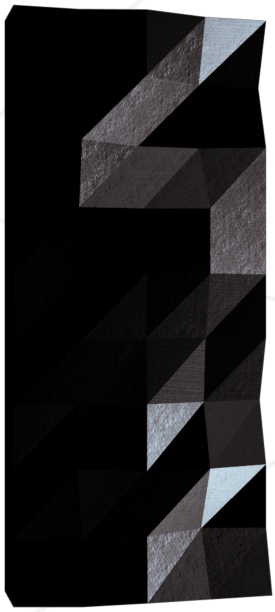

C Piscine

Day 05

Staff 42 pedago@42.fr

Abstract: This document is the subject for Day05 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 up to an hour 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 `Norminator` to check if your files respect the norm. TL;DR: it would be idiotic to submit a piece of work that doesn't pass `Norminator`'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.
- 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, 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 your right. Otherwise, try your peer on your left.
- Your reference guide is called `Google / man / the Internet / ....`
- Check out the "C Piscine" part of the forum on the intranet.
- 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 !!!

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

Chapter II

Foreword

Here is the rules of the internet, according to Encyclopedia Dramatica :

1. Do not talk about /b/
2. Do NOT talk about /b/
3. We are Anonymous
4. Anonymous is legion
5. Anonymous never forgives
6. Anonymous can be a horrible, senseless, uncaring monster
7. Anonymous is still able to deliver
8. There are no real rules about posting
9. There are no real rules about moderation either - enjoy your ban
10. If you enjoy any rival sites - DON'T
11. All your carefully picked arguments can easily be ignored
12. Anything you say can and will be used against you
13. Anything you say can be turned into something else - fixed
14. Do not argue with trolls - it means that they win
15. The harder you try the harder you will fail
16. If you fail in epic proportions, it may just become a winning failure
17. Every win fails eventually
18. Everything that can be labeled can be hated
19. The more you hate it the stronger it gets
20. Nothing is to be taken seriously
21. Original content is original only for a few seconds before getting old
22. Copypasta is made to ruin every last bit of originality
23. Copypasta is made to ruin every last bit of originality
24. Every repost is always a repost of a repost
25. Relation to the original topic decreases with every single post
26. Any topic can be easily turned into something totally unrelated
27. Always question a person's sexual preferences without any real reason
28. Always question a person's gender - just in case it's really a man
29. In the internet all girls are men and all kids are undercover FBI agents
30. There are no girls on the internet
31. TITS or GTFO - the choice is yours
32. You must have pictures to prove your statements
33. Lurk more - it's never enough
34. There is porn of it, no exceptions

35. If no porn is found at the moment, it will be made
36. There will always be even more fucked up shit than what you just saw
37. You cannot divide by zero (just because the calculator says so)
38. No real limits of any kind apply here - not even the sky
39. CAPSLOCK IS CRUISE CONTROL FOR COOL
40. EVEN WITH CRUISE CONTROL YOU STILL HAVE TO STEER
41. Desu isn't funny. Seriously guys. It's worse than Chuck Norris jokes.
42. Nothing is Sacred
43. The more beautiful and pure a thing is - the more satisfying it is to corrupt it
44. Even one positive comment about Japanese things can make you a weeaboo
45. When one sees a lion, one must get into the car.
46. There is always furry porn of it.
47. The pool is always closed.

Hopefully, thoses rules are mandatory to complete the following exercices.

Chapter III

Exercise 00 : ft_putstr

	Exercice : 00
ft_putstr	
Turn-in directory : *ex*00/	
Files to turn in : `ft_putstr.c`	
Allowed functions : `ft_putchar`	
Remarks : `n/a`	

42 - Classics : Theses exercises are key assignments that do not earn points, but are mandatory to validate in order to access to the real assignments of the day.

- Create a function that displays a string of characters on the standard output.
- Here's how it should be prototyped :

```
void    ft_putstr(char *str);
```

Chapter IV

Exercise 01 : ft_putnbr

	Exercice : 01
ft_putnbr	
Turn-in directory : *ex*01/	
Files to turn in : `ft_putnbr.c`	
Allowed functions : `ft_putchar`	
Remarks : `n/a`	

42 - Classics : Theses exercises are key assignments that do not earn points, but are mandatory to validate in order to access to the real assignments of the day.

- Create a function that displays the number entered as a parameter. The function has to be able to display all possible values within an `int` type variable.
- Here's how it should be prototyped :

```
void ft_putnbr(int nb);
```

- For example:
  - `ft_putnbr(42)` displays "42".

Chapter V

Exercise 02 : ft_atoi

	Exercice : 02
ft_atoi	
Turn-in directory : *ex*02/	
Files to turn in : `ft_atoi.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

42 - Classics : Theses exercises are key assignments that do not earn points, but are mandatory to validate in order to access to the real assignments of the day.

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

```
int     ft_atoi(char *str);
```

Chapter VI

Exercise 03 : ft_strcpy

	Exercice : 03
ft_strcpy	
Turn-in directory : $ex03/$	
Files to turn in : `ft_strcpy.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

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

```
char        *ft_strcpy(char *dest, char *src);
```

Chapter VII

Exercise 04 : ft_strncpy

	Exercice : 04
ft_strncpy	
Turn-in directory : $ex04/$	
Files to turn in : `ft_strncpy.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

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

```
char        *ft_strncpy(char *dest, char *src, unsigned int n);
```

Chapter VIII

Exercise 05 : ft_strstr

	Exercice : 05
ft_strstr	
Turn-in directory : $ex05/$	
Files to turn in : `ft_strstr.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

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

```
char        *ft_strstr(char *str, char *to_find);
```

Chapter IX

Exercise 06 : ft_strcmp

	Exercice : 06
ft_strcmp	
Turn-in directory : $ex06/$	
Files to turn in : `ft_strcmp.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

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

```
int        ft_strcmp(char *s1, char *s2);
```

Chapter X

Exercise 07 : ft_strncmp

	Exercice : 07
ft_strncmp	
Turn-in directory : $ex07/$	
Files to turn in : `ft_strncmp.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

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

```
int        ft_strncmp(char *s1, char *s2, unsigned int n);
```

Chapter XI

Exercise 08 : ft_strupcase

	Exercice : 08
ft_strupcase	
Turn-in directory : $ex08/$	
Files to turn in : `ft_strupcase.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

- Create a function that transforms every letter of every word to uppercase.
- Here's how it should be prototyped :

```
char	*ft_strupcase(char *str);
```

- It should return `str`.

Chapter XII

Exercise 09 : ft_strlowcase

	Exercice : 09
ft_strlowcase	
Turn-in directory : $ex09/$	
Files to turn in : `ft_strlowcase.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

- Create a function that transforms every letter of every word to lowercase.
- Here's how it should be prototyped :

```
char        *ft_strlowcase(char *str);
```

- It should return `str`.

Chapter XIII

Exercise 10 : ft_strcapitalize

	Exercice : 10
ft_strcapitalize	
Turn-in directory : $ex10/$	
Files to turn in : `ft_strcapitalize.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

- Create a function that capitalizes the first letter of each word and transforms all other letters to lowercase.
- A word is a string of alphanumeric characters.
- Here's how it should be prototyped :

```
char        *ft_strcapitalize(char *str);
```

- It should return `str`.
- For example:

```
salut, comment tu vas ? 42mots quarante-deux; cinquante+et+un
```

- Becomes:

```
Salut, Comment Tu Vas ? 42mots Quarante-Deux; Cinquante+Et+Un
```

Chapter XIV

Exercise 11 : ft_str_is_alpha

	Exercice : 11
ft_str_is_alpha	
Turn-in directory : $ex11/$	
Files to turn in : `ft_str_is_alpha.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

- Create a function that returns 1 if the string given as a parameter contains only alphabetical characters, and 0 if it contains any other character.
- Here's how it should be prototyped :

```
int        ft_str_is_alpha(char *str);
```

- It should return 1 if `str` is empty.

Chapter XV

Exercise 12 : ft_str_is_numeric

	Exercice : 12
ft_str_is_numeric	
Turn-in directory : $ex12/$	
Files to turn in : `ft_str_is_numeric.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

- Create a function that returns 1 if the string given as a parameter contains only digits, and 0 if it contains any other character.
- Here's how it should be prototyped :

```
int        ft_str_is_numeric(char *str);
```

- It should return 1 if `str` is empty.

Chapter XVI

Exercise 13 : ft_str_is_lowercase

	Exercice : 13
ft_str_is_lowercase	
Turn-in directory : $ex13/$	
Files to turn in : `ft_str_is_lowercase.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

- Create a function that returns 1 if the string given as a parameter contains only lowercase alphabetical characters, and 0 if it contains any other character.
- Here's how it should be prototyped :

```
int        ft_str_is_lowercase(char *str);
```

- It should return 1 if `str` is empty.

Chapter XVII

Exercise 14 : ft_str_is_uppercase

	Exercice : 14
ft_str_is_uppercase	
Turn-in directory : $ex14/$	
Files to turn in : `ft_str_is_uppercase.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

- Create a function that returns 1 if the string given as a parameter contains only uppercase alphabetical characters, and 0 if it contains any other character.
- Here's how it should be prototyped :

```
int        ft_str_is_uppercase(char *str);
```

- It should return 1 if `str` is empty.

Chapter XVIII

Exercise 15 : ft_str_is_printable

	Exercice : 15
ft_str_is_printable	
Turn-in directory : $ex15/$	
Files to turn in : `ft_str_is_printable.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

- Create a function that returns 1 if the string given as a parameter contains only printable characters, and 0 if it contains any other character.
- Here's how it should be prototyped :

```
int        ft_str_is_printable(char *str);
```

- It should return 1 if `str` is empty.

Chapter XIX

Exercise 16 : ft_strcat

	Exercice : 16
ft_strcat	
Turn-in directory : $ex16/$	
Files to turn in : `ft_strcat.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

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

```
char *ft_strcat(char *dest, char *src);
```

Chapter XX

Exercise 17 : ft_strncat

	Exercice : 17
ft_strncat	
Turn-in directory : $ex17/$	
Files to turn in : `ft_strncat.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

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

```
char *ft_strncat(char *dest, char *src, int nb);
```

Chapter XXI

Exercise 18 : ft_strlcat

	Exercice : 18
ft_strlcat	
Turn-in directory : $ex18/$	
Files to turn in : `ft_strlcat.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

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

```
unsigned int ft_strlcat(char *dest, char *src, unsigned int size);
```

Chapter XXII

Exercise 19 : ft_strlcpy

	Exercice : 19
ft_strlcpy	
Turn-in directory : $ex19/$	
Files to turn in : `ft_strlcpy.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

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

```
unsigned int ft_strlcpy(char *dest, char *src, unsigned int size);
```

Chapter XXIII

Exercise 20 : ft_putnbr_base

	Exercice : 20
ft_putnbr_base	
Turn-in directory : *ex*20/	
Files to turn in : `ft_putnbr_base.c`	
Allowed functions : `ft_putchar`	
Remarks : `n/a`	

- Create a function that displays a number in a base system onscreen.
- This number is given in the shape of an `int`, and the radix in the shape of a `string of characters`.
- The base-system contains all useable symbols to display that number :
  - `0123456789` is the commonly used base system to represent decimal numbers ;
  - `01` is a binary base system ;
  - `0123456789ABCDEF` an hexadecimal base system ;
  - `poneyvif` is an octal base system.
- The function must handle negative numbers.
- If there's an invalid argument, nothing should be displayed. Examples of invalid arguments :
  - base is empty or size of 1;
  - base contains the same character twice ;

- base contains + or - ;
- etc.

- Here's how it should be prototyped :

```
void        ft_putnbr_base(int nbr, char *base);
```

Chapter XXIV

Exercise 21 : ft_atoi_base

	Exercice : 21
ft_atoi_base	
Turn-in directory : *ex*21/	
Files to turn in : `ft_atoi_base.c`	
Allowed functions : `Nothing`	
Remarks : `n/a`	

- Create a function that returns a number. This number is shaped as a `string of characters`.
- The string of characters reveals the number in a specific base, given as a second parameter.
- The function must handle negative numbers.
- The function must handle signs like `man atoi`.
- If there's an invalid argument, the function should return 0. Examples of invalid arguments :
  - str is an empty string ;
  - the base is empty or size of 1;
  - str contains characters that aren't part of the base, or aren't + nor - ;
  - the base contains the same character twice ;
  - the base contains + or - ;
  - etc.
- Here's how it should be prototyped :

```
int        ft_atoi_base(char *str, char *base);
```

Chapter XXV

Exercise 22 : ft_putstr_non_printable

	Exercice : 22
ft_putstr_with_non_printable	
Turn-in directory : *ex*22/	
Files to turn in : `ft_putstr_non_printable.c`	
Allowed functions : `ft_putchar`	
Remarks : `n/a`	

- Create a function that displays a string of characters onscreen. If this string contains characters that aren't printable, they'll have to be displayed in the shape of hexadecimals (lowercase), preceeded by a "backslash".
- For example :

```
Coucou\ntu vas bien ?
```

- The function should display :

```
Coucou\0atu vas bien ?
```

- Here's how it should be prototyped :

```
void	ft_putstr_non_printable(char *str);
```

Chapter XXVI

Exercise 23 : ft_print_memory

	Exercice : 23
ft_print_memory	
Turn-in directory : $ex23/$	
Files to turn in : `ft_print_memory.c`	
Allowed functions : `ft_putchar`	
Remarks : `n/a`	

- Create a function that displays the memory area onscreen.
- The display of this memory area should be split into three columns :
  - The hexadecimal address of the first line's first character ;
  - The content in hexadecimal ;
  - The content in printable characters.
- If a character is non-printable, it'll be replaced by a dot.
- Each line should handle sixteen characters.
- If `size` equals to 0, nothing should be displayed.

- Example:

```
guilla_i@seattle $> ./ft_print_memory
00000000: 5361 6c75 7420 6c65 7320 616d 696e 6368  Salut les aminch
00000010: 6573 2063 2765 7374 2063 6f6f 6c20 7368  es c'est cool sh
00000020: 6f77 206d 656d 206f 6e20 6661 6974 2064  ow mem on fait d
00000030: 6520 7472 7563 2074 6572 7269 626c 6500  e truc terrible.
00000040: 2e00 0102 0304 0506 0708 090e 0f1b 7f    ..............
guilla_i@seattle $> ./ft_print_memory | cat -te
00000000: 5361 6c75 7420 6c65 7320 616d 696e 6368  Salut les aminch$
00000010: 6573 2063 2765 7374 2063 6f6f 6c20 7368  es c'est cool sh$
00000020: 6f77 206d 656d 206f 6e20 6661 6974 2064  ow mem on fait d$
00000030: 6520 7472 7563 2074 6572 7269 626c 6500  e truc terrible.$
00000040: 2e00 0102 0304 0506 0708 090e 0f1b 7f    ..............$
guilla_i@seattle $>
```

- Here's how it should be prototyped :

```
void        *ft_print_memory(void *addr, unsigned int size);
```

- It should return `addr`.