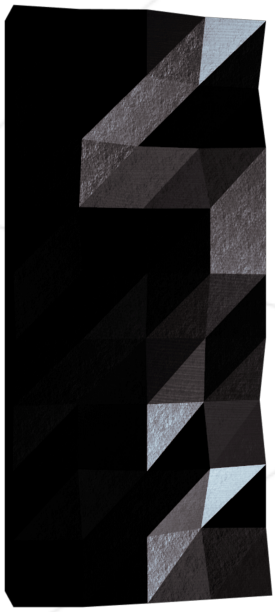


C Piscine

C 07

Staff 42 pedago@42.fr

Summary: This document is the subject for the module C 07 of the C Piscine @ 42.

Contents

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change before 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.
- 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. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of 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` take into account 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 `gcc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional file in your directory 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 "C Piscine" part of the forum on the intranet, or 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 !!!

`Norminette must be launched with the -R CheckForbiddenSourceHeader flag. Moulinette will use it too.`

Chapter II

Foreword

Morty: Rick!

Rick: Uhp-uhp-uhp! Morty, keep your hands off your ding-dong! It's the only way we can speak freely. Look around you, Morty. Do you really think this wuh-world is real? You'd have to be an idiot not to notice all the sloppy details. Look, that guy's putting a bun between two hot dogs.

Morty: I dunno, Rick, I mean, I've seen people do that before.

Rick: Well, look at that old lady. She's-she's walking a cat on a leash.

Morty: Uh, Mrs. Spencer does that all the time, Rick.

Rick: Look, I-I-I don't want to hear about Mrs. Spencer, Morty! She's an idiot! All right, all right, there. Wh-what about that, Morty?

Morty: Okay, okay, you got me on that one.

Rick: Oh, really, Morty? Are you sure you haven't seen that somewhere in real life before?

Morty: No, no, I haven't seen that. I mean, why would a Pop-Tart want to live inside a toaster, Rick? I mean, th-that would be like the scariest place for them to live. Y'know what I mean?

Rick: You're missing the point, Morty. Why would he drive a smaller toaster with wheels? I mean, does your car look like a smaller version of your house? No.

Morty: So, why are they doing this? W-what do they want?

Rick: Well, that would be obvious to you, Morty, if you'd been paying attention.
[an ambulance drives past Rick and Morty and stops; open back doors]
Paramedic: We got the President of the United States in here! We need 10cc of concentrated dark matter, stat, or he'll die!

Morty: Concentrated dark matter? They were asking about that in class.

Rick: Yeah, it’s a special fuel I invented to travel through space faster than anybody else. These Zigerions are always trying to scam me out of my secrets, but they made a big mistake this time, Morty. They dragged you into this. Now they’re gonna pay!

Morty: What do you- w-w-what are we gonna do?

Rick: We’re gonna scam the scammers, Morty. And we’re gonna take ’em for everything they’ve got.

The following exercices will be easier to complete if you are a fan of "Rick and Morty"

Chapter III

Exercise 00 : ft_strdup

	Exercise 00
ft_strdup	
Turn-in directory : $ex00/$	
Files to turn in : `ft_strdup.c`	
Allowed functions : `malloc`	

- Reproduce the behavior of the function `strdup` (man strdup).
- Here's how it should be prototyped :

```
char    *ft_strdup(char *src);
```

Chapter IV

Exercise 01 : ft_range

	Exercise 01
ft_range	
Turn-in directory : *ex*01/	
Files to turn in : `ft_range.c`	
Allowed functions : `malloc`	

- Create a function `ft_range` which returns an array of`ints`. This `int` array should contain all values between `min` and `max`.
- `Min` included - `max` excluded.
- Here's how it should be prototyped :

```
int    *ft_range(int min, int max);
```

- If `min`́value is greater or equal to `max`'s value, a null pointer should be returned.

Chapter V

Exercise 02 : ft_ultimate_range

	Exercise 02
ft_ultimate_range	
Turn-in directory : *ex*02/	
Files to turn in : `ft_ultimate_range.c`	
Allowed functions : `malloc`	

- Create a function `ft_ultimate_range` which allocates and assigns an array of `ints`. This `int` array should contain all values between `min` and `max`.
- `Min` included - `max` excluded.
- Here's how it should be prototyped :

```
int    ft_ultimate_range(int **range, int min, int max);
```

- The size of `range` should be returned (or -1 on error).
- If the value of `min` is greater or equal to `max`'s value, `range` will point on NULL and it should return 0.

Chapter VI

Exercice 03 : ft_strjoin

	Exercise 03
ft_strjoin	
Turn-in directory : $ex03/$	
Files to turn in : `ft_strjoin.c`	
Allowed functions : `malloc`	

- Write a function that will concatenate all the strings pointed by `strs` separated by `sep`.
- `size` is the number of strings in `strs`
- if `size` is 0, it should a freeable empty string.
- Here's how it should be prototyped :

```
char *ft_strjoin(int size, char **strs, char *sep);
```

Chapter VII

Exercise 04 : ft_convert_base

	Exercise 04
ft_convert_base	
Turn-in directory : *ex*04/	
Files to turn in : `ft_convert_base.c, ft_convert_base2.c`	
Allowed functions : `malloc, free`	

- Create a function that returns the result of the conversion of the string `nbr` from a base `base_from` to a base `base_to`.
- `nbr`, `base_from`, `base_to` may be not writable.
- `nbr` will follow the same rules as ft_atoi_base (from an other module). Beware of '+', '-' and whitespaces.
- The number represented by `nbr` must fit inside an `int`.
- If a base is wrong, NULL should be returned.
- The returned number must be prefix only by a single and uniq '-' if necessary, no whitespaces, no '+'.
- Here's how it should be prototyped :

```
char *ft_convert_base(char *nbr, char *base_from, char *base_to);
```

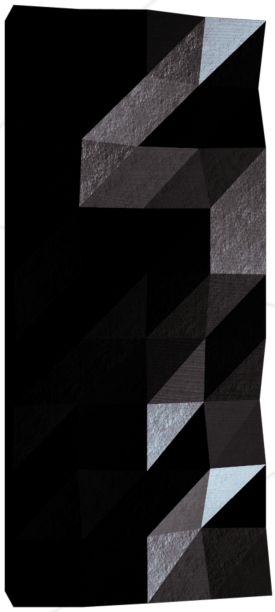
Chapter VIII

Exercise 05 : ft_split

	Exercise 05
ft_split	
Turn-in directory : $ex05/$	
Files to turn in : `ft_split.c`	
Allowed functions : `malloc`	

- Create a function that splits a string of character depending on another string of characters.
- You'll have to use each character from the string `charset` as a separator.
- The function returns an array where each element of the array contains the address of a string wrapped between two separators. The last element of that array should equal to 0 to indicate the end of the array.
- There cannot be any empty strings in your array. Get your own conclusions accordingly.
- The string given as argument won't be modifiable.
- Here's how it should be prototyped :

```
char **ft_split(char *str, char *charset);
```



C Piscine

C 08

Summary: This document is the subject for C 08 module of the C Piscine @ 42.

Contents

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change before 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.
- 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. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of 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` take into account 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 `gcc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional file in your directory 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 "C Piscine" part of the forum on the intranet, or 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 !!!

Chapter II

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 III

Exercise 00 : ft.h

	Exercise 00
ft.h	
Turn-in directory : $ex00/$	
Files to turn in : `ft.h`	
Allowed functions : `None`	

- Create your `ft.h` file.
- It contains all prototypes of functions :

```
void    ft_putchar(char c);
void    ft_swap(int *a, int *b);
void    ft_putstr(char *str);
int     ft_strlen(char *str);
int     ft_strcmp(char *s1, char *s2);
```

Chapter IV

Exercise 01 : ft_boolean.h

	Exercise 01
ft_boolean.h	
Turn-in directory : *ex*01/	
Files to turn in : `ft_boolean.h`	
Allowed functions : `None`	

- Create a `ft_boolean.h` file. It'll compile and run the following main appropriately :

```c
#include "ft_boolean.h"

void        ft_putstr(char *str)
{
        while (*str)
                write(1, str++, 1);
}

t_bool        ft_is_even(int nbr)
{
        return ((EVEN(nbr)) ? TRUE : FALSE);
}

int                main(int argc, char **argv)
{
        (void)argv;
        if (ft_is_even(argc - 1) == TRUE)
                ft_putstr(EVEN_MSG);
        else
                ft_putstr(ODD_MSG);
        return (SUCCESS);
}
```

- This program should display

```
I have an even number of arguments.
```

- ou

```
I have an odd number of arguments.
```

- followed by a line break when adequate.

`Norminette must be launched with the -R CheckForbiddenSourceHeader flag. Moulinette will use it too.`

Chapter V

Exercise 02 : ft_abs.h

	Exercise 02
ft_abs.h	
Turn-in directory : *ex*02/	
Files to turn in : `ft_abs.h`	
Allowed functions : `None`	

- Create a macro `ABS` which replaces its argument by it absolute value :

```
#define ABS(Value)
```

`Norminette must be launched with the -R CheckForbiddenSourceHeader flag. Moulinette will use it too.`

Chapter VI

Exercise 03 : ft_point.h

	Exercise 03
ft_point.h	
Turn-in directory : *ex*03/	
Files to turn in : `ft_point.h`	
Allowed functions : `None`	

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

```
#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 VII

Exercise 04 : ft_strs_to_tab

	Exercise 04
ft_strs_to_tab	
Turn-in directory : *ex*04/	
Files to turn in : `ft_strs_to_tab.c`	
Allowed functions : `malloc, free`	

- Create a function that takes an array of string as argument and the size of this array.
- Here's how it should be prototyped :

```
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 provided, like this :

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

  - `size` being the length of the string;
  - `str` being 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.
- We'll test your function with our `ft_show_tab` (next exercise). Make it work according to this !

Chapter VIII

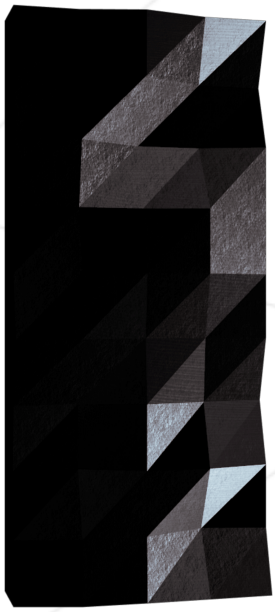
Exercise 05 : ft_show_tab

	Exercise 05
ft_show_tab	
Turn-in directory : $ex05/$	
Files to turn in : `ft_show_tab.c`	
Allowed functions : `write`	

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

```
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 !



C Piscine

C 09

Summary: This document is the subject for the module C 09 of the C Piscine @ 42.

Contents

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change before 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.
- 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. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of 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` take into account 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 `gcc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional file in your directory 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 "C Piscine" part of the forum on the intranet, or 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 !!!

`Norminette must be launched with the -R CheckForbiddenSourceHeader flag. Moulinette will use it too.`

Chapter II

Foreword

Dialog from the movie `The Big Lebowski`:

```
The Dude: Walter, ya know, it's Smokey, so his toe slipped over the line a little,
  big deal. It's just a game, man.
Walter Sobchak: Dude, this is a league game, this determines who enters the next
  round robin. Am I wrong? Am I wrong?
Smokey: Yeah, but I wasn't over. Gimme the marker Dude, I'm marking it 8.
Walter Sobchak: [pulls out a gun] Smokey, my friend, you are entering a world
  of pain.
The Dude: Walter...
Walter Sobchak: You mark that frame an 8, and you're entering a world of pain.
Smokey: I'm not...
Walter Sobchak: A world of pain.
Smokey: Dude, he's your partner...
Walter Sobchak: [shouting] Has the whole world gone crazy? Am I the only one
  around here who gives a shit about the rules? Mark it zero!
The Dude: They're calling the cops, put the piece away.
Walter Sobchak: Mark it zero!
[points gun in Smokey's face]
The Dude: Walter...
Walter Sobchak: [shouting] You think I'm fucking around here? Mark it zero!
Smokey: All right, it's fucking zero. Are you happy, you crazy fuck?
Walter Sobchak: ...It's a league game, Smokey.
```

Chapter III

Exercise 00 : libft

	Exercise 00
libft	
Turn-in directory : $ex00/$	
Files to turn in : `libft_creator.sh, ft_putchar.c, ft_swap.c, ft_putstr.c, ft_strlen.c, ft_strcmp.c`	
Allowed functions : `write`	

- Create your `ft` library. It'll be called `libft.a`.
- A shell script called `libft_creator.sh` will compile source files appropriately and will create your library.
- This library should contain all of the following functions :

```
void    ft_putchar(char c);
void    ft_swap(int *a, int *b);
void    ft_putstr(char *str);
int     ft_strlen(char *str);
int     ft_strcmp(char *s1, char *s2);
```

- We'll launch the following command-line :

```
sh libft_creator.sh
```

Chapter IV

Exercise 01 : Makefile

	Exercise 01
Makefile	
Turn-in directory : $ex01/$	
Files to turn in : `Makefile`	
Allowed functions : `None`	

- Create the `Makefile` that'll compile a library `libft.a`.
- Your makefile should print all the command it's running.
- Your makefile should not run any unnecessary command.
- The `Makefile` will get its source files from the "srcs" directory.
- Those files will be: ft_putchar.c, ft_swap.c, ft_putstr.c, ft_strlen.c, ft_strcmp.c
- The `Makefile` will get its header files from the "includes" directory.
- Those files will be: ft.h
- It should compile the .c files with gcc and with -Wall -Wextra -Werror flags in that order.
- The lib should be at the root of the exercise.
- .o files should be near their .c file.
- The `Makefile` should also implement the following rules: `clean`, `fclean`, `re`, `all` and of course `libft.a`.
- Running just `make` should be equal to `make all`
- The rule `all` should be equal to `make libft.a`.
- The rule `clean` should remove all the temporary generated files.

- The rule `fclean` should be like a `make clean` plus all the binary made with `make all`.
- The rule `re` should be like a `make fclean` followed by `make all`.
- Your makefile should not compile any file for nothing.
- We'll only fetch your Makefile and test it with our files.

`Watch out for wildcards!`

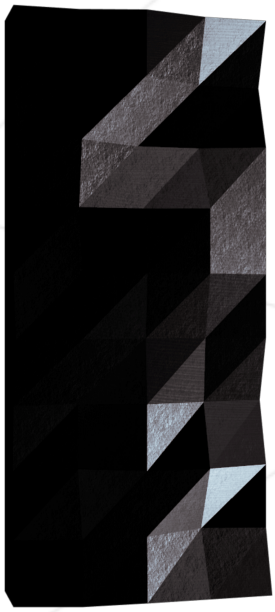
Chapter V

Exercise 02 : ft_split

	Exercise 02
ft_split	
Turn-in directory : *ex*02/	
Files to turn in : `ft_split.c`	
Allowed functions : `malloc`	

- Create a function that slits a string of character depending on another string of characters.
- You'll have to use each character from the string `charset` as a separator.
- The function returns an array where each box contains the address of a string wrapped between two separators. The last element of that array should equal to 0 to indicate the end of the array.
- There cannot be any empty strings in your array. Draw your conclusions accordingly.
- The string given as argument won't be modifiable.
- Here's how it should be prototyped :

```
char **ft_split(char *str, char *charset);
```



C Piscine

C 10

Summary: This document is the subject for the module C 10 of the C Piscine @ 42.

Contents

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change before 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.
- 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. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of 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` take into account 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 `gcc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional file in your directory 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 "C Piscine" part of the forum on the intranet, or 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 !!!

Chapter II

Foreword

Body Count is an American heavy metal band formed in Los Angeles, California, in 1990. The group is fronted by Ice-T, who co-founded the group with lead guitarist Ernie C out of their interest in heavy metal music. Ice-T took on the role of vocalist and writing the lyrics for most of the group's songs. Lead guitarist Ernie C has been responsible for writing the group's music. Their controversial self-titled debut album was released on Sire Records in 1992.

The song "Cop Killer" was the subject of much controversy. Although Sire Records' parent company, Warner Bros. Records, defended the single, Ice-T chose to remove the track from the album because he felt that the controversy had eclipsed the music itself. The group left Sire the following year. Since then, they have released three further albums on different labels, none of which have been received as commercially or critically well as their debut album.

Three out of the band's original six members are deceased: D-Roc died from lymphoma, Beatmaster V from leukemia and Mooseman in a drive-by shooting.

Click here, start it, and work... Right Now !

Chapter III

Exercise 00 : display_file

	Exercise 00
display_file	
Turn-in directory : $ex00/$	
Files to turn in : `Makefile, and files needed for your program`	
Allowed functions : `close, open, read, write`	

- Create a program called `ft_display_file` that displays, on the standard output, only the content of the file given as argument.
- The submission directory should have a `Makefile` with the following rules : `all`, `clean`, `fclean`. The binary will be called `ft_display_file`.
- The `malloc` function is forbidden. You can only do this exercise by declaring a fixed-sized array.
- All files given as arguments will be valid.
- Error messages have to be displayed on their reserved output followed by a new line.
- If no argument is given, it should display

```
File name missing.
```

- If there is more than one argument, it should display

```
Too many arguments.
```

- If the file cannot be read, it should display

```
Cannot read file.
```

Chapter IV

Exercise 01 : cat

	Exercise 01
cat	
Turn-in directory : $ex01/$	
Files to turn in : `Makefile, and files needed for your program`	
Allowed functions : `close, open, read, write, strerror, basename`	

- Create a program called `ft_cat` which does the same thing as the system's `cat` command-line.
- You don't have to handle options.
- The submission directory should have a `Makefile` with the following rules : `all`, `clean`, `fclean`.
- You may use the variable `errno` (check the `man` for `Errno`).
- You should read the man of all the authorized functions
- You can only do this exercise by declaring a fixed-sized array. This array will have a size limited to a little less than `30 ko`. In order to test that size-limit, use the `ulimit` command-line in your Shell.

Chapter V

Exercise 02 : tail

	Exercise 02
tail	
Turn-in directory : $ex02/$	
Files to turn in : `Makefile, and files needed for your program`	
Allowed functions : `close, open, read, write, malloc, free, strerror, basename`	

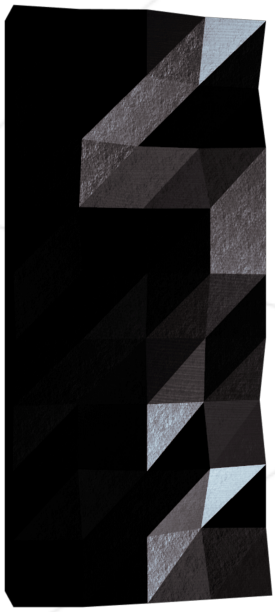
- Create a program called `ft_tail` which does the same thing as the system command `tail`.
- The only option you have to handle is `-c`, but you don't need to handle '+' or '-' signs.
- all the test will be done with the `-c` option.
- The submission directory should have a `Makefile` with the following rules : `all`, `clean`, `fclean`.
- You may use the variable `errno`.

Chapter VI

Exercise 03 : hexdump

	Exercise 03
hexdump	
Turn-in directory : $ex03/$	
Files to turn in : `Makefile, and files needed for your program`	
Allowed functions : `close, open, read, write, malloc, free, strerror, basename`	

- Create a program called `ft_hexdump` which does the same thing as the system's `hexdump` command-line without redirection.
- The only option you have to handle is `-C`.
- The submission directory should have a `Makefile` with the following rules : `all`, `clean`, `fclean`.
- You may use the variable `errno`.



C Piscine

C 11

Summary: This document is the subject for the module C 11 of the C Piscine @ 42.

Contents

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change before 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.
- 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. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of 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` take into account 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 `gcc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional file in your directory 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 "C Piscine" part of the forum on the intranet, or 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 !!!

Chapter II

Foreword

Here's a little story :

(1982, California) Larry Walters of Los Angeles is one of the few to contend for the Darwin Awards and live to tell the tale. "I have fulfilled my 20-year dream," said Walters, a former truck driver for a company that makes TV commercials. "I'm staying on the ground. I've proved the thing works."
Larry's boyhood dream was to fly. But fates conspired to keep him from his dream. He joined the Air Force, but his poor eyesight disqualified him from the job of pilot. After he was discharged from the military, he sat in his backyard watching jets fly overhead.

He hatched his weather balloon scheme while sitting outside in his "extremely comfortable" Sears lawnchair. He purchased 45 weather balloons from an Army-Navy surplus store, tied them to his tethered lawnchair (dubbed the Inspiration I) and filled the four-foot diameter balloons with helium. Then, armed with some sandwiches, Miller Lite, and a pellet gun, he strapped himself into his lawnchair. He figured he would shoot to pop a few of the many balloons when it was time to descend.

Larry planned to sever the anchor and lazily float to a height of about 30 feet above the backyard, where he would enjoy a few hours of flight before coming back down. But things didn't work out quite as Larry planned.

When his friends cut the cord anchoring the lawnchair to his Jeep, he did not float lazily up to 30 feet. Instead he streaked into the LA sky as if shot from a cannon, pulled by the lift of 45 helium balloons, holding 33 cubic feet of helium each.

He didn't level off at 100 feet, nor did he level off at 1000 feet. After climbing and climbing, he leveled off at 16,000 feet.

At that height he felt he couldn't risk shooting any of the balloons, lest he unbalance the load and really find himself in trouble. So he stayed there, drifting cold and frightened with his beer and sandwiches, for more than 14 hours. He crossed the primary approach corridor of LAX, where startled Trans World Airlines and Delta Airlines pilots radioed in reports

of the strange sight.

Eventually he gathered the nerve to shoot a few balloons, and slowly descended. The hanging tethers tangled and caught in a power line, blacking out a Long Beach neighborhood for 20 minutes. Larry climbed to safety, where he was arrested by waiting members of the LAPD. As he was led away in handcuffs, a reporter dispatched to cover the daring rescue asked him why he had done it. Larry replied nonchalantly, "A man can’t just sit around."

The Federal Aviation Administration was not amused. Safety Inspector Neal Savoy said, "We know he broke some part of the Federal Aviation Act, and as soon as we decide which part it is, a charge will be filed."

The moral of this story is Larry Walters should have stay on his chair and learn C....

Chapter III

Exercise 00 : ft_foreach

	Exercise 00
ft_foreach	
Turn-in directory : $ex00/$	
Files to turn in : `ft_foreach.c`	
Allowed functions : `None`	

- Create the function `ft_foreach` which, for a given ints array, applies a function on all elements of the array. This function will be applied following the array's order.
- Here's how the function should be prototyped :

```
void        ft_foreach(int *tab, int length, void(*f)(int));
```

- For example, the function `ft_foreach` could be called as follows in order to display all ints of the array :

```
ft_foreach(tab, 1337, &ft_putnbr);
```

Chapter IV

Exercise 01 : ft_map

	Exercise 01
ft_map	
Turn-in directory : $ex01/$	
Files to turn in : `ft_map.c`	
Allowed functions : `malloc`	

- Create the function `ft_map` which, for a given ints array, applies a function on all elements of the array (in order) and returns a array of all the return values.
- This function will be applied following the array's order.
- Here's how the function should be prototyped :

```
int        *ft_map(int *tab, int length, int(*f)(int));
```

Chapter V

Exercise 02 : ft_any

	Exercise 02
ft_any	
Turn-in directory : *ex*02/	
Files to turn in : `ft_any.c`	
Allowed functions : `None`	

- Create a function `ft_any` which will return 1 if, passed to the function `f`, at least one element of the array returns something else than 0. Else, it should return 0.
- This function will be applied following the array's order.
- Here's how the function should be prototyped :

```
int                ft_any(char **tab, int(*f)(char*));
```

- The array will be delimited with a null pointer.

Chapter VI

Exercise 03 : ft_count_if

	Exercise 03
ft_count_if	
Turn-in directory : *ex*03/	
Files to turn in : `ft_count_if.c`	
Allowed functions : `None`	

- Create a function `ft_count_if` which will return the number of elements of the array that return does not return 0 when passed to the function `f`.
- This function will be applied following the array's order.
- Here's how the function should be prototyped :

```
int                ft_count_if(char **tab, int length, int(*f)(char*));
```

Chapter VII

Exercise 04 : ft_is_sort

	Exercise 04
ft_is_sort	
Turn-in directory : *ex*04/	
Files to turn in : `ft_is_sort.c`	
Allowed functions : `None`	

- Create a function `ft_is_sort` which returns 1 if the array is sorted and 0 if it isn't.
- The function given as argument should return a negative integer if the first argument is lower than the second, 0 if they're equal or a positive integer for anything else.
- Here's how the function should be prototyped :

```
int		ft_is_sort(int *tab, int length, int(*f)(int, int));
```

Chapter VIII

Exercise 05 : do-op

	Exercise 05
do-op	
Turn-in directory : $ex05/$	
Files to turn in : `Your program files`	
Allowed functions : `write`	

- Create a program called `do-op`.
- The progam will be executed with three arguments: `do-op value1 operateur value2`
- Example :

```
$>./do-op 42 "+" 21
63
$>
```

- You should use an array of pointers to function to take care of the `operator`.
- In case of an invalid operator your program should print `0`.
- If the number of arguments is invalid, `do-op` doesn't display anything.
- Your program should accept and print the result for the following operators: '+' '-' '/' '*' and '%'
- In case of a division by 0, it should print:

```
Stop : division by zero
```

- In case of a modulo by 0, it should print:

```
Stop : modulo by zero
```

- Here's an example of tests the Moulinette will run :

```
$> make clean
$> make
$> ./do-op
$> ./do-op 1 + 1
2
$> ./do-op 42amis - --+-20toto12
62
$> ./do-op 1 p 1
0
$> ./do-op 1 + toto3
1
$>
$> ./do-op toto3 + 4
4
$> ./do-op foo plus bar
0
$> ./do-op 25 / 0
Stop : division by zero
$> ./do-op 25 % 0
Stop : modulo by zero
$>
```

Chapter IX

Exercise 06 : ft_sort_string_tab

	Exercise 06
ft_sort_string_tab	
Turn-in directory : $ex06/$	
Files to turn in : `ft_sort_string_tab.c`	
Allowed functions : `None`	

- Create the function `ft_sort_string_tab`, by `ascii` order the strings in `tab`.
- `tab` will be null terminated
- The sorting will be performed by exchanging the array's pointers.
- Here's how it should be prototyped :

```
void ft_sort_string_tab(char **tab);
```

Chapter X

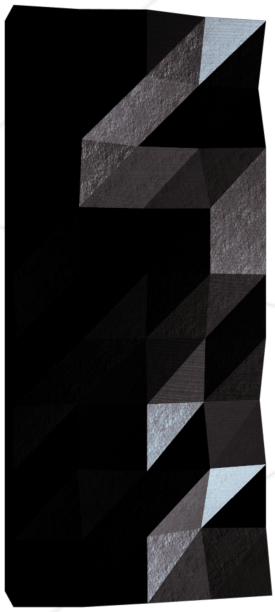
Exercise 07 : ft_advanced_sort_string_tab

	Exercise 07
ft_advanced_sort_string_tab	
Turn-in directory : *ex*07/	
Files to turn in : `ft_advanced_sort_string_tab.c`	
Allowed functions : `None`	

- Create the function `ft_advanced_sort_string_tab` which sorts, depending on the return of the function given as argument
- The sorting will be performed by exchanging the array's pointers.
- `tab` will be null terminated
- Here's how it should be prototyped :

```
void ft_advanced_sort_string_tab(char **tab, int(*cmp)(char *, char *));
```

Calling ft_advanced_sort_string_tab() with ft_strcmp as a second argument will return the same result as ft_sort_string_tab().



C Piscine

C 12

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

Contents

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 `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 II

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change before 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.
- 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. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of 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` take into account 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 `gcc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional file in your directory 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 "C Piscine" part of the forum on the intranet, or 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 !!!
- 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

Exercice 00 : ft_create_elem

	Exercise 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 `data` to the given argument and `next` to NULL.
- Here's how it should be prototyped :

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

Chapter IV

Exercice 01 : ft_list_push_front

	Exercise 01
ft_list_push_front	
Turn-in directory : $ex01/$	
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 `data` to the given argument.
- 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 V

Exercice 02 : ft_list_size

	Exercise 02
ft_list_size	
Turn-in directory : $ex02/$	
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 VI

Exercice 03 : ft_list_last

	Exercise 03
ft_list_last	
Turn-in directory : $ex03/$	
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 VII

Exercice 04 : ft_list_push_back

	Exercise 04
ft_list_push_back	
Turn-in directory : $ex04/$	
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 `data` to the given argument.
- 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 VIII

Exercice 05 : ft_list_push_strs

	Exercise 05
ft_list_push_strs	
Turn-in directory : $ex05/$	
Files to turn in : `ft_list_push_strs.c, ft_list.h`	
Allowed functions : `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.
- Here's how it should be prototyped :

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

Chapter IX

Exercice 06 : ft_list_clear

	Exercise 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 remove and free all links from the list.
- `free_fct` to free each `data`
- Here's how it should be prototyped :

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

Chapter X

Exercice 07 : ft_list_at

	Exercise 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, knowing that the first element of the list is when `nbr` equal 0.
- 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 XI

Exercice 08 : ft_list_reverse

	Exercise 08
ft_list_reverse	
Turn-in directory : *ex*08/	
Files to turn in : `ft_list_reverse.c`	
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 in that function we will use our own ft_list.h
- Here's how it should be prototyped :

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

Chapter XII

Exercice 09 : ft_list_foreach

	Exercise 09
ft_list_foreach	
Turn-in directory : *ex*09/	
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 argument to each of the list's elements.
- `f` should be apply with the same order as the list
- 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 XIII

Exercice 10 : ft_list_foreach_if

	Exercise 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 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 apply with the same order as the list
- Here's how it should be prototyped :

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

- 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 XIV

Exercice 11 : ft_list_find

	Exercise 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 element's data compared to `data_ref` with `cmp` makes `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)());
```

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

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

Chapter XV

Exercice 12 : ft_list_remove_if

	Exercise 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 erases off the list all elements whose data compared to `data_ref` with `cmp` makes `cmp` to return 0.
- The data from an element that should be erase, 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 *))
```

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

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

Chapter XVI

Exercice 13 : ft_list_merge

	Exercise 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 `begin2` at the end of an other 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 XVII

Exercice 14 : ft_list_sort

	Exercise 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 elements by ascending order by comparing two elements by comparing their data with a function.
- Here's how it should be prototyped :

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

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

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

`cmp could be for instance ft_strcmp.`

Chapter XVIII

Exercice 15 : ft_list_reverse_fun

	Exercise 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.
- Here's how it should be prototyped :

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

Chapter XIX

Exercice 16 : ft_sorted_list_insert

	Exercise 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 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 XX

Exercice 17 : ft_sorted_list_merge

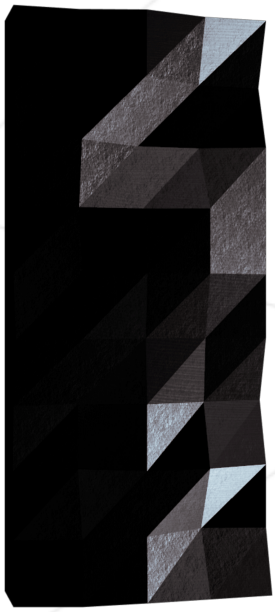
	Exercise 17
ft_sorted_list_merge	
Turn-in directory : *ex*17/	
Files to turn in : `ft_sorted_list_merge.c, ft_list.h`	
Allowed functions : `None`	

- Create the function `ft_sorted_list_merge` which integrates the elements of a sorted list `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);
```



C Piscine

C 13

Summary: This document is the subject for the module C 13 of the C Piscine @ 42.

Contents

Chapter I

Instructions

- Only this page will serve as reference: do not trust rumors.
- Watch out! This document could potentially change before 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.
- 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. So if you want to avoid bad surprises, be as thorough as possible.
- Moulinette is not very open-minded. It won't try and understand your code if it doesn't respect the Norm. Moulinette relies on a program called `norminette` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of 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` take into account 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 `gcc`.
- If your program doesn't compile, you'll get `0`.
- You cannot leave any additional file in your directory 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 "C Piscine" part of the forum on the intranet, or 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 !!!
- For the following exercises, we'll use the following structure :

```
typedef struct            s_btree
{
        struct s_btree    *left;
        struct s_btree    *right;
        void              *item;
}                         t_btree;
```

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

Chapter II

Foreword

Here's the list of releases for `Venom` :

```
- In League with Satan (single, 1980)
- Welcome to Hell (1981)
- Black Metal (1982)
- Bloodlust (single, 1983)
- Die Hard (single, 1983)
- Warhead (single, 1984)
- At War with Satan (1984)
- Hell at Hammersmith (EP, 1985)
- American Assault (EP, 1985)
- Canadian Assault (EP, 1985)
- French Assault (EP, 1985)
- Japanese Assault (EP, 1985)
- Scandinavian Assault (EP, 1985)
- Manitou (single, 1985)
- Nightmare (single, 1985)
- Possessed (1985)
- German Assault (EP, 1987)
- Calm Before the Storm (1987)
- Prime Evil (1989)
- Tear Your Soul Apart (EP, 1990)
- Temples of Ice (1991)
- The Waste Lands (1992)
- Venom '96 (EP, 1996)
- Cast in Stone (1997)
- Resurrection (2000)
- Anti Christ (single, 2006)
- Metal Black (2006)
- Hell (2008)
- Fallen Angels (2011)
```

Today's subject will seem easier if you listen to `Venom`.

Chapter III

Exercise 00 : btree_create_node

	Exercise 00
btree_create_node	
Turn-in directory : *ex*00/	
Files to turn in : `btree_create_node.c, ft_btree.h`	
Allowed functions : `malloc`	

- Create the function `btree_create_node` which allocates a new element. It should initialise its `item` to the argument's value, and all other elements to 0.
- The created node's address is returned.
- Here's how it should be prototyped :

```
t_btree *btree_create_node(void *item);
```

Chapter IV

Exercise 01 : btree_apply_prefix

	Exercise 01
btree_apply_prefix	
Turn-in directory : $ex01/$	
Files to turn in : `btree_apply_prefix.c, ft_btree.h`	
Allowed functions : `None`	

- Create a function `btree_apply_prefix` which applies the function given as argument to the `item` of each node, using `prefix traversal` to search the tree.
- Here's how it should be prototyped :

```
void btree_apply_prefix(t_btree *root, void (*applyf)(void *));
```

Chapter V

Exercise 02 : btree_apply_infix

	Exercise 02
btree_apply_infix	
Turn-in directory : $ex02/$	
Files to turn in : `btree_apply_infix.c, ft_btree.h`	
Allowed functions : `None`	

- Create a function `btree_apply_infix` which applies the function given as argument to the `item` of each node, using `infix traversal` to search the tree.
- Here's how it should be prototyped :

```
void btree_apply_infix(t_btree *root, void (*applyf)(void *));
```

Chapter VI

Exercise 03 : btree_apply_suffix

	Exercise 03
btree_apply_suffix	
Turn-in directory : $ex03/$	
Files to turn in : `btree_apply_suffix.c, ft_btree.h`	
Allowed functions : `None`	

- Create a function `btree_apply_suffix` which applies the function given as argument to the `item` of each node, using `suffix traversal` to search the tree.
- Here's how it should be prototyped :

```
void btree_apply_suffix(t_btree *root, void (*applyf)(void *));
```

Chapter VII

Exercise 04 : btree_insert_data

	Exercise 04
btree_insert_data	
Turn-in directory : *ex*04/	
Files to turn in : `btree_insert_data.c, ft_btree.h`	
Allowed functions : `btree_create_node`	

- Create a function `btree_insert_data` which inserts the element `item` into a tree. The tree passed as argument will be sorted : for each `node` all lower elements are located on the left side and all higher or equal elements on the right. We'll also pass a comparison function similar to `strcmp` as argument.
- The `root` parameter points to the root node of the tree. First time called, it should point to `NULL`.
- Here's how it should be prototyped :

```
void btree_insert_data(t_btree **root, void *item, int (*cmpf)(void *, void *));
```

Chapter VIII

Exercise 05 : btree_search_item

	Exercise 05
btree_search_item	
Turn-in directory : *ex*05/	
Files to turn in : `btree_search_item.c, ft_btree.h`	
Allowed functions : `None`	

- Create a function `btree_search_item` which returns the first element related to the reference data given as argument. The tree should be browsed using `infix traversal` . If the element isn't found, the function should return `NULL`.
- Here's how it should be prototyped :

```
void *btree_search_item(t_btree *root, void *data_ref, int (*cmpf)(void *, void *));
```

Chapter IX

Exercise 06 : btree_level_count

	Exercise 06
btree_level_count	
Turn-in directory : $ex06/$	
Files to turn in : `btree_level_count.c, ft_btree.h`	
Allowed functions : `None`	

- Create a function `btree_level_count` which returns the size of the largest branch passed as argument.
- Here's how it should be prototyped :

```
int btree_level_count(t_btree *root);
```

Chapter X

Exercise 07 : btree_apply_by_level

	Exercise 07
btree_apply_by_level	
Turn-in directory : $ex07/$	
Files to turn in : `btree_apply_by_level.c, ft_btree.h`	
Allowed functions : `malloc, free`	

- Create a function `btree_apply_by_level` which applies the function passed as argument to each node of the tree. The tree must be browsed level by level. The function called will take three arguments :
  - The first argument, of type `void *`, will correspond to the node's item ;
  - The second argument, of type `int`, corresponds to the level on which we find : 0 for root, 1 for children, 2 for grand-children, etc. ;
  - The third argument, of type `int`, is worth 1 if it's the first `node` of the level, or worth 0 otherwise.
- Here's how it should be prototyped :

```
void btree_apply_by_level(t_btree *root, void (*applyf)(void *item, int current_level, int is_first_elem))
```