Version: 1.0

Selection

C-Linked-Lists

Summary

Implement functions to create, modify, and manipulate singly linked lists in C, handling insertion, deletion, and sorting operations.

#C #Data-Structures #Linked-Lists

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

1 Instructions 1

2 Foreword 5

3 Information for this project 7

4 Exercice 0: ft_create_elem 8

5 Exercice 1: ft_list_push_front 9

6 Exercice 2: ft_list_size 10

7 Exercice 3: ft_list_last 11

8 Exercice 4: ft_list_push_back 12

9 Exercice 5: ft_list_push_strs 13

10 Exercice 6: ft_list_clear 14

11 Exercice 7: ft_list_at 15

12 Exercice 8: ft_list_reverse 16

13 Exercice 9: ft_list_foreach 17

14 Exercice 10: ft_list_foreach_if 18

15 Exercice 11: ft_list_find 19

16 Exercice 12: ft_list_remove_if 20

17 Exercice 13: ft_list_merge 21

18 Exercice 14: ft_list_sort 22

19 Exercice 15: ft_list_reverse_fun 23

20 Exercice 16: ft_sorted_list_insert 24

21 Exercice 17: ft_sorted_list_merge 25

22 Submission and peer-evaluation 26

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.

© 2026 Association 42. - All rights reserved

- 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.

© 2026 Association 42. - All rights reserved

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.

© 2026 Association 42. - All rights reserved

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.

© 2026 Association 42. - All rights reserved

Chapter 2

Foreword

SPOILER ALERT
DON'T READ THE NEXT PAGE

© 2026 Association 42. - All rights reserved

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 `Sixth Sens`, Bruce Willis is 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 Antartica.
- In `The Dark Knight Rises`, Miranda Tate is Talia Al'Gul.
- In `Super Mario Bros`, The princess is in another castle.

© 2026 Association 42. - All rights reserved

Chapter 3

Information for this project

For the following exercises, you have to use the following structure :

ft_list.h

```
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...).

© 2026 Association 42. - All rights reserved

Chapter 4

Exercice 0: ft_create_elem

	Exercise: 0	
ft_create_elem		
Directory: *ex0/*		
Files to Submit: `ft_create_elem.c, ft_list.h`		
Authorized: `malloc`		

- 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.

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 5

Exercice 1: ft_list_push_front

	Exercise: 1	
ft_list_push_front		
Directory: *ex1/*		
Files to Submit: `ft_list_push_front.c, ft_list.h`		
Authorized: `ft_create_elem`		

- 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'll update the pointer at the beginning of the list.

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 6

Exercice 2: ft_list_size

	Exercise: 2	
ft_list_size		
Directory: *ex2/*		
Files to Submit: `ft_list_size.c, ft_list.h`		
Authorized: `None`		

- Create the function `ft_list_size` which returns the number of elements in the list.

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 7

Exercice 3: ft_list_last

	Exercise: 3	
ft_list_last		
Directory: *ex3/*		
Files to Submit: `ft_list_last.c, ft_list.h`		
Authorized: `None`		

- Create the function `ft_list_last` which returns the last element of the list.

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 8

Exercice 4: ft_list_push_back

	Exercise: 4	
ft_list_push_back		
Directory: *ex4/*		
Files to Submit: `ft_list_push_back.c, ft_list.h`		
Authorized: `ft_create_elem`		

- 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'll update the pointer at the beginning of the list.

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 9

Exercice 5: ft_list_push_strs

	Exercise: 5	
ft_list_push_strs		
Directory: *ex*5/		
Files to Submit: `ft_list_push_strs.c, ft_list.h`		
Authorized: `ft_create_elem`		

- Create the function `ft_list_push_strs` which creates a new list that includes all the string pointed by the element 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.

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 10

Exercice 6: ft_list_clear

	Exercise: 6	
ft_list_clear		
Directory: *ex6/*		
Files to Submit: `ft_list_clear.c, ft_list.h`		
Authorized: `free`		

- Create the function `ft_list_clear` which removes and frees all links from the list.
- `free_fct` is used to free each `data`

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 11

Exercice 7: ft_list_at

	Exercise: 7	
ft_list_at		
Directory: *ex7/*		
Files to Submit: `ft_list_at.c, ft_list.h`		
Authorized: `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` equal 0.
- In case of error, it should return a null pointer.

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 12

Exercice 8: ft_list_reverse

	Exercise: 8	
ft_list_reverse		
Directory: *ex8/*		
Files to Submit: `ft_list_reverse.c`		
Authorized: `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 in that function we will use our own ft_list.h

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 13

Exercice 9: ft_list_foreach

	Exercise: 9	
ft_list_foreach		
Directory: *ex9/*		
Files to Submit: `ft_list_foreach.c, ft_list.h`		
Authorized: `None`		

- Create the function `ft_list_foreach` which applies the function given as argument to each of the list's elements.
- `f` should be applied in the same order as the list.

Prototype:

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

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

Usage:

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

© 2026 Association 42. - All rights reserved

Chapter 14

Exercice 10: ft_list_foreach_if

	Exercise: 10	
ft_list_foreach_if		
Directory: *ex10/*		
Files to Submit: `ft_list_foreach_if.c, ft_list.h`		
Authorized: `None`		

- Create the function `ft_list_foreach_if` which applies the function given as argument to some of the list's elements.
- Only apply the function to the elements when `cmp` with `data_ref`, `cmp` returns 0
- `f` should be applied in the same order as the list.

Prototype:

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

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

Usage:

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

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

© 2026 Association 42. - All rights reserved

Chapter 15

Exercice 11: ft_list_find

	Exercise: 11	
ft_list_find		
Directory: *ex11/*		
Files to Submit: `ft_list_find.c, ft_list.h`		
Authorized: `None`		

- Create the function `ft_list_find` which returns the address of the first element's data compared to `data_ref` with `cmp` makes `cmp` to return 0.

Prototype:

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

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

Usage:

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

© 2026 Association 42. - All rights reserved

Chapter 16

Exercice 12: ft_list_remove_if

	Exercise: 12	
ft_list_remove_if		
Directory: *ex12/*		
Files to Submit: `ft_list_remove_if.c, ft_list.h`		
Authorized: `free`		

- Create the function `ft_list_remove_if`, which removes from the list all elements whose data, when compared to `data_ref` using `cmp`, makes `cmp` return 0.
- The data from an element to be erased should be freed using `free_fct`.

Prototype:

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

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

Usage:

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

© 2026 Association 42. - All rights reserved

Chapter 17

Exercice 13: ft_list_merge

	Exercise: 13	
ft_list_merge		
Directory: *ex*13/		
Files to Submit: `ft_list_merge.c, ft_list.h`		
Authorized: `None`		

- Create the function `ft_list_merge`, which appends all elements of the list `begin_list2` at the end of another list `begin_list1`.
- Element creation is not allowed.

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 18

Exercice 14: ft_list_sort

	Exercise: 14	
ft_list_sort		
Directory: *ex*14/		
Files to Submit: `ft_list_sort.c, ft_list.h`		
Authorized: `None`		

- Create the function `ft_list_sort`, which sorts the elements of the list in ascending order by comparing their data using a function.

Prototype:

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

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

Usage:

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

`cmp` could be, for instance, `ft_strcmp`.

© 2026 Association 42. - All rights reserved

Chapter 19

Exercice 15: ft_list_reverse_fun

	Exercise: 15	
ft_list_reverse_fun		
Directory: *ex*15/		
Files to Submit: `ft_list_reverse_fun.c, ft_list.h`		
Authorized: `None`		

- Create the function `ft_list_reverse_fun`, which reverses the order of the elements in the list.

Prototype:

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

© 2026 Association 42. - All rights reserved

Chapter 20

Exercice 16: ft_sorted_list_insert

	Exercise: 16	
ft_sorted_list_insert		
Directory: *ex16/*		
Files to Submit: `ft_sorted_list_insert.c, ft_list.h`		
Authorized: `ft_create_elem`		

- Create the function `ft_sorted_list_insert`, which creates a new element and inserts it into a list while ensuring the list remains sorted in ascending order.

Prototype:

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

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

Usage:

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

© 2026 Association 42. - All rights reserved

Chapter 21

Exercice 17: ft_sorted_list_merge

	Exercise: 17	
ft_sorted_list_merge		
Directory: *ex*17/		
Files to Submit: `ft_sorted_list_merge.c, ft_list.h`		
Authorized: `None`		

- Create the function `ft_sorted_list_merge`, which integrates the elements of a sorted list `begin_list2` into another sorted list `begin_list1`, ensuring that `begin_list1` remains sorted in ascending order.

Prototype:

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

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

Usage:

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

© 2026 Association 42. - All rights reserved

Chapter 22

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.

© 2026 Association 42. - All rights reserved