11 : ft_list_find

	Exercise 11
	ft_list_find
	Turn-in directory: <i>ex11/</i>
	Files to turn in: ft_list_find.c , ft_list.h
	Allowed functions: None

- Create the function `ft_list_find` which returns the address of the first element's data where comparing it to `data_ref` with `cmp` causes `cmp` to return 0.
- Here's how it should be prototyped:

```
t_list *ft_list_find(t_list *begin_list, void *data_ref, int (*cmp)());
```

- The function pointed to by `cmp` will be used as follows:

```
(*cmp)(list_ptr->data, data_ref);
```

Chapter XVI

Exercise 12 : `ft_list_remove_if`

	Exercise 12
	<code>ft_list_remove_if</code>
	Turn-in directory: <code>ex12/</code>
	Files to turn in: <code>ft_list_remove_if.c</code> , <code>ft_list.h</code>
	Allowed functions: <code>free</code>

- Create the function `ft_list_remove_if` which removes from the list all elements whose data, when compared to `data_ref` using `cmp`, causes `cmp` to return 0.
- The data from an element to be erased should be freed using `free_fct`.
- Here's how it should be prototyped:

```
void ft_list_remove_if(t_list **begin_list, void *data_ref, int (*cmp)(), void (*free_fct)(void *));
```

- The functions pointed to by `cmp` and `free_fct` will be used as follows:

```
(*cmp)(list_ptr->data, data_ref);  
(*free_fct)(list_ptr->data);
```

Chapter XVII

Exercise 13 : ft_list_merge

	Exercise 13
	ft_list_merge
Turn-in directory:	<i>ex13/</i>
Files to turn in:	<code>ft_list_merge.c, ft_list.h</code>
Allowed functions:	None

- Create the function `ft_list_merge` which places elements of a list `begin2` at the end of another list `begin1`.
- Element creation is not authorised.
- Here's how it should be prototyped:

```
void ft_list_merge(t_list **begin_list1, t_list *begin_list2);
```

Chapter XVIII

Exercise 14 : ft_list_sort

	Exercise 14
	ft_list_sort
	Turn-in directory: <i>ex14/</i>
	Files to turn in: ft_list_sort.c, ft_list.h
	Allowed functions: None

- Create the function **ft_list_sort** which sorts the list's elements in ascending order by comparing two elements and their data using a comparison function.
- Here's how it should be prototyped:

```
void ft_list_sort(t_list **begin_list, int (*cmp)());
```

- The function pointed to by **cmp** will be used as follows:

```
(*cmp)(list_ptr->data, list_other_ptr->data);
```



`cmp` could be for instance `ft_strcmp`.

Chapter XIX

Exercise 15 : ft_list_reverse_fun

	Exercise 15
	ft_list_reverse_fun
	Turn-in directory: <i>ex15/</i>
	Files to turn in: ft_list_reverse_fun.c, ft_list.h
	Allowed functions: None

- Create the function `ft_list_reverse_fun` which reverses the order of the elements in the list.
- Here's how it should be prototyped:

```
void ft_list_reverse_fun(t_list *begin_list);
```

Chapter XX

Exercise 16 : ft_sorted_list_insert

	Exercise 16
	ft_sorted_list_insert
	Turn-in directory: <i>ex16/</i>
	Files to turn in: ft_sorted_list_insert.c , ft_list.h
	Allowed functions: ft_create_elem

- Create the function `ft_sorted_list_insert` which creates a new element and inserts it into a list sorted so that it remains sorted in ascending order.
- Here's how it should be prototyped:

```
void ft_sorted_list_insert(t_list **begin_list, void *data, int (*cmp)());
```

- Function pointed by `cmp` will be used as follows:

```
(*cmp)(list_ptr->data, list_other_ptr->data);
```

Chapter XXI

Exercise 17 : ft_sorted_list_merge

	Exercise 17
	ft_sorted_list_merge
	Turn-in directory: <i>ex17/</i>
	Files to turn in: <i>ft_sorted_list_merge.c, ft_list.h</i>
	Allowed functions: None

- Create the function `ft_sorted_list_merge` which integrates the elements of a sorted list `begin2` in another sorted list `begin1`, so that `begin1` remains sorted by ascending order.
- Here's how it should be prototyped:

```
void ft_sorted_list_merge(t_list **begin_list1, t_list *begin_list2, int (*cmp)());
```

- Function pointed by `cmp` will be used as follows:

```
(*cmp)(list_ptr->data, list_other_ptr->data);
```

Chapter XXII

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. Make sure to double-check the filenames to ensure they are correct.



You must submit only the files specified in the project instructions.