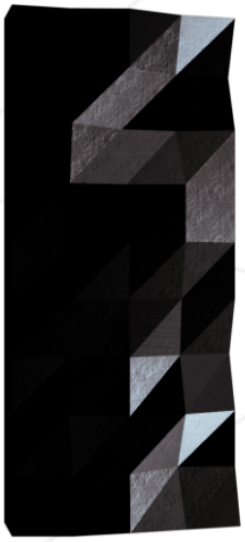

Piscine

09

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

Version: 2

Contents

Ch pter 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? sk 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 l unched with the *-R CheckForbiddenSourceHe der* fl g. Moulinette will use it too.

Ch pter 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.  m I wrong?  m 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:   world of pain.
Smokey: Dude, he's your partner...
Walter Sobchak: [shouting] Has the whole world gone crazy?  m 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:  ll right, it's fucking zero.  re you happy, you crazy fuck?
Walter Sobchak: ...It's a league game, Smokey.
```

Ch pter III

Exercise 00 : libft

	Exercise 00
libft	
Turn-in directory : $ex00$	
Files to turn in : `libft_cre tor.sh, ft_putch r.c, ft_sw p.c, ft_putstr.c, ft_strlen.c, ft_strcmp.c`	
llowed functions : `write`	

Create your `ft` library. It'll be called `libft.a`.

shell script called `libft_creator.sh` will compile source files appropriately and will create your library.

This library should contain <u>all</u> of the following functions :

```
void    ft_putch r(ch r c);
void    ft_sw p(int * , int *b);
void    ft_putstr(ch r *str);
int     ft_strlen(ch r *str);
int     ft_strcmp(ch r *s1, ch r *s2);
```

We'll launch the following command-line :

```
sh libft_cre tor.sh
```

Ch pter IV

Exercise 01 : M kefile

	Exercise 01
Makefile	
Turn-in directory : $ex01$	
Files to turn in : `M kefile`	
llowed 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.

`W tch out for wildc rds!`

Ch pter V

Exercise 02 : ft_split

	Exercise 02
ft_split	
Turn-in directory : *ex*02	
Files to turn in : `ft_split.c`	
llowed functions : `m lloc`	

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 :

```
ch r **ft_split(ch r *str, ch r *ch rset);
```