C Piscine

Day 11

Staff 42 **pedago@42.fr**

Summary: *This document is the subject for Day11 of the C Piscine @ 42.*

Contents

Chapter 1

Instructions

- The exercises are carefully laid out in order of difficulty, from easiest to hardest. An exercise is only graded if all previous ones are correct. In other words: the grading for a day stops at the first mistake.

- Be mindful of the submission procedures indicated at the start of every exercise.

- Your exercises will be checked and graded by your fellow classmates.

- On top of that, 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. Be as thorough as possible!

- Moulinette relies on a program called **norminette** to check if your files respect the Norm. An exercise containing files that do not respect the Norm will be graded 0.

- Using a forbidden function is considered cheating. Cheaters get -42, and this grade is non-negotiable.

- If **ft_putchar()** is an authorized function, we will compile your code with our **ft_putchar.c**.

- 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 **gcc**.

- If your program doesn't compile, it will be graded 0.

- You should not leave any additional file in your directory than those specified in the subject.

> **For this day, norminette is launched without any particular flag!**

(i) The forewords are entirely unrelated to the subjects and can safely be ignored.

Chapter 2

Topics

Today, you will have to learn about:

- Linked lists

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 own ft_create_elem, so make sure to include its prototype in **ft_list.h**.

Chapter 3

Foreword

SPOILER ALERT DON'T READ THE NEXT PAGE

<u>You've been warned.</u>

- 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.

Chapter 4

Exercice 00 : ft_create_elem

Turn-in directory : ex00/

Files to turn in: ft_create_elem.c, ft_list.h

Allowed functions: malloc

- Create the function **ft_create_elem** which creates a new element of **t_list** type.

- It should assign the given argument to **data** and NULL to **next**.

- Here's how it should be prototyped:

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

Chapter 5

Exercice 01 : ft_list_push_back

Turn-in directory : ex01/

Files to turn in: ft_list_push_back.c, ft_list.h

Allowed functions: 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 the given argument to **data**.

- If necessary, it'll update the pointer at the beginning of the list.

- Here's how it should be prototyped:

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

Chapter 6

Exercice 02 : ft_list_push_front

Turn-in directory : ex02/

Files to turn in: ft_list_push_front.c, ft_list.h

Allowed functions: 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 the given argument to **data**.

- If necessary, it'll update the pointer at the beginning of the list.

- Here's how it should be prototyped:

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

Chapter 7

Exercice 03 : ft_list_size

Turn-in directory : ex03/

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's how it should be prototyped:

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

Chapter 8

Exercice 04 : ft_list_last

Turn-in directory : ex04/

Files to turn in: ft_list_last.c, ft_list.h

Allowed functions: None

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

- Here's how it should be prototyped:

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

Chapter 9

Exercice 05 : ft_list_push_params

Turn-in directory : ex05/

Files to turn in: ft_list_push_params.c, ft_list.h

Allowed functions: ft_create_elem

- Create the function **ft_list_push_params** which creates a new list that includes command-line arguments.

- The first argument should be at the end of the list.

- The first link's address in the list is returned.

- Here's how it should be prototyped:

```
t_list *ft_list_push_params(int ac, char **av);
```

Chapter 10

Exercice 06 : ft_list_clear

Turn-in directory : ex06/

Files to turn in: ft_list_clear.c, ft_list.h

Allowed functions: free

- Create the function **ft_list_clear** which clears all links from the list.

- It'll then set the given argument to NULL.

- Here's how it should be prototyped:

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

Chapter 11

Exercice 07 : ft_list_at

Turn-in directory : ex07/

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.

- In case of error, it should return a null pointer.

- Here's how it should be prototyped:

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

Chapter 12

Exercice 08 : ft_list_reverse

Turn-in directory : ex08/

Files to turn in: ft_list_reverse.c, ft_list.h

Allowed functions: None

- Create the function **ft_list_reverse** which reverses the order of a list's elements. You may only use pointers related stuff.

- Here's how it should be prototyped:

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

Chapter 13

Exercice 09 : ft_list_foreach

Turn-in directory : ex09/

Files to turn in: ft_list_foreach.c, ft_list.h

Allowed functions: None

- Create the function **ft_list_foreach** which applies a function given as argument to the information within each of the list's links.

- Here's how it should be prototyped:

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

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

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

Chapter 14

Exercice 10 : ft_list_foreach_if

Turn-in directory : ex10/

Files to turn in: ft_list_foreach_if.c, ft_list.h

Allowed functions: None

- Create the function **ft_list_foreach_if** which applies a function given as argument to the information held in some links of the list. A reference information as well as a comparative function should allow us to select the right links of the list: those that are "equal" to the reference information.

- Here's how it should be prototyped:

```
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 :

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

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

Chapter 15

Exercice 11 : ft_list_find

Turn-in directory : ex11/

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 link, whose data is "equal" to the reference data.

- Here's how it should be prototyped:

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

Chapter 16

Exercice 12 : ft_list_remove_if

Turn-in directory : ex12/

Files to turn in: ft_list_remove_if.c, ft_list.h

Allowed functions: free

- Create the function **ft_list_remove_if** which removes from the list all elements whose data is "equal" to the reference data.

- Here's how it should be prototyped:

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

Chapter 17

Exercice 13 : ft_list_merge

Turn-in directory : ex13/

Files to turn in: ft_list_merge.c, ft_list.h

Allowed functions: None

- Create the function **ft_list_merge** which places elements of a list **begin_list2** at the end of an other list **begin_list1**.

- 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 18

Exercice 14 : ft_list_sort

Turn-in directory : ex14/

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 contents by ascending order by comparing two links thanks to a function that can compare the data held in those two links.

- Here's how it should be prototyped:

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

> La fonction cmp pourrait être par exemple ft_strcmp.

Chapter 19

Exercice 15 : ft_list_reverse_fun

Turn-in directory : ex15/

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 of the list. You may only use pointers related stuff.

- Here's how it should be prototyped:

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

Chapter 20

Exercice 16 : ft_sorted_list_insert

Turn-in directory : ex16/

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
  sortend in ascending order.

- Here's how it should be prototyped:

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

Chapter 21

Exercice 17 : ft_sorted_list_merge

Turn-in directory : ex17/

Files to turn in: ft_sorted_list_merge.c, ft_list.h

Allowed functions: None

- Create the function **ft_sorted_list_merge** which integrates the elements of a sorted list **begin_list2** in another sorted list **begin_list1**, so that **begin_list1** 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)());
```