C Piscine

C 05

Summary: this document is the subject for the C 05 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 !!!

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

Chapter II

Foreword

Here are some lyrics extract from the `Harry Potter` saga:

```
Oh you may not think me pretty,
But don't judge on what you see,
I'll eat myself if you can find
A smarter hat than me.

You can keep your bowlers black,
Your top hats sleek and tall,
For I'm the Hogwarts Sorting Hat
And I can cap them all.

The Sorting Hat, stored in the Headmaster's Office.
There's nothing hidden in your head
The Sorting Hat can't see,
So try me on and I will tell you
Where you ought to be.

You might belong in Gryffindor,
Where dwell the brave at heart,
Their daring, nerve, and chivalry
Set Gryffindors apart;

You might belong in Hufflepuff,
Where they are just and loyal,
Those patient Hufflepuffs are true
And unafraid of toil;

Or yet in wise old Ravenclaw,
If you've a ready mind,
Where those of wit and learning,
Will always find their kind;

Or perhaps in Slytherin
You'll make your real friends,
Those cunning folks use any means
```

To achieve their ends.

So put me on! Don't be afraid!
And don't get in a flap!
You're in safe hands (though I have none)
For I'm a Thinking Cap!

Unfortunately, this subject's got nothing to do with the `Harry Potter` saga, which is too bad, because your exercises won't be done by `magic`.

Chapter III

Exercise 00 : ft__iterative__factorial

	Exercise 00
	ft__iterative__factorial

Turn-in directory : *ex00/*
Files to turn in : `ft_iterative_factorial.c`
Allowed functions : `None`

- Create an iterated function that returns a number. This number is the result of a factorial operation based on the number given as a parameter.

- If the argument is not valid the function should return 0.

- Overflows must not be handled, the function return will be undefined.

- Here's how it should be prototyped :

```
int ft_iterative_factorial(int nb);
```

Chapter IV

Exercise 01 : ft_recursive_factorial

	Exercise 01
	ft_recursive_factorial
Turn-in directory : *ex01/*	
Files to turn in : `ft_recursive_factorial.c`	
Allowed functions : `None`	

- Create a recursive function that returns the factorial of the number given as a parameter.

- If the argument is not valid the function should return 0.

- Overflows must not be handled, the function return will be undefined.

- Here's how it should be prototyped :

```
int ft_recursive_factorial(int nb);
```

Chapter V

Exercise 02 : ft_iterative_power

	Exercise 02
	ft_iterative_power
Turn-in directory : *ex02/*	
Files to turn in : `ft_iterative_power.c`	
Allowed functions : `None`	

- Create an iterated function that returns the value of a power applied to a number. An power lower than 0 returns 0. Overflows must not be handled.

- We've decided that 0 power 0 will returns 1

- Here's how it should be prototyped :

```
int ft_iterative_power(int nb, int power);
```

Chapter VI

Exercise 03 : ft__recursive__power

	Exercise 03
	ft__recursive__power
Turn-in directory : *ex03/*	
Files to turn in : `ft_recursive_power.c`	
Allowed functions : `None`	

- Create a recursive function that returns the value of a power applied to a number.

- Overflows must not be handled, the function return will be undefined.

- We've decided that 0 power 0 will returns 1

- Here's how it should be prototyped :

```
int ft_recursive_power(int nb, int power);
```

Chapter VII

Exercise 04 : ft__fibonacci

	Exercise 04
	ft__fibonacci
Turn-in directory : *ex04/*	
Files to turn in : `ft_fibonacci.c`	
Allowed functions : `None`	

- Create a function `ft_fibonacci` that returns the **n**-th element of the Fibonacci sequence, the first element being at the 0 index. We'll consider that the Fibonacci sequence starts like this: 0, 1, 1, 2.

- Overflows must not be handled, the function return will be undefined.

- Here's how it should be prototyped :

```
int ft_fibonacci(int index);
```

- Obviously, `ft_fibonacci` has to be recursive.

- If the `index` is less than 0, the function should return -1.

Chapter VIII

Exercise 05 : ft__sqrt

	Exercise 05
	ft__sqrt
Turn-in directory : *ex05/*	
Files to turn in : `ft_sqrt.c`	
Allowed functions : `None`	

- Create a function that returns the square root of a number (if it exists), or 0 if the square root is an irrational number.

- Here's how it should be prototyped :

```
int ft_sqrt(int nb);
```

Chapter IX

Exercise 06 : ft__is__prime

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

- Create a function that returns 1 if the number given as a parameter is a prime number, and 0 if it isn't.

- Here's how it should be prototyped :

```
int ft_is_prime(int nb);
```

💡 0 and 1 are not prime numbers.

Chapter X

Exercise 07 : ft_find_next_prime

	Exercise 07
	ft_find_next_prime
Turn-in directory : *ex07/*	
Files to turn in : `ft_find_next_prime.c`	
Allowed functions : `None`	

- Create a function that returns the next prime number greater or equal to the number given as argument.

- Here's how it should be prototyped :

```
int ft_find_next_prime(int nb);
```

Chapter XI

Exercise 08 : The Ten Queens

	Exercise 08
	The Ten Queens
Turn-in directory : *ex08/*	
Files to turn in : `ft_ten_queens_puzzle.c`	
Allowed functions : `write`	

- Create a function that displays all possible placements of the ten queens on a chessboard which would contain ten columns and ten lines, without them being able to reach each other in a single move, and returns the number of possibilities.

- Recursivity is required to solve this problem.

- Here's how it should be prototyped :

```
int ft_ten_queens_puzzle(void);
```

- Here's how it'll be displayed :

```
$>./a.out | cat -e
0257948136$
0258693147$
...
4605713829$
4609582731$
...
9742051863$
$>
```

- The sequence goes from left to right. The first digit represents the first Queen's position in the first column (the index starting from 0). The Nth digit represents the Nth Queen's position in the Nth column.

- The return value must be the total number of displayed solutions.